DISCLAIMER

WORKEM: Representing and Emulating Distributed Scientific Workflow Execution State

Lavanya Ramakrishnan
Lawrence Berkeley National Lab, Berkeley, CA
LRamakrishnan@lbl.gov

Dennis Gannon
Microsoft Research, Redmond, WA
dennis.gannon@microsoft.com

Beth Plale
Indiana University Bloomington, IN
plale@cs.indiana.edu

Abstract - **Scientific workflows have become an integral part of cyberinfrastructure as their computational complexity and data sizes have grown. However, the complexity of the distributed infrastructure makes design of new workflows, determining the right management policies, debugging, testing or reproduction of errors challenging. Today, workflow engines manage the dependencies between tasks of workflows and there are tools available to wrap scientific codes. There is a need for a customizable, isolated and manageable testing container for design, evaluation and deployment of distributed workflows. To build such an environment, we need to be able to model and represent, capture and possibly reuse the execution flows within each task of a workflow that accurately captures the execution behavior. In this paper, we present the design and implementation of WORKEM, an extensible framework that can be used to represent and emulate workflow execution state. We also detail the use of the framework in two specific case studies (a) design and testing of an orchestration system (b) generation of a provenance database. Our evaluation shows that the framework has minimal overheads and can be scaled to run hundreds of workflows in short durations of time and with a high amount of parallelism.**

Keywords – Scientific Workflows, workflow emulator, distributed systems.

I. INTRODUCTION

In the last few years, workflows and workflow tools have become an integral part of cyberinfrastructure [1]. Workflow tools allow scientists to compose and manage complex computation and data in distributed environments. Each task of a workflow is a legacy scientific code and cyberinfrastructure environments today use application web services to represent and manage the execution of each task. There are tools [11],[13] available today that help scientists wrap scientific codes with a web service frontend. The workflow systems interact with grid and cloud middleware including workflow planners, schedulers, web services, provenance systems and Globus services at the sites. For example, the LEAD system [3] has around thirty services deployed that support different parts of the infrastructure. In addition, the middleware interacts with other Globus services on TeraGrid sites. The scientific exploration process varies significantly and often has requirements that need to be resolved at runtime in conjunction with the variability associated with the underlying systems. These complex requirements drive the development and research to investigate and apply innovative techniques and mechanisms to manage these environments. The complexity and cost of these systems such as LEAD often makes it hard to experiment and test new mechanisms and policies during actual workflow execution. Often there is also a need to replay a workflow execution to generate, inspect and verify the sequence of events associated with workflow execution. Thus there is a need for a contained environment where it is possible to plug-and-play various workflow execution states to study, evaluate and develop new solutions to manage these complex environments.

There are emulation or simulation frameworks associated with testing isolated components in the cyberinfrastructure [2]. However we have a limited understanding of the execution within each task of a workflow today. Each task has a complex execution flow that includes ingest of data products from distributed sources, interaction with diverse web services for data, resource and job execution management, transfer of output products to archives, other execution dependencies, etc. There are no tools available that abstract and represent the execution state of each task. The accurate abstraction of the states of each task would make development, customization, reuse and deployment of these services more generally accessible.

There are few tools available today that provide an integrated environment that enables representation of the various task states and reproduce workflow execution state. In this paper, we describe WORKEM, a framework for representing and emulating distributed scientific workflow execution state. Specifically, WORKEM provides:

- a task state model that aims to capture the intricacies in the execution flow for each task and,
- a workflow emulation container that uses the extensible task state model to allow a plug-and-play framework.

WORKEM is a web service based workflow emulation framework and serves as a benchmark platform to experiment with mechanisms and policies in a controlled environment. It is a minimalistic framework that can be configured with external event handlers to register the execution activities that might be of interest for a particular case study. WORKEM has been successfully deployed and used in two contexts for large-scale provenance data generation and testing and evaluating workflow orchestration strategies.

The rest of the paper is organized as follows. First, we detail use cases and design principles in Section II and related work in Section III. We present an overview of the emulation architecture and task state model in Section IV. We describe the system components of the emulation framework in Section V. The implementation details and a discussion of the case studies using the emulation framework are provided in Sections VI. We discuss scalability of the emulation framework in our evaluation in Section VII. Finally, we present our conclusions in Section VIII

II. BACKGROUND

Distributed cyberinfrastructure environments consist of a multitude of web services, tools and resources. The complexity of these environments makes it hard to design, test and verify operation of various components. First, we detail our diverse set of use cases that motivates the design and implementation of WORKEM.

A. USE CASES

Workflows are being widely deployed in diverse scientific domains for use with distributed grid resources. More recently tools like Hadoop[15] are being tested to see their applicability for managing parameteric scientific studies[14] in cloud environments. We discuss here some use cases that demonstrate the need for contained debug and test emulation environments.

Workflow Composition Verification. Users might want to test composed workflows for structure and logic validity before executing the workflow on distributed resources. This issue is important since users have to wait for access to resources that are often over-subscribed.

