Insightful Workflow For Grid Computing

Charles Earl, Ph.D. – Principal Investigator

Contract Number DE-FG02-06ER84519

Phase I Final Technical Report

Contractor:
Stottler Henke Associates, Inc. (SHAI)
951 Mariner's Island Blvd Suite 360
San Mateo, CA 94404

Government Agency: Department of Energy

Summary

We developed a workflow adaptation and scheduling system for Grid workflow. The system currently interfaces with and uses the Karajan workflow system. We developed machine learning agents that provide the planner/scheduler with information needed to make decisions about when and how to replan. The Kubrick restructures workflow at runtime, making it unique among workflow scheduling systems. The existing Kubrick system provides a platform on which to integrate additional quality of service constraints and in which to explore the use of an ensemble of scheduling and planning algorithms. This will be the principle thrust of our Phase II work.

1 Background

1.1 Problem

Coordinating the scientific and business oriented computing tasks of a *virtual organization* – a distributed collaboration that may span organizational and national boundaries -- requires sophisticated resource management technology. A *computing Grid* is a loosely coupled collection of compute, storage, and network resources assembled by a virtual organization to support its activities; with those resources coordinated by software called *Grid middleware*. The plan that describes how those distributed services and software components interact with each other to achieve a particular computing task is termed a *Grid workflow* and is an essential Grid middleware element. Workflow management software needs to be developed to provide a way of coordinating job scheduling, file transfer, and other services involved in the execution of a Grid workflow.

Although there are many heuristics that have been developed for Grid workflow scheduling, most of the research has focused on static scheduling. That is, a mapping from an abstract description of the workflow to concrete resources is done prior to the start of the workflow's tasks, and this mapping is not altered while the workflow's tasks are processed. A Grid however is by definition a loosely couple collection of resources, some of which could be volatile: storage and compute resources are constantly being upgraded and modified; new releases of software libraries become available; Grid services have different levels of stability and responsiveness. To achieve acceptable levels of service, workflow management systems must be responsive to these varying conditions. For example, a long-running workflow might be able to increase its reliability by moving computation to a set of machines having more robust versions of critical software libraries. Another workflow might need to be altered by its user in response to unexpected intermediate results. Real time and interactive workflows in particular need tools that allow the workflow to be adapted in response to changing conditions and user requirements. Further, scheduling and planning heuristics should be employed in response to the structure of the workflow and associated Quality of Service (QoS) requirements. The ability to alter workflow in response to changing Grid resources and user requirements is essential.

Although the Grid community has delivered several Grid workflow systems, among them DAGMan [Thain03], Karajan [vonLaszewski05], Chimera [Foster02], and Gridflow[Cao03], these systems would benefit from the ability to re-plan on the fly. Workflow execution systems lack the information about high level workflow goals that would allow effective optimization. Further, they do not provide the users with both the tools needed to quickly assess how the workflow is progressing and how to change it.

Our purpose in this project is to develop a Grid workflow management and monitoring platform called Kubrick that will enable the runtime adaptation of Grid

workflow. The system will provide users an interface that provides them with information on the status of the workflow and viable options for optimization. In accordance with user provided constraints on QoS, Kubrick will be able monitor the workflow and relevant Grid resources and determine when it is appropriate to alter the workflow: either restructuring it or rescheduling the workflow using the most appropriate scheduling heuristic. We will achieve this goal using technologies in machine learning, automated planning, agent-based system design, and advance visual interface technology.

1.2 Opportunity

Grid computing is going to be a multi-billion dollar business according to Gartner studies. Hence it provides the potential for an enormous growth within the software industry. We believe that by providing a sophisticated workflow framework that integrates knowledge about the way services and resources are utilized, we will develop a much needed Grid upperware component. Hence, we will address needs that are crucial for Grid a success within the business and research community. We believe that Kubrick's runtime workflow management is one major component that will be needed by most Grid efforts.

Our technical plan is to develop a workflow system that is more adaptive and robust than current systems while at the same time allowing more expressiveness in the workflow formulation.

We see four opportunities for substantial improvement in management of Grid workflow. First is the ability to reschedule or rewrite workflow in response to changing Grid conditions. Current monitoring and prediction services do not deal effectively with temporary resource unavailability as many resources are integrated in a static fashion into the Grid. For example, suppose that a number of tasks of a workflow have been dispatched to a set of local scheduler queues. The workflow system could determine that a particular queue was having consistent delays. In response to this, the workflow manager could move the dispatched tasks to another queue, and modify bindings so that subsequent decisions in scheduling took this information into account. For example, Deelman[Deelman05] notes that the queue limitation in the Condor scheduler results in jobs being held idle in the queue. Giving the workflow system the ability to insightfully learn system dependencies and act upon that learned information would give it the power of adapting to unforeseen interactions likely to occur in a complex computing environment.

The second is the ability to take a wide range of QoS parameters into account in the scheduling of workflow. For example, the system could incorporate a QoS feature of reliability in scheduling workflow. In response to the known failure of a particular workflow branch, a workflow planning system could spawn mirror images of a task on a scheduler besides the one initially targeted, or could add debugging tasks to the branch that was known to have failed. Dong & Akl's survey of Grid scheduling algorithms [Dong06] reviews several Grid workflow scheduling heuristics (among them list heuristics, genetic algorithm approaches, and clustering algorithms). They identified just one attempt [vonLaszewski03] to involve QoS requirements into Grid workflow scheduling, and this limited to the QoS dimension of bandwidth. Further, while von Laszewki's algorithm takes into

account QoS in assigning resources to tasks (e.g. selection of host having connection guaranteeing highest bandwidth), the issue of how QoS constraints could be used to define workflow objectives is left open. A user should therefore be able to express QoS requirements at a number of levels (e.g. application, workflow task, and workflow) and have the scheduling determined by this information.

The third feature is the providing of a powerful user interface that would users to both inspect and alter the current status of the workflow as it executes.

The fourth is the ability to adjust the selection of scheduling algorithms to the structure and context of the workflow. For example, in the QoS-aware Grid scheduler [vonLaszewski03], the scheduling algorithm used was a min-min algorithm augmented with the ability to consider QoS (bandwidth). A workflow which is to be optimized with respect to reliability (probability of completion) over makespan might be better served with another scheduling algorithm. A large workflow having a large number of similar and separable branches might be better scheduled with clustering algorithms. Our system will be unique in its ability to take both performance goals and structure of the workflow into account in selecting which scheduling heuristic to apply. Workflow scheduling studies suggest that this kind of context specific approach could result in significant performance improvements over existing systems.

To develop this system, we will leverage existing Grid middleware such as the Java CogKit including the Karajan workflow engine, and the Pegasus workflow planning and scheduling engine. Additionally, we will also use AI tools developed by Stottler Henke Associates, to make the workflow systems truly insightful. Among them are:

- Machine learning tools to do job characterization and to discover dependencies between jobs and resources. Relevant here are data mining algorithms developed for detection of adverse drug reactions (SafetyMiner project) and anomaly detection (MASRR project and ChAD CVFDT data mining system).
- Distributed agents to support the event based identification of opportunities to repair and optimize workflow. Relevant is our Simbionic agent toolkit and Agent Based High Availability (ABHA) distributed agent system.
- Additional planning and scheduling tools where needed to support the online adaptation of workflow. Here, the Aurora planning and scheduling system can provide relevant technology.

As a result a powerful enhanced workflow system will exist that provides a number of critical and unique features such as: 1) user directed adaptation of Grid workflow during execution; 2) flexible delegation of workflow management tasks amongst distributed workflow engines and planners; 3) goal directed learning of workflow properties and dependencies; 4) automated integration of workflow failure diagnosis into subsequent workflow execution; and 5) the ability to selectively add robustness features to workflow branches.

1.3 Technical approach

To recap, our approach is to develop a Grid workflow adaptation service that can be used by workflow generators, schedulers, and execution engines to achieve

runtime performance not previously possible. There are two reasons we believe that we will be able to offer improved performance. First, the Kubrick service will be able to take into account a broad spectrum of Quality of Service (QoS) metrics in assessing the state of workflow and in determining whether and how to adapt the workflow. Previous studies in Grid workflow management have shown that QoS can figure prominently in improving workflow performance, but few metrics (beyond bandwidth and makespan) have been considered. Secondly, there has been no detailed exploration of how an ensemble of workflow scheduling heuristics can be used selectively taking into account workflow structure and the dynamics of the Grid environment in which the workflow executes. Further, there has been no serious examination of how best to apply rewriting of workflow graph structure at runtime. We will develop algorithms that allow situation-driven scheduling and workflow graph rewriting.

Our work will be conducted in the context of a production Grid being used to support analysis of data produced by the Relativistic Heavy Ion Collider at Brookhaven National Laboratory. This is the STAR Grid and has a number of production workflows that could potentially benefit from the technology that will be developed in our project. The Advanced Photon Source project is another potential testing bed for our service: the workflow being developed for analysis of APS data is stream oriented and thus could benefit from the ability to reschedule workflow during execution time.

To achieve the maximum benefit to DoE for our service, we will develop the capability of rewriting abstract as well as concrete workflow. The Condor DAGMan workflow execution system is used to manage concrete workflow for a number of advance workflow systems – among them Karajan and Pegasus. Thus, by providing an adaptation layer to DAGMan, potentially many workflow systems and users will be able to benefit from Kubrick’s capabilities. We will develop this capability in collaboration with the Condor development team.

During the second year of our Phase II effort, we will conduct extensive performance evaluations of the Kubrick service on the STAR Grid and with the APS project. Our commercial strategy will be to use these results as the basis for forming alliances in the electronic design automation (EDA) and financial services industries.

2 Benefits

Grid architectures are playing an increasing role in public services. Three high profile examples include the real-time sensor Grids responsible for earthquake and tsunami alerting, the real time Grid efforts that are focused on the surveillance of critical national infrastructure such as groundwater, and the massive web-service Grids maintained by companies such as Yahoo! and Google that provide what have come to be perceived as essential daily services. This effort will provide a platform that will increase the efficiency and reliability of Grids – particularly those that are highly heterogeneous and dynamic. The monitoring and alerting Grids sited are particularly compelling examples.

2.1 Relevance to federal and commercial markets

The Department of Energy supports a large number of international Grid efforts – both in the biological and physical sciences. Workflow management tools are being increasingly used to manage the complex data management and processing involved in these efforts. We are building a platform that can be used by workflow engines, static workflow schedulers, and workflow generators. We have demonstrated Kubrick's ability to work with the Karajan workflow system, which is used by many projects within Department of Energy. Our work in Phase II will provide more functionality to Karajan. Further, during Phase II we will be working with the Condor distributed computing project to develop the capability to rewrite Condor workflow elements (directed acyclic graphs or DAGs) during runtime. Since Condor is used by hundreds of distributed computing efforts throughout the world, Kubrick could be an immensely valuable service to thousands of users planet wide.

In the commercial sector, workflow is increasingly becoming critical to business function. A critical issue is dealing with varying reliabilities of legacy code wrapped by web service interfaces. The Kubrick platform's ability to restructure workflow in real time in response to quality of service constraints could provide one solution to this problem. Kubrick is particularly focused on the management of high performance scientific workflow. Applications in the electronic design automation (EDA) and financial services markets have requirements that are similar to those of the scientific workflows that Kubrick will initially support.

Each of these software markets generate on the order of $10billion in sales per year. Given its potential benefit to users in these markets, we believe it reasonable to achieve $10million revenue from a commercial offering of Kubrick over the next five years.

The commercial offering of Kubrick could be integrated into an existing workflow management system – following the same approach taken in the development being planned for the deployment within Department of Energy. It could alternatively be sold as a complete workflow solution when packaged with an open source engine such as Karajan. Revenue would come from direct sales of the Kubrick system, through consulting services, and through licensing arrangements. The advantages to the consumers of Kubrick would be in faster time to market and increased efficiency.

3 Phase I goals

To review, the main goal of the Phase I work was development of a proof of concept of insightful workflow management for the Grid: in others words a system which could adapt workflow using insights derived from monitoring and modeling the workflow and associated Grid resources. To that end, our work focused upon the particular case of how an insightful workflow system could identify, describe, and deal with changes to the performance of critical resources.

The principal deliverables and tasks – each now completed – of the Phase I efforts were as follows:

1. Knowledge engineering. With Argonne National Laboratory (ANL) staff, we will develop use cases workflow use and also refine system requirements.

Additional feedback from collaborators involved in the STAR Grid project will also be sought. This will include at least one face to face meeting with ANL staff and teleconferences as needed. ***Use cases involving workflow for the Advanced Photon Source experiment at Argonne National Laboratory were explored. The workflow adaptation technology developed during Phase I has been integrated and tested with the Karajan workflow engine.***

2. Develop Workflow Agent Architecture. This architecture, based on the Karajan event model, serves as the basis for coordinating the adaptive components of workflow. We will use the ABHA and Simbionic agent systems to develop this core infrastructure. ***Event-based monitoring agents were developed to support workflow optimization and repair. This is discussed in Sections 3.4 and 3.5.***
3. Develop Learning Agents. We will develop a collection of learning agents for characterizing workflow behavior and learning dependencies between resources and workflow elements. For example, the CVFDT algorithm could be used to develop a decision tree for identifying factors impacting the completion time of a workflow item. Rules are stored with the workflow template and are actively integrated into workflow monitoring to detect critical events. ***Learning agents were developed using he WEKA machine learning API and provide information necessary for workflow scheduling algorithms. Experiments are discussed in Section 3.7***
4. Develop Adaptation Agent. A class of agents that respond to Karajan requests to adapt workflow in response to conditions of the grid. An initial implementation would involve developing a wrapper for the Pegasus system. Additional agents might include an interface to the Aurora planner. ***We implemented runtime workflow rewriting – discussed in Section 3.5 – as well as scheduling algorithms used in Pegasus and some not implemented in that scheduler.***
5. Develop Robustness Annotations and Features for Karajan. This would include the ability to partition workflow amongst running engines and support for reliability primitives. ***Support for reliability-based workflow adaptation was implemented and results are discussed in Section 3.6***
6. Develop Initial Prototype. ***Discussed in the remainder of this Section.***
7. Test Prototype ***Discussed in the remainder of this Section.***
8. Prepare Final Report
9. Prepare Phase II Proposal.

The learning and adaptation mechanisms implemented in Phase I provide a core set of features from which to start development of the Phase II system.

4 Solution: The Kubrick Grid Workflow Management System

Kubrick is a system that provides scheduling and workflow repair capabilities. A workflow engine (e.g. Karajan) will ask Kubrick to schedule an abstract task.

Kubrick will transform this task into concrete executables and resources to use (a concrete job), or into an abstract workflow that represents how to accomplish the task. Kubrick then monitors the execution of these concrete jobs and workflows and will consider other transformations for the original task whenever the ones it has tried have not completed successfully or progressed as expected. Next we summarize these Kubrick functionalities and describe the techniques used to achieve them:

- Kubrick schedules tasks. It decides on the task resources to use and use existing Karajan's[1] facilities to actually execute the task. Scheduling algorithms use performance models (e.g., jobs' expected execution times) in order to optimize the assignment of resources. These algorithms consider the workflow structure when making these assignments.
- Kubrick transforms abstract tasks into concrete executables or abstract DAGs. It executes both with the help of Karajan. Which executable or DAG is to be chosen involves a "scheduling/optimization" decision. Kubrick chooses the executable/DAG that will run faster given the resource's state. If needed, Kubrick tries different executables/DAGs without the workflow engine (Karajan) knowing that these transformations have been done for the original workflow. Performance models (see below) are used by the service to prune the space of possible executables/DAGs to associate with a task.
- Kubrick learns performance models from historical data. These models provide estimates of execution times for an executable under a given resource configuration. These estimates are used by the scheduler when assigning resources and are used by the monitoring component (see below) to evaluate progress and consider re-scheduling if needed. Kubrick infers performance models for DAGs from those for executables.
- Kubrick monitors task/DAGs execution by comparing expected versus actual execution time and progress. It uses a CPM/PERT criterion to decide when it should re-schedule a task: the system monitors the critical path (CPM) of a workflow and uses probabilistic estimates of expected progress (PERT) to decide whether to re-schedule [Sample et. al, 2002, Lawrence 1997].

In the next sections we explain each of the implemented Kubrick components supporting the above functionality.

4.1 Workflow adaptations: abstract task representations

Kubrick takes as input **abstract tasks** and produces concrete jobs to be executed. A task is abstract in that (i) does not specify particular resources to use, (ii) does not specify the specific executable program to use, or (iii) it denotes a high-level specification whose realization accounts to **execute one of many** possible workflows describing different ways to achieve the task. Conditions (i) and (ii) are

[1] Any workflow engine shall use Kubrick. Kubrick might actually use different workflow engines in order to execute a task (for instance, transforming a task into a workflow that is executed by Condor). The phase I prototype used Karajan as the default workflow engine.

common to other systems (e.g., Pegasus [Deelman05]). Condition (iii) is a Kubrick feature inspired by Artificial Intelligence Hierarchical Task Planning Languages [Ghalab et al., 2004]: an abstract task represents “a goal” that needs to be achieved (e.g., solve a linear equation), and there could be many ways such goal can be achieved (e.g., use LU factorization, use an iterative method, BiCG). Using this representation feature, the workflow designer does not specify how to perform the task but rather which task to perform. Kubrick is in charge of deciding how to execute the task by assigning resources, deciding on the executable configurations, and by recursively transforming an abstract workflow into concrete executables. Abstract tasks let Grid users specify workflows in high level terms proper of their problem domain facilitating the use of the Grid by non technically Grid savvy users. A shared library of workflows can be used by a group of users. The technical challenge is to efficiently find a way to realize an abstract task without completely exploring the possibly huge set of different ways a task could be performed.