Policy design and debugging. Workflow execution in distributed environments is complex and includes coordination of services and resources at distributed supercomputing centers. It is hard to debug, design new policies or solutions in these environments without significant overhead in system personnel time and resources. Thus we need a contained environment to emulate execution while enabling plug and play model for other system components.

Provenance Data Generation. Digital data from scientific experiments are being generated every day. Tracking the origin of the data and determining the validity and quality of data is crucial for the scientific exploration process. The NSF funded Digital Data Provenance [16] project is building tools to collect and provide case-based reasoning on provenance data. There is a need for large amounts of data reflecting real-time workflow execution, both success and failures, to validate and test these tools.

Workflow orchestration. Linked Environments for Atmospheric Discovery (LEAD) is a cyberinfrastructure for mesoscale meteorology. LEAD weather prediction workflows have many configurable parameters and computing an exact result is often impossible by a given deadline [19]. However, the confidence in the result can be improved by getting QoS guarantees on at least a minimal number of workflows completing by a deadline. This requires advanced workflow orchestration methods. Experimentation and testing in the LEAD production environment is tedious given the multitude of services and components [3]. We need an isolated, but representative environment to test and demonstrate the various policy choices.

Cloud MapReduce for Scientific Workflows. Cloud computing is a recent resource model that enables on-demand access to resources based on a pay-as-you-go model. The main programming model in cloud environments today is MapReduce [18]. Hadoop[15], an open source implementation is now increasingly used to manage parallel scientific computations. The MapReduce model enables a high level definition of tasks (maps and reduces) that operate on data. The maps and reduces required for scientific workflows require a number of operations such as data management, data transfers and computations. Tools do not allow such fine-grained representation of execution state. Thus there is a need to be able to represent application execution state in this paradigm as well.

B. DESIGN GOALS

Based on our use cases, we use the following goals in the design and implementation of the emulation framework.

Simple. Managing multiple setups (for development, testing and production) is cumbersome and a major administrative burden for site administrators and developers. A *simple* emulation framework is a good tool for early development and testing of system components if a developer can easily manage it on his or her desktop. This minimizes the resource and human overhead required for managing test environments.

Accurate. It is important that the emulation framework is an accurate depiction of the sequence of events that happen in the real system. An accurate representation of the execution then enables us to develop and test policies and new components in a realistic setting and also reduces the time for moving new components from development to production.

Scalable. Scalability is inherent to most distributed systems. For large-scale research and analysis we need large amounts of data that mirror realistic execution of workflows. Hence, scalability of the emulation framework to scale to hundreds of workflows is a critical aspect of our design

Customizable. Next-generation cyberinfrastructure environment needs tools to test and iterate various scenarios and policy choices. Thus the emulation framework is designed to be

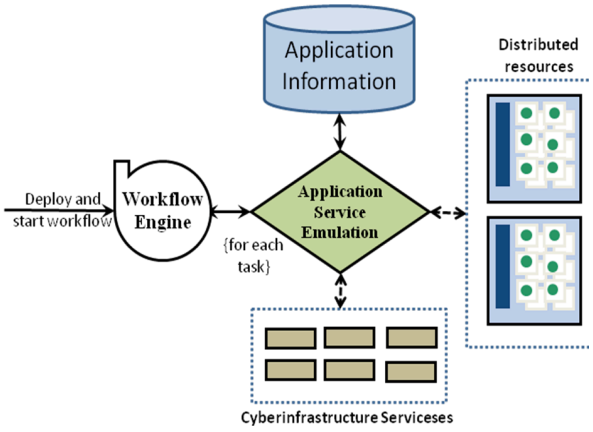

Figure 1. Workflow Emulation Architecture**.** The workflow engine invokes the application service emulation. The emulation uses the application information to determine what emulation needs to occur for the particular task. The application emulation interacts with other cyberinfastructure services or other distributed resources as required.

customizable with different parameters (e.g. failure levels) during execution allowing thorough testing and playback of various scenarios.

Extensible. There is a need for tools that application users and middleware developers can customize for specific functionality and/or what-if scenarios. WORKEM enables an extensible framework that other users and developers can use for specific purposes.

Repeatable. High variability is inherent to the nature of distributed systems. It is often a challenge to repeat the same experiment and it is almost impossible to replicate an identical sequence of events, making debugging and analysis very difficult. WORKEM is designed to be repeatable such that experiments would yield the same results enabling isolation for identifying problems and testing solutions.

Portable. Emulation frameworks are useful for testing and debugging but it is critical to be able to quickly transfer the code and results to production environments. A state based task simulation environment allows easy portability of emulation code to real environments.

III. RELATED WORK

This paper describes a representation for execution state of scientific applications and an emulation framework for large-scale workflow execution and testing. We describe some related work in this section.

Workflow engines [20] are used for representing task dependencies and controlling execution. Generic Application Factory (GFac)[11], Opal toolkit [13], etc provide tools to wrap legacy scientific application codes as web services. The wrapper handles grid security and interaction with other grid services for file transfer and job submission. However the execution logic state for each application has to be managed individually and there is no easy way to abstract out, customize and reuse policies (e.g., resource selection) or code (e.g., provenance instrumentation) across implementations.