Example

The following workflow is a Karajan specification of a workflow to solve a linear equation Ax=B :

```
<project>
  <include file="cogkit.xml"/>
  <include file="Kubrick.xml"/>
  <solveEquation A="a.data" B="b.data"/>
</project>
```

The file “Kubrick.xml” is a Karajan module which defines the element *solveEquation* and other Kubrick convenient elements. Two methods are associated with this task: LU and BiCG. The LU method is added to Kubrick as follows (see Figure 1 for a graphical representation of the LU and BiCG workflows):

```
<addMethod name=”LU-factorization” task=”solveEquation”>
  <applicability> true </applicability>
  <rank n=1/>
  <logic>
       <task name=”split” A={A} />
       <task name=”lu-process-1” A={A}  B={B} output= L />
       <task name=”lu-process-2”/  A={A} B={B} output =U/ >
       <task name=”merge” L={L} U={U} output=X />
   </logic>
  <resourceConstraints>
         split.processors > 4
  </resourceConstraints>
```

```
</addMethod>
```

As illustrated in the declaration above, a task method will has associated the following information:

- *Applicability*: conditions that must be satisfied for Kubrick to apply the method (not implemented in phase 1)
- *Rank*: the user can "prioritize" the different methods. Kubrick will use such rank when deciding which method to use. In the prototype, Kubrick selects methods by calculating the expected completion time of the method's workflow (more later) and used the rank to favor a method when completions times are about the same for different methods.
- *Logic*: a DAG description defining the subtasks that should be executed. The DAG structure is derived from the relationship among tasks' inputs and outputs. Languages like Chimera [Foster 2002] can be used to specify the DAG. In phase II we will explore the idea of providing a graphical interface to define such graphs.
- *Resource constraints*: description of constraints used when selecting machines/clusters where to execute a task. In the example above, the split task should be executed in a machine with at least 4 processors. The language to specify constraints is similar to that of classAds used by the Condor system [Thain03].

In the next sections we will illustrate how and when the LU and BiCG workflows are used.

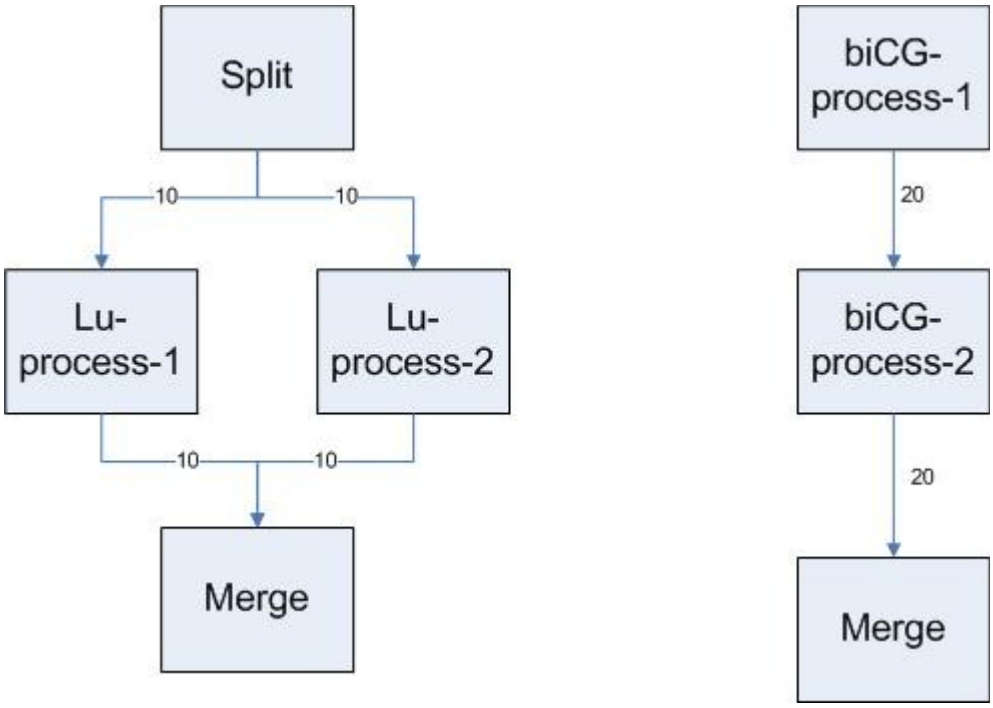


Figure 1. DAGs associated with the LU and BiCG transformations. The DAGs do not correspond to actual implementations of the LU and BiCG. The DAGs serve to illustrate the scheduling of workflows using parallel tasks (LU) versus those using sequencing. Edges values indicate the size of data communicated between tasks, values that influence the scheduler decision.

{end of example}

4.2 Kubrick interactions with the workflow engine

Kubrick provides a workflow engine independent mechanism to efficiently execute an abstract task description. Each of the possible ways a task can be realized is called a task transformation. The result of a task transformation is either a concrete workflow or an abstract workflow. Kubrick chooses which transformations to apply, applies such transformation, and executes the resulting workflow. To execute a DAG, Kubrick instantiates the DAGs for a given task, creates a Karajan workflow specification, and asks Karajan to execute the workflow. The major work of Kubrick is to keep track of all transformation and resource configurations that have been tried for a task.

A high-level description of Kubrick's algorithm to apply task transformations is as follows:

Receive an abstract task specification to execute

Create a concrete job description, possibly by defining a DAG workflow to execute the job

- The concrete job description depends on the availability of resources: the "best way to execute the job".
- To execute a DAG workflow, Kubrick uses the same workflow engine that submitted the job in the first place.

- Kubrick maintains the relationship between the original job and any tag transformations derived from it.
- To execute a basic job (e.g., submit a job to a Condor queue), the Kubrick service uses the usual workflow engine facilities to do so.

Monitor the execution of the job and re-schedule the job if QoS are violated.

Notice that Kubrick does not replace any of the functions of the workflow engine (Karajan in our case). Kubrick uses the workflow engine to carry on the execution of any task transformation and hides from the workflow engine all the transformations carried on to successfully execute an abstract task. By applying these transformations to a task, Kubrick is dynamically adapting the original task until it succeeds or it should be declared to fail.

This way, Kubrick enhances a workflow engine by allowing the specification, execution and repair of abstract tasks without the workflow engine making a difference between concrete and abstract tasks. The following example illustrates the interactions between Kubrick and the workflow engine.

Example

Consider Karajan executing the abstract workflow {A;B} (do A then B). Suppose task A can be done in any of the following 2 ways: {A_1_1; A_1_2;A_1_3} or {A_2_1; A_2_2}, where each of the abstract task A_i_j can be done using 2 actual executables. Suppose B can be done in only one way. Then, without taking into account possible resource assignments, there are 12 different ways the original workflow can be executed.

Here is one possible way the abstract workflow {A;B} could end up being executed when using Kubrick.

1. Karajan asks Kubrick to schedule task A
2. Kubrick decides to transform A into workflow {A_2_1; A_2_2}.
3. Kubrick asks Karajan to execute {A_2_1; A_2_2}

Notice that the workflow engine (Karajan) does not maintain any relationship between {A,B} and {A_2_1;A_2_2}. Kubrick maintains such relationships. Here is how the execution continues:

4. Karajan asks Kubrick to schedule task A_2_1
5. Kubrick assigns an executable to A_2_1, say A_2_1’

To execute A_2_1’, a concrete job, Kubrick relies on Karajan facilities to execute job.

6. Kubrick asks Karajan to execute job A_2_1’
7. Eventually A_2_1’ is executed successfully. Karajan informs Kubrick that A_2_1’ is done.

8. Kubrick informs Karajan that A_2_1 is done
9. Karajan continues with the execution of workflow {A_2_1; A_2_2}
10. Karajan asks Kubrick to schedule task A_2_2

Notice that Kubrick repeatedly uses Karajan to execute different workflow transformations. Technically, these are different instances of the Karajan workflow engine, each executing a particular workflow. All these Karajan instances use the same Kubrick service, which has all the information to relate these instances of the Karajan engine and the corresponding workflows associated with a task transformation.

Suppose the execution of task A_2_2 eventually fails, causing the execution of workflow {A_2_1;A_2_2} to fail. Kubrick will then **repair** the original workflow {A,B} by another transformation for A, namely {A_1_1;A_1_2;A_1_3}. Assuming the execution of this workflow and those generated for B will eventually succeed, the original workflow {A, B} will then succeed. The instance of Karajan executing such workflow will declare the workflow successful, and it will never know of any of the many other workflows that were tried when executing A.

In Section 3.6 we will show a trace illustrating the above interactions when executing the solveEquation workflow.

4.3 Kubrick scheduling component: generating concrete workflows

Scheduling refers to decide which resources assign to a task or workflow. In our case, to which clusters and queues should a job be sent. Kubrick leverages in state of the art workflow scheduling techniques to select such resources (see Section 5.3.4 for a review). These techniques consider the whole workflow structure when doing scheduling decisions rather than stand alone tasks. Since solving this problem is NP-complete, most algorithms use scheduling heuristics like min-min, min-max, sufferage, genetic algorithms, etc., to find a good but sub-optimal solution to the problem.

Kubrick provides implementation of some of these techniques (i.e., min-min, max-min, suffrage), and defines APIs to add new techniques to the service. Since more of these techniques require performance models of task execution (e.g., expected job runtime on a given cluster), Kubrick has a learning component providing such estimates. The more jobs are run through Kurbrick, the more accurate these estimates are, and the better the scheduling decisions are.

In addition to the conventional functions of a Grid scheduler, Kubrick's scheduling component is used to evaluate workflow transformation when repairing a workflow (see details below). The component is used to answer two types of questions before doing any actual scheduling: (i) the expected execution time of a given workflow/task, and (ii) provide alternative resource assignments for an already scheduled tasks.

The algorithm below illustrates the overall schema of how heuristics like min-min, max-min, suffrage, etc. are used to map workflow [Mandal et al. 2005] :

```
Scheduling algorithm:
 Input = A Workflow (DAG or unmapped tasks)
Output: map from tasks to resources

Mapping = {}
While all tasks in workflow not mapped do
     NextToMap := unmapped tasks whose parents have been scheduled
     findBestSchedule(nextToMap, mapping)
  endWhile
```

```
FindBestSchedule
   Input:  set of unmapped tasks and current mapping
   Output: extend mapping by mapping all tasks

 while all tasks not mapped do
   foreach  Task t do
        foreach Resource R do
             ECT(t,R)= rank(t,R) + EAT(R);
        endForeach

        Find minECT(t,R) over all R
       (t',R') = use heuristic to select next task to schedule (e.g., min-min, max-min)
        mapping = mapping + { (t',R') }
        update EAT(R')
endWhile

where:
   ECT(t,R)  = Estimated Completion Time of executing task t on resource R
   EAT(R) = Estimated Available Time of resource R
   Rank(t,R) = time/cost to setup task t on resource R. This includes time needed to
move the
       data produced by predecessor tasks to the resource R.

  heuristic = criteria used to decide which task should be scheduled next.
```

Example

Consider the workflow to solve a linear equation. The abstract task *solveEquation* can be solved using either LU or BiCG methods. Kubrick will calculate the estimated completion time (ECT) for each of these methods, and choose the one with the minimum completion time. For purpose of the illustration, suppose we have 4 machines (m1,m2,m3,m4) with the following estimated for task execution and data moving costs:

1. Assume a homogeneous system where the execution of a task is more or less the same in each machine.
2. Tasks split, lu-process-1, lu-process-2 and merge have EET of 12.4, 13.09, 12.33 and 16.13 (seconds).
3. Tasks BiCT-process-1 and BiCG-process-2 have EET of 20.46 and 26.25.
4. The cost of moving one unit of data (e.g., a megabyte) from machine m_i to m_j is 0.5 (if i >j), 0 (if i=j) and 1.0 (if i < j).

(The values above facilitate to verification and understanding of the trace shown below. Real workflows have thousands of tasks and Grids are heterogeneous and large).

Using a min-min heuristic, the algorithm described above will generate the following schedules, indicating that the LU transformation should be executed:

```
Execution context karajanWorkflows/solveEquation.xml
Execution context waiting for done karajanWorkflows/solveEquation.xml
Kubrick solving equation Ax=B
SHAI Min_Min_Scheduler: scheduling solve-equation

***** BiCG-Transformation schedule
=========================================
 Schedule Expected completion time = 62.850716
 Resource assignments:

// added comment: ECT = Estimated completion time, EET= Estimated Execution
Time
Resource: m2
  biCG-process-1  [ECT =20.461637, EET = 20.461637]
  biCG-process-2  [ECT =46.720356, EET = 26.258718]
  merge[ECT =62.850716 , EET = 16.130358]
---------------------------------

***** LU-Transformation schedule
=========================================
 Schedule Expected completion time = 46.867905
 Resource assignments:

Resource: m2
  merge  [ECT =46.867905 , EET = 16.130358]
```

```
  lu-process-1  [ECT =30.49587, EET = 13.090506]
---------------------------------
Resource: m3
  lu-process-2  [ECT =25.737549 , EET = 13.332184]
  split  [ECT =12.405364 , EET = 12.405364]
---------------------------------

Kubrick adapting solve-equation --> using LU-Transformation
```

The scheduler does a sensible work given our assumptions: (i) BiCG being a sequential task makes sense to be executed in one machine so there is not cost associated with moving data among machines; (ii) LU being parallel should be split among machines. Given the cost of moving data, parent tasks (e.g., split) whose children are executed in a different machine should be scheduled in a machine with higher index than their children: for instance, since the split task is scheduled in m3, lu-process-1 should not be scheduled in m4.

Once the LU-transformation schedule is chosen, Kubrick generates the following Karajan concrete workflow representing the transformation:

```
<project>
<include file="cogkit.xml"/>
<include file="Kubrick.xml"/>

//added comment: the property “kid” is a unique identifier added by Kubrick to a
task specification. The id
//  helps identify a task as part of a workflow, which in turns facilitates Kubrick’s
monitoring of a workflow
//  execution.
<execute task="split" A="a.data" B="b.data" kid="kid-15" machine="m3"/>
<parallel>
   <sequential>
        <execute task="stage-data" srchost="m3" desthost="m2" size="10"
kid="kid-18"/>
        <execute task="lu-process-1" A="a.data" B="b.data" kid="kid-16"
machine="m2"/>
  </sequential>

   <sequential>
       <execute task="lu-process-2" A="a.data" B="b.data" kid="kid-17"
machine="m3"/>
       <execute task="stage-data" srchost="m3" desthost="m2" size="10"
```

```
kid="kid-21"/>
   </sequential>
  </parallel>

 <execute task="merge" A="a.data" B="b.data" kid="kid-22" machine="m2"/>
</project>
```

Notice that the definition of the concrete workflow includes the data stage tasks entailed by the schedule. For instance, move the output of the split task from machine 3 to machine 2 where the the *lu-process-1* task is to be executed. Kubrick will ask then Karajan to execute the workflow, and start the monitoring of the workflow. In the next section we show the trace of the monitoring and re-scheduling actions taken by Kubrick.

{end of example}

4.4 Kubrick monitoring and rescheduling component

Re-scheduling in Kubrick happens whenever a task transformation is applied. More often this is the case when a previous task transformation fails. However, Kubrick proactively monitors a task/workflow execution to compare expected execution times versus actual execution times. If for instance, a task execution is taken too long as to change the critical paths of the workflow it belongs to, then Kubrick might consider re-scheduling the task, by for instance, trying other resources for the task (i.e., moving a job stuck in a queue to another queue), or considering another task transformation. The decision to re-schedule takes into account the probabilistic nature of the estimated execution times used when scheduling. In the trace below we illustrate this technique.

Example

Consider the execution of the LU concrete workflow described in the previous example. The following trace shows Kubrick monitoring of the workflow progress as it is executed by Karajan. The trace stops at the point were the execution of task *lu-process-1* starts taking longer than expected:

```
Starting monitoring for workflow with id kid-23
Execution context C:\Projects\Kubrick\karajan\cog code\cog\tmp\workflow__202.xml
Execution context waiting for done C:\Projects\Kubrick\karajan\cog
code\cog\tmp\workflow__202.xml


// added comment: workflow_202.xml is the name of the file containing the Karajan
concrete workflow
//   description for the LU-transformation
// As Kubrick detects that Karajan starts/ends the execution of a task in the
workflow the following
// information is shown: EST = Estimated Start Time (time in seconds), AST =
Actual Start time
```

```
//     ECT= Estimated Completion Time,  ACT = Actual Completion Time

SHAI Min_Min_Scheduler: scheduling split
Task split with id kid-15 started for workflow with id kid-23 [ EST= 0.0 ---  AST=0.391]
Running shai local: split
shai host completed the task --> split[execution time = 14]SUCCEED
Task split with id kid-15 ended for workflow with id kid-23 [ ECT= 12.405364 ---
ACT= 14.641]

SHAI Min_Min_Scheduler: scheduling stage-data

SHAI Min_Min_Scheduler: scheduling stage-data
Running shai local: stage-data
shai host completed the task --> stage-data[execution time = 0]SUCCEED
Running shai local: stage-data

SHAI Min_Min_Scheduler: scheduling lu-process-2
Task lu-process-2 with id kid-17 started for workflow with id kid-23 [ EST= 12.405364
---  AST=14.641]
Running shai local: lu-process-2
shai host completed the task --> stage-data[execution time = 5]SUCCEED

SHAI Min_Min_Scheduler: scheduling lu-process-1
Task lu-process-1 with id kid-16 started for workflow with id kid-23 [ EST= 17.405365
---  AST=19.656]
Running shai local: lu-process-1
shai host completed the task --> lu-process-2[execution time = 13]SUCCEED
Task lu-process-2 with id kid-17 ended for workflow with id kid-23 [ ECT= 25.737549
---  ACT= 28.625]

SHAI Min_Min_Scheduler: scheduling stage-data
Running shai local: stage-data
shai host completed the task --> stage-data[execution time = 5]SUCCEED
Monitor: abnormal task execution for task lu-process-1 [EET= 13.090506 VAR=
2.6181011 AET= 17.703]

//added comment: VAR refers to the Variance of the estimated  execution time for
lu-process-1
//      in machine 2.
```

In the trace above, the *split* operation is executed, and the *lu-process-1* and *lu-process-2* are started in parallel. The task *lu-process-2* finishes and the

lu-process-1 has not finished after 17 seconds, time at which the task execution is declared “abnormal”. A task execution is declared “abnormal” when the execution time is greater than the expected completion time plus one variance of the estimation. The declaration of a task being abnormal triggers the rescheduling policy which might or not cancel the task execution (see below). If the task is not canceled, the rescheduling is repeatedly triggered after “variance” every seconds pass without the task being finished.