GridSim [21] and CloudSim [23] provide a simulator framework of grid and cloud resources enabling modeling of large grid and cloud resources. Simgrid [22] is a simulation toolkit that enables the study of scheduling algorithms for distributed applications. Mumak is a Hadoop based simulator that can be used with the real job and task trackers to simulate execution on thousands of nodes for testing and debugging [15]. These simulators represent and reflect various resource level properties and behavior. However these tools do not reflect application level execution intricacies that require extensive testing.

IV. OVERVIEW

We provide an overview of our emulation architecture and describe the task state model that is a central component of our architecture in greater detail.

A. EMULATION ARCHITECTURE

Figure 1 shows the workflow emulation architecture. It consists of a workflow engine that invokes an emulation service to recreate the execution flow. In normal execution an application service is invoked during execution for each step in the workflow or directed acyclic graph (DAG). The engine invokes the emulation service in place of the application service. The application service emulation follows a state based execution flow that captures different stages of task execution including data transfer, computation, post-processing, etc.

The specific information for a task such as execution time and data transfer details are retrieved from a local database during execution. The emulation service may also interact with external systems like a grid emulator [2] (that emulates application running on different resource provider sites) or other

cyberinfastructure services [3] for specific functions (detailed in Section VI).

B. TASK STATE MODEL

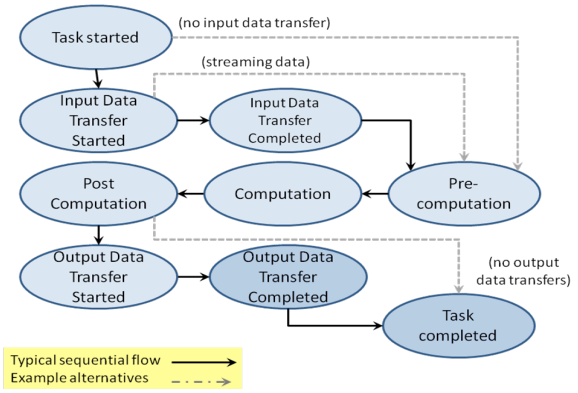

Figure 2. Application Service Emulation Execution Flow. A failure in any of the states would result in the application flow to result in the failure state (not shown in the figure).

The state diagram in Figure 2 depicts the application service execution flow. The flow captures the various states that a scientific application might traverse during its execution. The knowledge of the states a particular application might traverse is captured in the information model of the emulator (Section V.E) and stored in a database before execution. The state diagram shows the nine states defined in our system. The arrows in the diagram show the typical sequential flow between the different states for scientific application codes. However, applications might have specific characteristics that skip one or more states. For example, if there was no input data to be transferred after the task-started state, the pre-computing state would be invoked. Similarly an application with streaming input data might not have the input data transfer completed state.

The service is configured to invoke the appropriate handler. The service might use multiple event handlers, e.g., for provenance generation and scheduling. The states are described below:

Task Started. This marks the start of an activity or task. This is the state that marks various initialization activities for task execution. For example, handlers might invoke a workflow planning component to get resource information and then retrieve the task information (e.g. list of input data files) based on the resource information.

Input Data Transfer Started and Input Data Transfer Completed. These states are used to capture the data transfers that might be required for the particular task at hand. These steps will be repeated for each data product that might be required by the computation. These steps can also be skipped for special circumstances – e.g. if there are no input data transfers that are required or the input data transfer completed state might not be relevant for streaming input data.

Pre-computation. In this state typically pre-computation steps are invoked. For example, input data products might be registered with a meta-data catalog or specific resource based information for the computation might be retrieved

Computation. In this state the emulation of the computation stage might be done. This could be a NOP (no operation) for the application execution time or an external grid emulator might be invoked.

Post computation. This represents the post-computation activities that might include data product registration

Output Data Transfer Started and Output Data Transfer Completed. These states are used to capture the data transfers that might be required for the output data generated by the emulated task. These steps will be repeated for each data product that might be required by the computation. These steps can also be skipped if data transfers are not required for this task.

Task Complete. This is the final state in a normal execution flow and the task result is sent back to the workflow engine, which then uses that to invoke the next task in the DAG.

Failure. If a failure occurs in any of the above states, the execution flow transitions to a failure state. The failure state (not shown in the diagram) might be defined to perform remedial measures that might enable normal execution flow to be resumed.

This generic state model enables us to accurately capture the execution flow resulting in a richer emulation framework. The state handler enables us to implement different handlers for specific policy choices and thus provides a customizable and extensible framework for application execution flow. In this paper, we describe how this state model is used to implement a framework for representing and emulating workflow execution state. However, the application state model has wider applicability and can be used as a model to represent execution flow in building application services that operate in grid and cloud environments. In addition as additional performance data on different stages of task execution become available, this base model can be expanded for more statistical characterization.

V. SYSTEM COMPONENTS

We describe the various system components in the container framework that enables the emulation of distributed workflows.

A. WORKFLOW ENGINE

We use an existing workflow engine - Apache ODE [4] which supports the WS-BPEL [5] specification, to support workflow execution. Using an existing workflow engine enables reuse of workflow descriptions and support of different workflow patterns as available in the WS-BPEL specifications.

Apache ODE operates as a web application in containers (such as Apache Tomcat) and supports Axis and JBI based communication layers. We configure ODE to use the MySQL database to store transaction information. The workflow documents (the BPEL file, the WSDL file and the ODE deployment descriptor) are deployed in a running instance of the workflow engine. These documents are identical to the original documents used in production environments except for one change. The service invocations are redirected to use the emulator's proxy service instead of the original service locations. The proxy service simulates the task execution and returns a result back to the engine that then triggers the other activities.

Apache ODE, designed primarily for business workflows, supports web-service based invocations that are run for short durations of time. Thus, ODE by default uses synchronous communication between the engine and the services. However scientific workflows can take anywhere from a few minutes to hours during execution [19]. Thus, in addition to the vanilla ODE version, we also support a modified ODE engine (originally used in the LEAD project). The asynchronous version also enables us to achieve higher levels of scalability. The patches support asynchronous communication that facilitates long invocations, and provenance notifications to track invocations from the engine.

B. APPLICATION EMULATION

The application service emulation or the proxy service receives an invocation for each task in the workflow. The service is responsible for parsing the incoming input SOAP message and generating the appropriate output messages. Once the proxy service receives the input message, it invokes the task state simulation execution. Once the task simulation finishes execution, the proxy service generates the correct output types using the web service descriptions (i.e, WSDL files). The output is initialized with dummy data and control of workflow execution is transferred back to the workflow engine. The proxy service supports both synchronous and asynchronous responses back to the workflow engine. The proxy service also manages the emulation clock that is detailed in Section D.

Our implementation works with BPEL and web services at the moment. However the application emulation design is general enough to work with other scenarios. The emulation service could be invoked by Hadoop to emulate map and reduce tasks. In addition, the emulator is powerful enough to handle scenarios where resources might be pre-selected for execution.

C. INPUT MESSAGES

Workflow tools need the ability to differentiate between different workflow instances. Workflow systems provide a methodology to identify and differentiate between different instances. In the LEAD production system, instance identification numbers are generated by the meta-workflow management tool, Xbaya[7]. Similarly in our emulation framework, the workflow engine client generates the instance identification. We use the LEAD header specification to pass this and other parameters through the workflow execution process. The LEAD header enables us to pass workflow instance and node identification.

D. CLOCK

The proxy service manages the clock for the simulation. It uses real system time or a trace file to determine the start time for the first task in the workflow at the first invocation time. This time is then handed to the task simulation execution, which generates appropriate time stamps for different task completion stages. This time is then used as the start time for the next task in the workflow.

E. INFORMATION MODEL

The emulator needs information about the application and its typical behavior to provide a realistic environment. We use the following sources of information for the same.

State Transitions. The emulator maintains a state transition database for each task in the workflow. When the emulator is invoked for a task, it invokes the specified handler that handles execution of each of the states.

Performance model. For each task, we also maintain timing information for each state. This performance information, captured from real workflow executions is used to track the task and workflow execution times.

Data transfer. In addition to the running time of the application, it is important to consider the data transfer times between the tasks in the workflow. The data sizes for the workflow inputs, intermediate data products and workflow outputs are maintained. The bandwidth is then used to determine the data transfer time. We use a Pareto distribution to model bandwidth in these systems in conjunction with data captured on real systems [8].

Traces. An important consideration for emulation is to realistically emulate submission time of workflows. In addition to the emulator clock that is used to track submission time of workflows, we enable a trace mode in the emulator. In the trace mode, the emulator is initialized with start times provided in a trace file for a set of workflows. On arrival of a workflow, the initialization table queried to determine submission time and execution time is managed relative to the specified time. This mode is useful for situations where real traces might need to be replayed for parameter studies.

F. AVAILABILITY AND FAILURE MODEL

Availability variations and failures are inherent to distributed systems. An emulator has to capture this behavior for realistic representation of workflow execution. We model three availability properties in our emulator model that can be optionally enabled as required by the application.

Performance Variance Generator. We provide a performance variance generator that uses a normal distribution to model application execution times. Previous studies show that variation in performance of application execution follows the normal distribution [9, 10].

Failure Model. We provide a configurable failure model. The user of the emulator can set the failure percentage in each of the above states of the task handler i.e., there is a 50% probability that a task might fail in the computation stage. Similarly communication failures can be modeled as failure rates in the data transfer states. A failure in a particular state results in the workflow to fail. During execution, these specified values are used in a uniform distribution to model if a particular invocation must fail.

Information Loss. Finally in real environments there are cases where a task will execute but information about the execution, such as a notification is lost due to communication failures. We model this behavior by allowing a user to specify the percentage of dropped messages in the system. Thus with the performance variation plug-in and the failure model, applications can realistically emulate variability in these environments.