To decide whether to repair a workflow given a task execution delay we compare the estimated completion time for the workflow (WEC) against a new estimation of the workflow completion given a task delay and the actual execution time of finished tasks (WECT’). We reschedule if the task delay implies a delay in the overall workflow execution (WECT’ > WECT) and such workflow delay is significant, as given by the expression

$$[\ (WECT' - WECT) / \sigma\ ] > 2$$

where σ is the standard deviation of the workflow completion time estimate, and such random variable has a normal distribution with mean WECT [Sample et. al, 2002, Lawrence 1997](See section 5.3.4 for review of PERT techniques used to estimate workflow completion times). Our re-scheduling expression then do a 90% confidence test to accept the hypothesis that a workflow execution is going to be delayed. WECT and σ are defined as

$$WECT = \Sigma\ ECT_i \quad \text{and } \sigma = \sqrt{\Sigma\ \sigma_i^2}$$

where the sum is taken over the tasks in the workflow critical path, ECT_i is the estimated completion time of the ith task in the critical path and σ_i^2 is the variance of such estimation.

The trace below show how the criteria above work in our example:

Monitor: abnormal task execution for task lu-process-1 [EET= 13.090506 VAR= 2.6181011 AET= 17.703]

Kubrick: Task delay causes delays in workflow execution [delay= 6.6214523 , previous workflow ECT = 46.867905 , new workflow ECT = 53.489357

Kubrick: checking if delay is significant ...

[delay = 6.6214523 , workflow ect standard deviation = 5.3028994 ,x=delay/sigma = 1.2486476]

Delay is not significant (x < 2.0 standard deviations)

Monitor: abnormal task execution for task lu-process-1 [EET= 13.090506 VAR= 2.6181011 AET= 19.703]

Kubrick: Task delay causes delays in workflow execution [delay= 8.621452 , previous workflow ECT = 46.867905 , new workflow ECT = 55.489357

Kubrick: checking if delay is significant ...

[delay = 8.621452 , workflow ect standard deviation = 5.3028994 ,x=delay/sigma = 1.6257998]

Delay is not significant (x < 2.0 standard deviations)

Monitor: abnormal task execution for task lu-process-1 [EET= 13.090506 VAR= 2.6181011 AET= 21.703]

Kubrick: Task delay causes delays in workflow execution [delay= 10.621452 , previous workflow ECT = 46.867905 , new workflow ECT = 57.489357

Kubrick: checking if delay is significant ...

[delay = 10.621452 , workflow ect standard deviation = 5.3028994 ,x=delay/sigma = 2.0029519]

*** Delay is significant with a 99% probability (x > 2.0 standard deviations)

Considering re-scheduling possibilities

Possible to re-schedule using transformation BiCG-Transformation, with ECT = 62.850716

Kubrick: do not re-schedule.. adaptation has greater ECT than current schedule

```
Monitor: abnormal task execution for task lu-process-1 [EET= 13.090506 VAR=
2.6181011 AET= 23.734]
Kubrick: Task delay causes delays in workflow execution [ delay= 12.652451 ,
previous workflow ECT = 46.867905 , new workflow ECT = 59.520355
Kubrick: checking if delay is significant ...
[ delay = 12.652451 , workflow ect standard deviation = 5.3028994
,x=delay/sigma = 2.3859496]
*** Delay is significant with a 99% probability ( x > 2.0 standard deviations)
 Considering re-scheduling possibilities
Possible to re-schedule using transformation BiCG-Transformation, with ECT =
62.850716
Kubrick: do not re-schedule.. adaptation has greater ECT than current schedule


Monitor: abnormal task execution for task lu-process-1 [EET= 13.090506 VAR=
2.6181011 AET= 27.765]
Kubrick: Task delay causes delays in workflow execution [ delay= 16.683456 ,
previous workflow ECT = 46.867905 , new workflow ECT = 63.55136
Kubrick: checking if delay is significant ...
[ delay = 16.683456 , workflow ect standard deviation = 5.3028994
,x=delay/sigma = 3.146101]
*** Delay is significant with a 99% probability ( x > 2.0 standard deviations
 Considering re-scheduling possibilities
Possible to re-schedule using transformation BiCG-Transformation, with ECT =
62.850716


Executing adaptation
```

Once the execution of a workflow is deemed to be delayed, Kubrick must decide on how to fix it (if possible). In the prototype we considered two options: leave things as they are or try another adaptation for the abstract task the workflow is about. In order to decide which option to take, the estimate completion time of each option is calculated and the one with minimum estimated completion time is chosen. There is another alternative that was not evaluated during phase I: re-schedule the unexecuted task of the given workflow [Sakellariou and Zhao, 2004a].

The trace below completes the execution of the original abstract task to solve the linear equation:

```
Karajan canceling current workflow ....
Execution failed:
Aborted
```

```
        :kexecute @ karajanWorkflows\Kubrick.xml, line: 67
        sys:sequential @ C:\Projects\Kubrick\karajan\cog
code\cog\tmp\workflow__202.xml, line: 7
        sys:parallel @ C:\Projects\Kubrick\karajan\cog
code\cog\tmp\workflow__202.xml, line: 6
        kernel:project @ C:\Projects\Kubrick\karajan\cog
code\cog\tmp\workflow__202.xml, line: 2
Execution context set done C:\Projects\Kubrick\karajan\cog
code\cog\tmp\workflow__202.xml



Kubrick adapting solve-equation --> using BiCG-Transformation
==============================================
 Schedule Expected completion time = 62.850716
 Resource assigments:

Resource: m2
  BiCG-process-2[ect =46.720356][eet = 26.258718]
  BiCG-process-1[ect =20.461637][eet = 20.461637]
  merge[ect =62.850716][eet = 16.130358]
---------------------------------
Starting monitoring for workflow with id kid-45
Execution context C:\Projects\Kubrick\karajan\cog code\cog\tmp\workflow__231.xml
Execution context waiting for done C:\Projects\Kubrick\karajan\cog
code\cog\tmp\workflow__231.xml
{}
SHAI Min_Min_Scheduler: scheduling biCG-process-1
Task BiCG-process-1 with id kid-40 started for workflow with id kid-45 [ EST= 0.0 ---
AST=0.313]
Running shai local: biCG-process-1


shai host completed the task --> biCG-process-1[execution time = 22]SUCCEED
Task BiCG-process-1 with id kid-40 ended for workflow with id kid-45 [ ECT=
20.461637 ---  ACT= 23.031]
```

```


SHAI Min_Min_Scheduler: scheduling stage-data
Running shai local: stage-data
shai host completed the task --> stage-data[execution time = 0]SUCCEED


SHAI Min_Min_Scheduler: scheduling biCG-process-2
Task BiCG-process-2 with id kid-41 started for workflow with id kid-45 [ EST=
20.461637 ---  AST=23.031]
```

```
Running shai local: BiCG-process-2
shai host completed the task --> BiCG-process-2[execution time = 29]SUCCEED
Task BiCG-process-2 with id kid-41 ended for workflow with id kid-45 [ ECT=
46.720356 ---  ACT= 52.42]


SHAI Min_Min_Scheduler: scheduling stage-data
Running shai local: stage-data
shai host completed the task --> stage-data[execution time = 0]SUCCEED


SHAI Min_Min_Scheduler: scheduling merge
Task merge with id kid-43 started for workflow with id kid-45 [ EST= 46.720356 ---
AST=52.436]
Running shai local: merge
shai host completed the task --> merge[execution time = 17]SUCCEED
Execution context set done C:\Projects\Kubrick\karajan\cog
code\cog\tmp\workflow__231.xml
Task merge with id kid-43 ended for workflow with id kid-45 [ ECT= 62.850716 ---
ACT= 69.638]


Kubrick: adaptation BiCG-Transformation  succeed for task solve-equation
Execution context set done karajanWorkflows/solveEquation.xml
Ending monitoring for workflow with id kid-45
Got workflow status: SUCCEED
```

{end of example}

4.5 Kubrick QoS and service performance component

So far our description of Kubrick has focused on abstract tasks that can be decomposed into a workflow. Such decomposition stops when a service or program executable needs to be determined. Even at that level of detail Kubrick will use performance metrics to decide which service or program to use. Kubrick maintains a map from an abstract service category (e.g., data-transfer) to the specific providers of such service (e.g., ftp, gridFTp). When an abstract service is referred in a workflow then Kubrick will decide which provider to use. This selection is based on QoS characteristics of the service invocation (e.g., reliability, cost, speed), constraints of the service (e.g., the provider should be installed in both the source and destination hosts), and a user defined selection criteria (e.g., choose the fastest service, choose the most reliable).

Example

A Kubrick abstract service has associated a set of user defined properties that are used to evaluate services able to perform such task. For example,

```
<define-abstract-service>
   <name>data-transfer</name>
   <args  names="srcHost, file, destHost"/>

   <properties>
     <property name="reliability" type="double"/>
     <property name="cost" type="double"/>
     <property name="speed" type="double" info="in Mbits/sec">
   </properties>


<selectionCriteria>com.stottlerhenke.Kubrick.services.dataTransfer.SelectionCriteria
</selectionCriteria>
</define-abstract-service>
```

defines the abstract service "data-transfer", with parameters *srcHost, file* and *destHost* , and associated properties *reliability*, *cost* and *speed*. These properties are characteristics that can be asked of any provider for the task. The selection criterion class encapsulates the algorithm to select among providers for the task. For example, the implementation to select the most reliable service will look as follows:

```
public Service selectService(Task task, Set<Services> services, KubrickHandle
Kubrick) {
   Double maxReliability = -1 ;//not set
   Service choosenService = null;

   For (Service service : services) {
     Double reliability = Kubrick.getTaskServiceProperty(task,service,"reliability");
     If (reliability > maxReliability) {
         maxReliability = reliability;
         choosenService = service;
     }
   }

   return choosenService;
}
```

A provider for the service is defined as follows:

```
  <service name=”ftp”>
    <abstract-service>data-transfer</abstract-service>
    <arguments names=”srcFile,srcHost,destHost” />
    <serviceHandler>com.stottlerhenke.Kubrick.services.ftp</serviceHandler>
    <constraints>
      <!-- the host and destination have ftp installed -->
      <and>
         <installed exec=”ftp” host={srcHost}>
         <installed exec=”ftp” host={destHost}>
      </and>
   </constraints>

   <property name=”number of times a service is run” type=”int”/>
    <property name=”number of times a service failed” type=”int”/>
</service>
```

defines the ftp service stating that it will be used only if “ftp” is installed in both the source host and the destination. The declaration also states “private” properties about the system that will be maintained by Kubrick. The *serviceHandler* class define the Java code that will be responsible for executing the service, updating the service “private” properties, updating the “task” public properties (i.e., reliability), and solving basic queries about the system (e.g., is ftp installed in a given machine). [2] Kubrick maintains the tuple <taskName, service, properties> storing the current value of properties for a particular service providing a particular task. The service handler for a service, informs Kubrick of changes in the properties values. For example, every time the “ftp” service is run, the ftp task handler will tell Kubrick of the new values for its properties. Kubrick uses the value of these properties to calculate the value of more complex Boolean expressions denoting service’s constraints.

{End of example}

Prototype trace

This section describes how Kubrick prototype implements the ideas above. As an example, we considered the following file transfer task, defined in Karajan using Kubrick enhanced tags (i.e., <transfer>):

```
<project>
  <include file="cogkit.xml"/>
  <include file="Kubrick.xml"/>
```

[2] A service handler is in charge of executing a given task. Since a Kubrick service definition just adds meta information about existing services, the implementation of a Kubrick task handler most often relies in the existing implementations for the underlying service.

```
 <set name="host" value="arbat.mcs.anl.gov"/>

 <for name="i">  <range from="1" to="5"/>
     <transfer srcfile="Test.data" desthost="{host}"/>
 </for>

</project>
```

The task is to move 5 times the same file “Test.data” from the current host to “arbat.mcs.anl.gov”. The Kubrick’s tag <transfer> denotes the abstract service of moving a data file to a destination. The service is “abstract” in that it does not specify the provider that should be used to do such transfer. The choice of such provider is Kubrick’s responsibility.

In the prototype Kubrick knows about 3 providers that could potentially be used to carry on the transfer task: ftp, gridFtp and gsFtp. The latest provider is not be installed in the destination host, thus Kubrick has two possible providers to use: ftp or gridFtp. Moreover, we assume that ftp is 50% reliable whereas gridFtp is 100% reliable. These are not the actual services reliability. We “injected” ftp failures for the purposes of the prototype demo.

The trace below shows Kubrick executing the workflow, selecting providers to transfer the data, and updating the service provider “reliability” estimates as the service is used. Kubrick selects the most reliable service when doing a data transfer (see below). Given the services reliabilities, the 5 data transfers are executed using the following order of services: gridFtp, ftp, gridFtp, gridFtp, ftp. GridFtp is used twice as much as ftp since gridFtp is twice as reliable as ftp. Below we discuss the service selection criteria illustrated by the trace, show simulation results when considering larger number of transfers, and compare nominal behavior of the service selection criterion (when the actual service reliability is known) versus its behavior when using reliability estimates.

```
Execution context karajanWorkflows/Test_FileTransfer.xml
Execution context waiting for done karajanWorkflows/Test_FileTransfer.xml
SHAI Min_Min_Scheduler: scheduling file-transfer
Available services for task of type file-transfer
kftp
kgt2
kgsiftp
Service kgsiftp: the following constraint is false (installed  exec=gridftp
host=arbat.mcs.anl.gov)
Service kgsiftp: the following constraint is false (installed  exec=gridftp
host=localhost)
Service kgsiftp not considered.

Using criteria  --choose most reliable service –

Reliability of kftp=1.0 weightedReliability = 0.5
 tries 1.0 total tries =2.0
Reliability of kgt2=1.0 weightedReliability = 0.5
 tries 1.0 total tries =2.0

chosen service= kgt2
GT2: submitting task ... done
```

```
// added comment:  second file to transfer
Kubrick transfering Test.data to arbat.mcs.anl.gov
SHAI Min_Min_Scheduler: scheduling file-transfer
Using criteria  --choose most reliable service --
Reliability of kftp=1.0 weightedReliability = 0.6666666666666667
 tries 1.0 total tries =3.0
Reliability of kgt2=1.0 weightedReliability = 0.33333333333333337
 tries 2.0 total tries =3.0

chosen service= kftp
Ftp: submitting task
    *** File transfer completed but too many packets lost

// added comment:  third file to transfer
Kubrick transfering Test.data to arbat.mcs.anl.gov
SHAI Min_Min_Scheduler: scheduling file-transfer
Using criteria  --choose most reliable service --
Reliability of kftp=0.5 weightedReliability = 0.25
 tries 2.0 total tries =4.0
Reliability of kgt2=1.0 weightedReliability = 0.5
 tries 2.0 total tries =4.0

chosen service= kgt2
```

```
// added comment:  fourth file to transfer
SHAI Min_Min_Scheduler: scheduling Kubrick_transfer
Running shai local: Kubrick_transfer

Using criteria  --choose most reliable service --
Reliability of kftp=0.5 weightedReliability = 0.3
 tries 2.0 total tries =5.0
Reliability of kgt2=1.0 weightedReliability = 0.4
 tries 3.0 total tries =5.0
chosen service= kgt2

// added comment:  fifth file to transfer
SHAI Min_Min_Scheduler: scheduling Kubrick_transfer
Using criteria  --choose most reliable service --
Reliability of kftp=0.5 weightedReliability = 0.33333333333333337
 tries 2.0 total tries =6.0
Reliability of kgt2=1.0 weightedReliability = 0.33333333333333337
 tries 4.0 total tries =6.0
chosen service= kftp
```

Data transfer service selection criterion.

In the prototype, Kubrick's select the most reliable service when doing a data transfer. The reliability of a software product is usually defined to be "*the probability of execution without failure for some specified interval of natural units or time*" [Musa, 1998]. The probability of successful execution is measured by repeatedly operating a system according to the selected operational profile, i.e. selecting inputs according to the frequency constraints of the profile, for the specified unit of time. The reliability is computed by measuring the percentage of those executions that terminate successfully. A reliability value is reported for each operational profile. The service reliability is then the weighted reliability over the given profiles (weights correspond to the frequency an operational profile is to be found under deployment).

For our purposes, we have three operational profiles: small, medium and large size file transfers. A data transmission is successful if (i) the file is transmitted in his totality and (ii) the packet loss of the transmission is less than a given threshold (e.g., 10%). Packet loss is defined as the fraction of packets sent for which the host does not receive an acknowledgment from the destination. This includes packets that are not received by the destination as well as acknowledgments that are lost before returning to the host. Acknowledgments that do not arrive within a

predefined round trip delay at the source are also considered lost. This way, our measure of "data transfer reliability" includes a minimum of data quality transfer and speed performance.

In the prototype Kubrick estimates the reliability of a service as it uses the services. This is done according to the formula

Rel(s) = (# of successful service's transfers) / (# of times the service is used)

where Rel(s) denotes the service reliability. The count of successful services transfers is always made with respect to the last *SSM* tries of the service. SSM represents the "service success memory". Since we filter the service tries history, the value of SSM determines how responsive Kubrick is to the actual value of the service reliability. Kubrick could also *proactively* estimate the value of the service reliability by setting up measurement tests for such purpose, like is done in the INCA project [Smallen et al, 2007].

The implementation of choosing the most reliable service criterion takes into account the frequency with which a service has been tried, so that services are "sampled" proportionally to their reliability. Let n_i denote the numbers of times service *i* has been tried (we assume $n_i > 0$), and *n* denote the number of total data transfers (i.e., $n = \Sigma\, n_i$). Let $Rel(s_i)$ denote the "reliability of service *i*" and let w_i denote the "weighted reliability of service i" defined as follows:

$$w_i = Rel(s_i) * [\, 1 - n_i/n].$$

Then the service selection criteria will select the service whose weighted reliability is maximum, breaking ties in favor of services that have been tried the less.