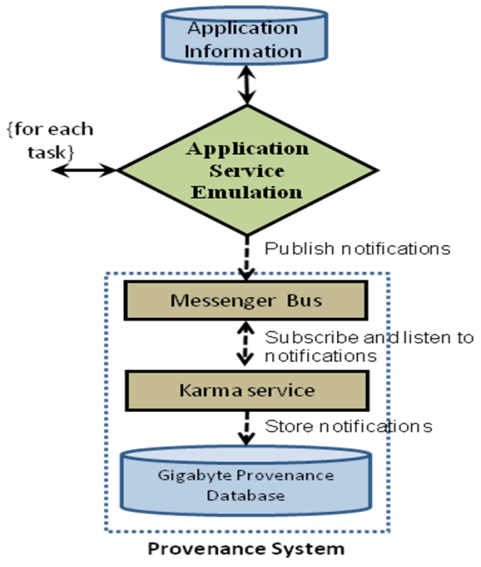

Figure 3. Gigabyte Provenance database generation using the workflow emulator

VI. IMPLEMENTATION

Description of workflow	Structure of workflow
LEAD Weather Prediction. weather forecasting initialized with terrain and observational data	Small computational steps that operate on large sized data sets followed by a computationally intensive weather model
Motif. Domain analysis of genome sized collections of input sequences	A large sized workflow with about 140 tasks where sequences are processed in parallel and then fed to the motif analysis code
SCOOP. Storm surge modeling ensemble	A number of parallel model execution followed by a post processing that aggregates the results
NCFS. Flood-plain modeling of the North Carolina cost	Long running workflow that has use a large number of processors and takes about a day of execution time.
Animation. Frame processing for animation	A pre-processing step that launches parallel frame processing followed by a post-processing step
Gene2Life. Molecular Biology analysis of DNA sequences	Two parallel execution paths that are triggered with the same input.

Table 1. Examples of workflows deployed in the current implementation of the emulator

The emulator framework has been used in the following context: a) gigabyte provenance database generation and, b) workflow orchestration across grid and cloud systems. These scenarios illustrate how the emulator can be used in design and development of workflows. In addition, the ability to compose and execute experiments in a controlled framework helps workflow composers and designers to debug workflow structures and execution characteristics.

In this section, we present the workflow examples we have deployed in our system and the implementation details of the above use cases. We also discuss the design of the emulation framework that enables easy extension of the framework for a) adding new workflows b) customized task simulation.

A. WORKFLOW EXAMPLES

In previous work, we conducted a survey of scientific workflows from different domains[6]. We have modeled a subset of these workflows using Xbaya, a workflow composition tool [7]. Xbaya generates WS-BPEL documents that can then be deployed in the workflow engine used with the

emulator. Table 1 provides a description of the workflows deployed in our implementation. The workflows are from diverse scientific domains and have different levels of parallelism and length or duration of the workflow.

B. PROVENANCE DATABASE

In the provenance database use case we implement an emulator handler to publish provenance messages. Figure 3 shows the interaction of the application service emulation with the provenance system. Karma [24] is a tool that collects and manages provenance data. Karma has a modular architecture that supports multiple types of data sources for provenance data. Karma can listen to notifications on a messenger bus or receive messages synchronously and process the notifications to determine provenance information.

The application service emulation supports both modes of operation. The Karma2.0 handler in the emulator publishes provenance messages to a WS-messenger bus [11]. The Karma3.0 handler in the emulator directly pushes the provenance information to the Karma service. For each state, a message is published that details the activity information, e.g., service invocations, data transfers, and computational messages. The Karma service listens to all relevant messages and populates a database. The goal of this plug-in is to create a large database representing workflow execution messages. Using the failure model in the emulator, we generate realistic representations of the provenance data. The emulator enables us to create a controlled execution environment and use a minimal set of components to generate data that then is useful in design and testing of tools related to provenance collection and analysis [12].

C. WORKFLOW ORCHESTRATION

Figure 4 shows the implementation details of the handler that interacts with the orchestration system. In this scenario, the task service interacts with a limited subset of components and helps in the study of workflow orchestration policies. In the first step, the proxy service interacts with the workflow planner to determine the resources on which the task must execute. The orchestration components then interact with the Virtual Grid Execution System (vgES) which is a pluggable component for the emulator. The vgES provides a uniform execution interface to query and manage executions on grid and cloud resources. The execution system relies on middleware such as Globus for job submission and data transfer.

Next, the task is launched for execution on the determined resources. The task simulation then waits for the job completion. This particular implementation has resources at a total of seven sites that include grid and cloud deployments. More details of the orchestration infrastructure can be found in previous work [17].

This integrated infrastructure is an example of managing workflow environments for demonstration in conjunction with real resource sites. The emulation environment enables us to experiment with specific policies in a hybrid environment of workflow emulation with real execution.

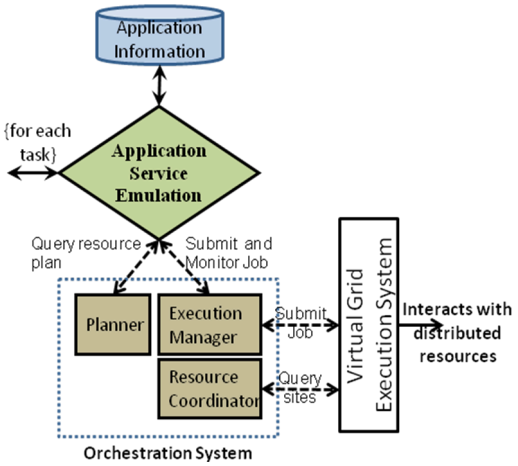

Figure 4. Orchestration system using the workflow emulator.

D. ADDING A WORKFLOW

In our implementation, we compose and deploy about ten workflows from different domains with varying complexity. However, we anticipate that users will want to add new workflows. Any pre-composed workflow that is WS-BPEL[5] compliant can be directly deployed in the emulation framework. In addition, existing tools can be used for composition of new workflows. For our setup, we use Xbaya[6] to compose workflows and export the required documents.

```
public interface TaskSimulator{
   public void Task_started(long curtime)
   public void InputDataXfer_started(long curtime)
   public void InputDataXfer_completed(long curtime)
   public void Pre_computation(long curtime)
   public void Computation(long curtime)
   public void Post_computation(long curtime)
   public void OutputDataXfer_started(long curtime)
   public void OutputDataXfer_completed(long curtime)
   public void Task_completed(long curtime) throws
}
```

Table 2. Task State Execution Handler Interface.

E. CUSTOMIZED TASK SIMULATION

The basic interface for the task simulation defines the functions for the states (Section IV.B) that will be

executed for each task. A customized task simulation can overload one or more of the functions to emulate different aspects of task execution. Table 2 shows the interface for the application service emulation. There is a function for each application state. For a specific use case, a developer can implement a handler by providing functionality to be performed for each of the functions. In addition various types of handlers can be extended to get hybrid behavior. For example, the failure generation handler might also optionally invoke the provenance generation handler thus generating failures and appropriate notifications as part of the emulation.

VII. EVALUATION

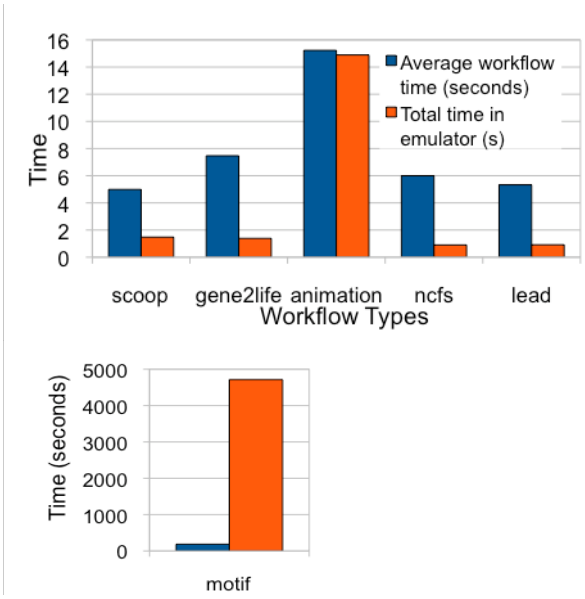

Figure 5. Workflow turnaround time for different workflows and the total emulation time performed at the application service emulation.

We perform a series of experiments to understand the characteristics and performance of the emulator framework in reproducing workflow execution. Specifically we measure system overheads, the time scaling factor and system scalability.

Workloads: Our experimental workload consists of the workflows described in the earlier section. The workflows vary in their degree of parallelism (in hundreds) and the duration or length of the workflow (from minutes to hours) enabling us to perform extensive testing on the emulator. For scalability testing and the time scale representation experiments we select the lead workflow since it is representative of the cyberinfrastructure workflows and additionally we have large amounts of data of lead execution on real systems for a meaningful comparison.

Machine configuration and Software setup: We run the experiments on a machine with a Core 2 Duo processor running at 2.4 GHz, 2GB RAM running Ubuntu 9.04 (Jaunty Jackalope). The machine hosts the modified Apache ODE 1.1 workflow engine supported by a MySQL 5.0 server for its state. In addition, the application emulation service runs with a simple TestSimulator that logs messages for each of the task's states. This is a very simple implementation of the TaskSimulator enabling us to understand the overheads of the systems without external influences.

A. EMULATION TIME

First, we measure the turnaround time from the emulator for execution of different workflows and the total time spent in the emulator for emulating the tasks of the workflow. For the motif workflow the results are averaged across a total of 25 workflows (5 workflows in 5 concurrent threads launched periodically). For other workflow types, the results are averaged over 100 workflows (10 workflows in 10 concurrent threads launched periodically). Figure 5 shows the workflow turnaround time from the emulation and the total emulation time spent at the task service level. We see that the scoop, gene2life, ncfs and lead workflows take between 4 to 8 seconds per workflow and since the number of tasks and the parallelism of these workflows is small, the time spent in the task emulation is small (~1 to 2 seconds). The animation workflow that has a higher degree of parallelism (~ 20) takes about 13 seconds of emulation time per workflow. The motif workflow is one of our largest workflows with 138 tasks out of which 135 run in parallel. The total workflow turnaround time is about 3 minutes and the total effective time spent is about 78 minutes. The motif workflow would take hours to run in a current grid system and the emulator enables us to test the workflow at a larger scale with simplicity. This gives us an understanding of the level of parallelism that the workflow engine as well as the task service emulation is able to handle.