The number of times a service is used is always calculated with respect to the last *DTM* services tries made by Kubrick. The value of DTM represents the "*data transfer memory*" of the selection criteria. In particular, it will be the case that a service is guaranteed to be selected at least every DTM data transfers.

In our example we have two services. Let's assume service 1 has higher reliability than service 2. Then the selection criteria rule indicates that service 2 (the service with less reliability) should be selected in favor of service 1 whenever

$$w_1 < w_2$$

which reduces to

$$Rel(s_1) * [1 - (n_1/n_1+n_2)] < Rel(s_2) * [1 - n_2/n_1+n_2]$$

$$Rel(s_1) * n_2 < Rel(s_2)*n_1$$
$$n_2/n_1 < (Rel(s_2) / Rel(s_1))$$

Thus the "service sampling rate" should be the same as the rate of the services reliability. For example, suppose service 1 is twice as reliable as service 2. Then the sampling rate will be

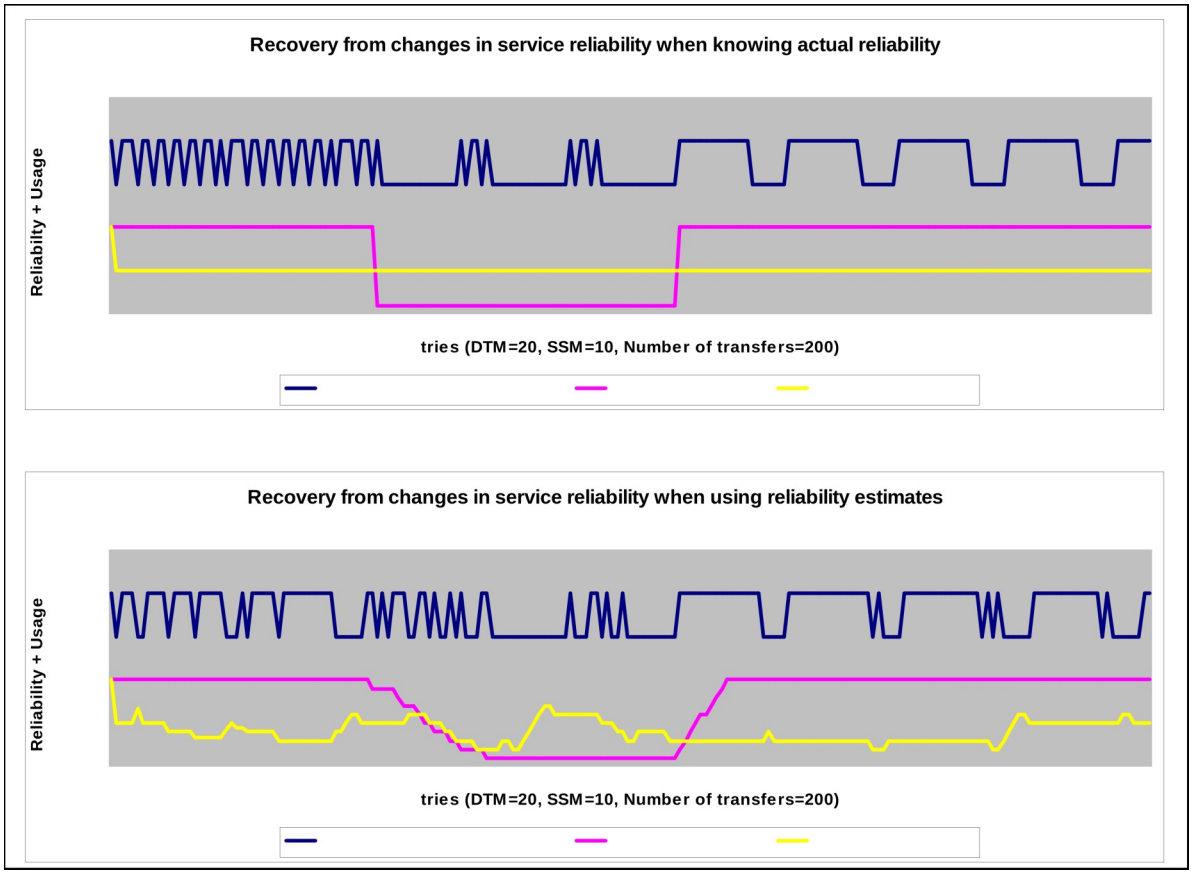
$n_1 = 2 * n_2$

so that for each 2 tries of service 1, one try of service 2 is done. The figures below show a simulation of the rule behavior when using perfect knowledge of the service's reliability and when using reliability estimates. The actual reliability of service 1 is 100%, and the actual reliability of service 2 is 50%. When using reliability estimates, the reliability of service 2 tends to be underestimated as 40%, causing Kubrick to use the service more often than in the ideal case.

Adapting to changes in reliability

The reliability of a transfer data service depends on some “hidden” variables that are out of the simple model of reliability here presented. These variables include things like workload of the service server, load of the destination host, rate of service requests, and seasonal usage of the service.

Our service selection rule adapts to changes of service reliability as illustrated in the figure below. We assume that service 1 is 100% reliable the first 50 transfers, then 10% reliable for the next 50 transfers, and finally 100% reliable for the next 100 transfers. Service 2 remains 50% reliable all the time. When knowing the actual reliability, Kubrick “immediately” starts using service 2 when service 1 reliability decays to 10%. When using reliability estimates, there is a “lag” period between when the service “true” reliability changes and Kubrick’s estimation of the reliability reflects that change (about 25 transfers). During that time Kubrick equally uses both services until the estimate for service 1 reliability is close to its actual value (10%), time at which Kubrick starts using service 2 more often.



4.6 Kubrick learning component

Kubrick uses job execution times to learn performance models that predict workflow execution times under a given resource configuration. Much work exist learning performance models for job execution [Smith et al. 1998, Jang et al, 2004, Wu and Xu, 2006]. We have used decision tree like approaches to learn these models from workload logs collected from large scale parallel systems [http://www.cs.huji.ac.il/labs/parallel/workload/logs.html]. Below we discuss some initial results.

The table below summarizes the performance of two learning algorithms using the logs for the San Diego Super Computer (SDSC) center, with 23 queues. The algorithms are M5 [Quinlan 1992, Wang & Witten, 1997] and a combination of Kmeans and M5 (we first apply Kmeans, with K=12, for the 12 months of the year, and then apply M5 to each data cluster). Although the results vary per queue, two observations can be made from the data: (i) we obtain high standard deviations on the predictions, whereas (ii) the mean error seems acceptable.

Queue	Weka km5p Mean error Standard deviation Percentage error	Kubrick Kmean + m5 Mean error Standard deviation Percentage error
0 (interactive)	677.3158920014358 110.3411699636768 12%	521.3423794902953 133.37535981577432 11%
1 (express)	552.5078970958339 271.6066280206868 52%	630.3617670528236 219.91734993432067 41%
2 (high)	2919.174511162218 333.0247814208248 44%	2231.2343609757704 297.24746226866085 45%
3 (normal)	3275.045535285894 205.96520488208276 26%	1451.1399200967567 230.54473072420268 46%
4 (low)	5039.532617391392 2020.6239638615225 69%	2572.418660686894 1131.6901212805194 71%
5 (standby)	764.5333333333334 1286.4 207%	10.666666666666666 32.0 70%
23 (legion)	222.15126050420167 262.29411764705884 437%	26.357142857142858 5.0 57%

5 References cited

[Aggarwal et al, 2005] M. Aggarwal, R.D. Kent and A. Ngom, “Genetic Algorithm Based Scheduler for Computational Grids”, In Proc. Of the 19th Annual International Symposium of High Performance Computing Systems and Applications (HPCS’05, pp. 209-215, Guelph, Ontario Canada, May 2005.

[Casanova et al., 2000] H. Casanova, A. Legrand, D. Zagorodnov and F. Berman, “Heuristics for Scheduling Parameter Sweep Applications in Grid Environments”, in Proc. O f the 9th hereogeneous Computing Workshop (HCW’00), pp. 349-363, Cancun, Mexico, May 2000.

[Cooper et al., 2004] K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z. Shi, A. Yarkhan and J. Dongarra, “New Grid Scheduling and Rescheduling Methods in the GrADS Project”, In Proc. of the 18th International Parallel and Distributed Processing Symposium (IPDPS’04), pp. 199-206, Santa Fe, New Mexico USA, April 2004.

[Braun et al., 2001] R. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen and R. Freund, “A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems”, in J. of Parallel and Distributed Computing, Vol. 61, No 6. pp 810-837, 2001.

[Cao03]Junwei Cao, Stephen A. Jarvis, Subhash Saini, and Graham R. Nudd. *Gridflow: Workflow management for grid computing*. In Proceedings of the 3st International Symposium on Cluster Computing and the Grid, page 198. IEEE Computer Society, 2003.