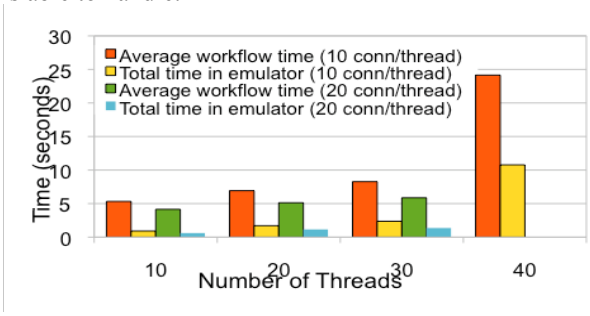

Figure 6. Understanding the scalability and the impact on the LEAD workflow turnaround time and time spent at the emulator level.

B. SCALABILITY

Figure 6 shows the scalability impact on the LEAD workflow. In this case we increase the number of concurrent threads from 10 to 40 and repeat the experiment with 10 workflows per thread and 20

workflows per thread and measure turnaround time per workflow. There is an initial startup cost associated with the first few workflows and that gets averaged out as we run more workflows resulting in lesser average turnaround time when we run more workflows per thread. There is no significant difference in time spent in the application service emulation. The workflow emulation takes longer if there are 40 simultaneous clients taking around 25 seconds. However this is a small time for emulation considering some of these workflows would run for hours in a real environment. Running multiple application emulation services and/or replicating the workflow engine can achieve further scalability.

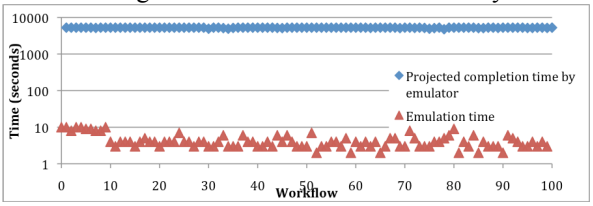

Figure 7. Comparison of the timestamps generated by the simulator for workflow execution times with the actual time taken to emulate the workflow. NOTE: The Y-axis is in log scale

C. TIME SCALE

The workflow emulator supports a trace mode where information about start times can be initialized and used to generate appropriate workflow completion time. We initialize the emulator with start times for the workflows from a real trace from the LEAD production system. Figure 7 shows the projected completion time generated by the emulation (which is the order of 5400 seconds) whereas the actual time taken for the workflow execution in the given environment is between 5 to 10 seconds. The emulation is used for 100 workflows run over a period of a month in the system and it takes about 15 minutes to emulate the same set in our framework.

D. FAILURE GENERATION

The emulator enables us to configure a task level failure model associating the failure levels for each state. We configure the "Computation" state to have different task failure levels from 10% to 50% and measure the percentage of tasks that fail and the percentage of workflows that fail for 100 (in 10 threads) and 300 (in 15 threads) workflows. As we see in Figure 8(a) the failure task percentages closely follow what we configured at the emulator. In Figure 8 (b) we see the corresponding percentage of workflows that fail for each of the failure levels. As we reach close to 40% task failure level greater than 90% of the workflows in our set fail. Thus even small task failure levels can result in a large number of workflows failing.

VIII. CONCLUSION

In this paper, we present WORKEM, a framework that enables the representation and emulation of workflow execution state. WORKEM includes a powerful and flexible task state model that allows users to accurately model different aspects of application behavior in distributed systems. Our evaluation shows that the emulator framework easily scales to hundreds of workflows with minimal overhead. The flexibility in the design of the emulation container makes it a strong, low-cost and low-overhead framework for testing and evaluating workflow execution in grid and cloud environments.

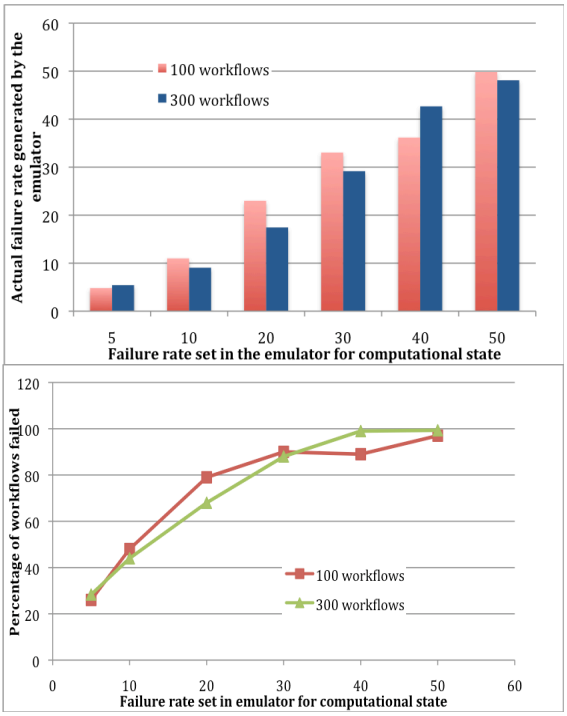

Figure 8. Understanding the failure model in the emulator (a) shows the percentage of tasks that failed at different failure levels b) percentage of workflows that failed at different failure levels. The tests were conducted for 100 and 300 workflows

IX. ACKNOWLEDGMENTS

The emulator work is funded in part by the National Science Foundation under grant OCI-0721674. An early version of the emulator was developed under the Linked Environments for Atmospheric Discovery funded by National Science Foundation under Cooperative Agreement ATM-0331480 (IU). This work was supported by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. The authors thank Ai Zhang, Sribabu Doddapaneni, You-Wei Cheah, Thilina Gunarathne, Chathura Herath, Suresh Marru, Yogesh Simmhan, Girish Subramanian and Bin Cao for discussion and

implementation details of the provenance plugins, Apache ODE and Xbaya.

REFERENCES

[1] D. Atkins, A Report from the U.S. National Science Foundation Blue Ribbon Panel on Cyberinfrastructure. *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002: p. 16.

[2] L. Ramakrishnan et al., Toward a Doctrine of Containment: Grid Hosting with Adaptive Resource Control. *Proceedings of the ACM/ IEEE Conf. on High Performance Computing, Networking, Storage and Analysis*, 2006.

[3] K.K. Droegemeier et. al, Service-Oriented Environments for Dynamically Interacting with Mesoscale Weather. *Computing in Science and Engg.*, 2005. 7(6): p. 12-29.

[4] Apache ODE - http://ode.apache.org/.

[5] Web Services Business Process Execution Language Ver. 2.0, *Public review draft*. 2006.

[6] S. Shirasuna, A Dynamic Scientific Workflow System for Web Services Architecture. *Department of Computer Science, Indiana University*, 2007.

[7] J.Schopf and F. Berman, Performance Prediction in Production Environments, *12th. International Parallel Processing Symposium*, 1998.

[8] L. Ramakrishnan and D.A. Reed, Performability Modeling for Scheduling and Fault Tolerance Strategies for Grid Workflows. *ACM/IEEE International Symposium on High Performance Distributed Computing*, 2008.

[9] W. Kramer and C. Ryan, Performance Variability of Highly Parallel Architectures, *Computational Science — ICCS*, 2003.

[10] Y. Huang, et al., WS-Messenger: A Web Services-Based Messaging System for Service-Oriented Grid Computing. *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, 2006: p. 166-173.

[11] D. Leake and K.-M. Joseph, Towards Case-Based Support for e-Science Workflow Generation by Mining Provenance, *in Proc of the 9th European conference on Advances in Case-Based Reasoning*. 2008.

[12] G. Kandaswamy and D. Gannon. A Mechanism for Creating Scientific Application Services On-demand from Workflows. *Proceedings of the 2006 International Conference Workshops on Parallel Processing*, pages 25–32, 2006.

[13] S. Krishnan et al. Design and Evaluation of Opal2: A Toolkit for Scientific Software as a Service. *IEEE Congress on Services (SERVICES-1 2009)*, July, 2009.

[14] J. Wilkening et al. "Using Clouds for Metagenomics: A Case Study," Preprint ANL/MCS-P1665-0809, Aug 2009.

[15] Apache Hadoop http://hadoop.apache.org/

[16] Digital Data Provenance Project Website http://www.dataandsearch.org/provenance/

[17] L. Ramakrishnan and et al. VGrADS: Enabling e-Science Workflows on Grids and Clouds with Fault Tolerance, *The International Conference for High Performance Computing, Networking, Storage and Analysis*, 2009.

[18] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proceedings of OSDI '04: 6th Symposium on Operating System Design and Implemention*, San Francisco, CA, Dec. 2004.

[19] L. Ramakrishnan, D. Gannon, "A Survey of Distributed Workflow Characteristics and Resource Requirements", Technical Report TR671, *Department of Computer Science, Indiana University,Bloomington*, Sept 2008,

[20] I.J. Taylor et al., eds. Workflows for e-Science: Scientific Workflows for Grids, Springer-Verlag, 2006

[21] R. Buyya and M. Murshed, GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing, *The J. of Concurrency and Computation: Practice and Experience*, Volume 14, Issue 13-15, Wiley Press, Nov.-Dec., 2002.

[22] H. Casanova, Simgrid: A toolkit for the simulation of application scheduling. *Proc. 1st IEEE/ACM International Symposium on Cluster Computing and the Grid* May, 2001.

[23] R. N. Calheiros, R. Ranjan, C. A. F. De Rose, and R. Buyya, CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services, Tech. Report, GRIDS-TR-2009-1, *The University of Melbourne, Australia*, March, 2009.

[24] Bin Cao, Beth Plale, Girish Subramanian, Ed Robertson, Yogesh Simmhan, "Provenance Information Model of Karma Version 3," Services, *IEEE Congress* on, pp. 348-351, 2009 Congress on Services - I, 2009.