[Czajkowski et al., 2001] K. Czajkowshi, S. Fitzgerald, I. Foster, and C. Kesselman, “Grid Information Services for Distributed Resource Sharing”, in Proc. the 10th IEEE International Symposium on High-Performance Distribute Computing (HPDS-10), pp. 181-194, San Francisco, California, USA, August 2001.

[Deelman04] Pegasus: Mapping Scientific Workflows onto the Grid.

[Deelman05] Task Scheduling Strategies for Workflow-based Applications in Grids.

[Dong and Akl, 2006] Dong F. and Akl S.G., “Scheduling Algorithms for Grid Computing: State of the Art and Open Problems”, Technical Report No. 2006-504, School of Computing, Queens University, 2006.

[El-Rewini et al., 1994] H. El-Rewini, T. Lewis, and H. Ali, “Task Scheduling in Parallel and Distributed Systems”, ISBN: 0130992356, PTR Prentice Hall, 1994.

[Foster 2002] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao, "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation," presented at Scientific and Statistical Database Management, 2002.

[Garey and Johnson, 1979] M. Garey and D. Johnson. “Computers and Intractability: A Guide to the Theory of NP-Completeness”, W.H. Freeman and Company, New York., 1979.

[Ghalab et. al., 2004] Ghallab, M., Nau, D. and Traverso, P. 2004 “Automated Planning: Theory and Practice” Morgan Kaufman, 2004.

[Gong et al., 2002] L. Gong, X. Sun and E.F. Watson, “Performance Modeling and Prediction of Nondedicated Network Computing”, in IEEE Transaction on Computers, Vol. 51, No. 9, pp. 1041-1055, September 2002.

[He X et al, 2003] X He, X. Sun and G. Laszewski, “A QoS Guided Min-Min Heuristic for Grid Task Scheduling”, in J. of Computer Science and Technology, Special Issue on Grid Computing, Vol. 18, No. 4, pp 442-451, July 2003.

[Jang et al, 2004] S. H. Jang, X. Wu, V. Taylor, G. Mehta, K. Vahi and E. Deelman, “Using Performance Prediction to Allocate Grid Resources”, GridPhyN Technical Report 2004-25, 2004.

[Jang05] Seung-Hye Jang, Valerie Taylor, Xingfu Wu, Mieke Prajugo, Ewa Deelman, Gaurang Mehta, Karan Vahi, Performance Prediction-based versus Load-based Site Selection: Quantifying the Difference, *the 18th International Conference on Parallel and Distributed Computing Systems (PDCS-2005)*, Las Vegas, Nevada, 12 -14 September 2005

[Kiciman02] Mike Y. Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, Eric A. Brewer: Pinpoint: Problem Determination in Large, Dynamic Internet Services. DSN 2002: 595-604

[Kim and Brown, 1988] S.J. Kim and J.C. Browne. “A general approach to mapping of parallel computations upon multiprocessor architectures”. In Proceedings of International Conference on Parallel Processing, pages 1–8, 1988.

[Kim04] Jihie Kim, Marc Spraragen, Yolanda Gil. "An Intelligent Assistant for Interactive Workflow Composition ", In Proceedings of the International Conference on Intelligent User Interfaces (IUI-2004); Madeira, Portugal, 2004

[Kim and Weissman, 2004] S. Kim and J.B. Weissman, “A Genetic Algorithm Based Approach for Scheduling Decomposable Data Grid Applications”, in Proc. Of the 2004 International Conference on Parallel Processing (ICPP’04), pp. 406-413, Montreal, Canada, August 2004

[Kwok and Ahmad, 1996] Yu-Kwong Kwok and Ishfaq Ahmad. “Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors”. IEEE Transactions on Parallel and Distributed Systems, 7(5):506–521,1996.

[Laurence 1997] P. Lawrence, editor, Workflow handbook 1997, John Wiley 1997.

[Liu, 2004] Y. Liu, “Survey on Grid Scheduling”, Department of Computer Science, University of Iowa, http://www.cs.uiowa.edu/~yanliu/, April 2004.

[Mandal et al. 2005] Mandall A., Kennedy K., Koelbel C., Liu B. and Johnsson L.,”Scheduling Strategies for Mapping Application Workflows onto the Grid”, in IEEE International Symposium on High Performance Distributed Computing (HPDC’05), 2005.

[Menasce04] "Composing Web Services: A QoS View," D. Menasce, IEEE Internet Computing, Vol. 8., No. 6, November/December 2004.

[Musa 1998] Musa, John. *Software Reliability Engineering*, New York, NY, McGraw-Hill, 1998.

[Quinlan, 1992] Ross J. Quinlan: Learning with Continuous Classes. In: 5th Australian Joint Conference on Artificial Intelligence, Singapore, 343-348, 1992.

[Radulescu and van Gemund, 1999] A. Radulescu and A.J.C van Gemund, “On the Complexity of List Scheduling Algorirhtms for Distributed Memory Systems”, In Proc. Of 13th International Conference on Supercomputing, pp. 68-75, Portland, Oregon, USA, November 1999.

[Sakellariou and Zhao, 2004a] R. Sakellariou and H. Zhao, “A Low-cost Rescheduling Policy for Efficient Mapping of Workflows on Grid Systems”, in J. of Scientific Programming, Vol. 12, No. 4, pp. 253-262, 2004.

[Sakellariou and Zhao, 2004b] R. Sakellariou and H. Zhao, “A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems”, In IPDPS, 2004.

[Sample et al., 2002] N. Sample, P. Keyani and G. Wiederhold, “Scheduling under Uncertainty: Planning for the Ubiquitous Grid”, In book, Coordination Models and Languages, pp. 300-316, 2002.

[Sarkar 1999] V. Sarkar. “Partitioning and Scheduling Parallel Programs for Multiprocessors”. MIT Press, Cambridge, MA, 1989.

[Smallen et al., 2007] S. Smallen, K. Ericson, J. Hayes, C. Olschanowsky, “User-level Grid Monitoring with Inca 2”, SDSC Technical Report (SDSC TR-2007-1) and submitted to the HPDC 2007 Workshop on Grid Monitoring, June 2007.

[Smith et al. 1998] W. Smith, I. Foster and V. Taylor, “Predicting Application Run Times Using Historical Information”, Lecture Notes in Computer Science, No 1459, 1998.

[Sun and Wu, 2003] X. Sun and M. Wu, “Grid Harvest Service: A System for Long-term Application-level Tasks Scheduling”, In Proc of the 2003 International Parallel and Distributed Processing Symposium (IPDPS 203), pp. 25-32, Nice , France, April 2003.

[Thain03]Douglas Thain, Todd Tannenbaum, and Miron Livny, "Condor and the Grid", in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, *Grid Computing: Making The Global Infrastructure a Reality*, John Wiley, 2003. ISBN: 0-470-85319-0

[Topcuoglu et al., 2002] H. Topcuoglu, S. Haririr, M.Y. Wu, “Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing”, IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 3, pp. 260-274, 2002.

[Truong05] Hong-Linh Truong and Thomas Fahringer and Francesco Nerieri and et al. Performance Metrics and Ontology for Describing Performance Data of Grid Workflows, 2005.

[Trong06] Hong-Linh Truong, Robert Samborski, Thomas Fahringer, Towards a Framework for Monitoring and Analyzing QoS Metrics of Grid Services ,2nd IEEE International Conference on e-Science and Grid Computing, (c) IEEE Computer Society Press, Dec. 4- 6, 2006, Amsterdam, Netherlands.

[von Laszewski05] Gregor von Laszewski and Mike Hartegan, “Grid Workflow an Integrated Approach”, Draft Karajan Overview, at http://www-unix.mcs.anl.gov/~laszewsk/papers/vonLaszewski-workflow-draft.pdf, 2005.

[Wang & Witten, 1997] Y. Wang, I. H. Witten: Induction of model trees for predicting continuous classes. In: Poster papers of the 9th European Conference on Machine Learning, 1997.

[Wolski et al, 1999] R. Wolski, N.T. Spring, J. Hayes, “The network weather service: a distributed resource performance forecasting service for metacomputing”, J. Future Generation Comput. Systems 15, pp. 757–768, 1999.

[Wu and Sun, 2004] M. Wu and X. Sun, “Self-adaptive Task Allocation and Scheduling of Meta-tasks in Non-dedicated Heterogeneous Computing”, special issue of the International J. of High Performance Computing and Networking (IJHPCN), 2004.

[Wu and Sun, 2006] M. Wu and X. Sun, “Grid harvest service:A performance system of grid computing”, J. Parallel Distrib. Comput., Vol. 66, pp. 1322 – 1337, 2006.

[Yang and Gerasoulis, 1994] T. Yang and A. Gerasoulis. “DSC: Scheduling parallel tasks on an unbounded number of processors”. Technical Report TRCS94-12, 20, 1994.

[Young et al., 2003] L. Young, S. McGough, S. Newhouse, and J. Darlington, “Scheduling Architecture and Algorithms within the ICENI Grid Middleware”, in Proc. of UK e-Science All Hands Meeting, pp. 5-12, Nottingham, UK, September 2003.