11.1 HOW THE NCSA/LEAD WORKFLOW BROKER MANAGES COMPLEX WORKFLOWS

Jay C. Alameda[1], Albert L. Rossi[1], Shawn D. Hampton[1], Brian F. Jewett[2], Robert B. Wilhelmson[1,2],
[1]National Center for Supercomputing Applications (NCSA), University of Illinois at Urbana-Champaign
[2]Atmospheric Sciences Department, University of Illinois at Urbana-Champaign

1. Supporting the very wide workflow

Over the course of approximately six years of work directed at supporting high-performance applications in meteorology, chemical engineering, astronomy and even the analysis of financial markets, we have developed and tested a number of variations on what might be referred to freely as a "service-stack" architecture encompassing, on one end, the user entry point, and on the other, the high-performance-resource-resident execution of application code (of which WRF is one well-tested example). While it is true that there now exist a myriad of scientific workflow systems, some of them exclusively client-side, some distributed, having many features and goals in common with ours, our work has perhaps distinguished itself by its ever-increasing focus on the management of very wide workflows, where the number of nodes (i.e., jobs) comprising a single submission is in the hundreds to many thousands, with these to be scheduled variously and efficiently across an array of resources as part of a single scientific experimental analysis. In particular, we have sought to support the "ensemble" or parameterized analysis: that is, the exploration of a large space of input permutations. This capability is currently used for idealized- and real-data parameter studies with WRF.

With a view to both configurability by the user and scalability, we have recently added two important features to our system: first, the elaboration of such workflows through a condensed XML description to be expanded inside our workflow engine ("Parametric Workflow Engine" or PWE), and second, a means of circumventing the barrier raised both by batch systems (on the submission window: it was our experience that to submit 100 members to the queue alone took upwards of 15 minutes), and the number of separate file movement and execution calls necessary to stage and launch multiple jobs. Inspired by the concept of the glide-in pioneered by Condor, we have devised a new application container which works in conjunction with a common data-store (a LINDA-like tuple-space) in order to explode a single batch submission into *k*-parameterized members, each doing a different version of work on some partition of the total processors allocated to it. This is a significant improvement in reliability and efficiency over submitting large numbers of jobs to production batch systems.

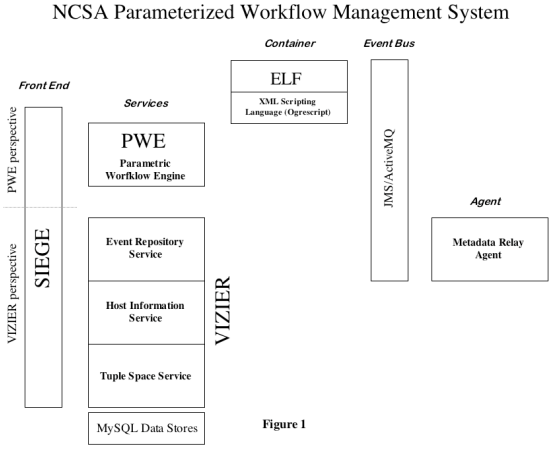

Figure 1: Infrastructure Architecture

2. Summary of the Components

2.1 *PWE (Parametric Workflow Engine)*

The principal player in this system is PWE, the logic engine which manages the state of the workflow by determining when nodes (e.g. WRF jobs) can run, configuring and launching them when they are ready, storing any output passed back to it from ELF and making this available to successive nodes. The service is directly queriable, so that the state of any given workflow can be inspected at any time; its API also allows for the manual cancellation and restarting of individual nodes or of the entire workflow.

PWE can be extended with any number of modules supporting payload type (currently just ELF running Ogrescript), submission type (currently local exec on the host where the service is located, or remote via GSISSH or SSH) or platform protocol (currently interactive launches to LINUX/UNIX machines and batch submissions to LSF or PBS; we will soon be adding LoadLeveler).

**Corresponding author address*: Jay C. Alameda, National Center for Supercomputing Applications, University of Illinois, 1205 W. Clark St., Urbana, IL 61801; e-mail: jalameda@ncsa.uiuc.edu

When a workflow contains a parametric node[*], but this is to be run interactively or on a machine where it is not feasible to do a glide-in, the individual members are submitted as separate jobs. Where glide-in through a batch system is supported, however, these additional steps take place:

1. The member-specific parameters are wrapped into an (XML) configuration object, and these are all stored in a common data service (a tuple-space);
2. A single glide-in container is submitted through batch on their behalf;
3. When this container begins to run, it takes the available configurations from the tuple-space, merges them with whatever configuration is common to all these members, partitions the total available processors/cores allocated to it among theses members, and launches them through a mechanism appropriate to the machine architecture (for instance, a simple exec on an SMP machine, or an SSH to a back-end node on a distributed-memory machine).

2.2 *VIZIER (Data/Information Services)*

This trio of services provides information to PWE vital to the scheduling and configuration of workflow nodes (Host Information), furnishes a history of events published over the event bus (Event Repository), and allows for the distributed many-membered configuration of ELF/Ogrescript jobs (Tuple Space).

2.3 *ELF/Ogrescript*

The execution of application codes, along with the local logic in which they are embedded, take place via Ogrescript run inside of the ELF container.

Ogrescript is a highly extensible XML scripting language which not only provides full programming flow of control features (conditions, loops, parallelism, join, sleep, wait, exception handling, etc.) and evaluation of variables or arithmetic-logical expressions in a scoped environment, but also a series of standard plugin modules for common programming tasks (file movement, string operations, system calls, events, etc.), along with some particularly hard-to-find capabilities useful in the context of Fortran-based applications (e.g., a set of tasks for parsing, modifying and writing out namelist files).

[*]An example of a WRF parametric workflow could be a multiple-physics and/or multiple-data parameter study.

The standard ELF container runs Ogrescript directly; the ELF glide-in container, however, acts as a monitor, pulling work (even as it becomes available) from the Tuple Space service and turning it into individual ELF jobs on the compute resources it has in its possession.

2.4 *Event Bus and Metadata*

The event bus in our system is strictly a mechanism for publishing provenance, metadata or debugging information (we now do not rely on it for critical state transitions).

It has been our experience, in fact, that the remote events produced by PWE and ELF are extremely convenient when debugging a distributed workflow, for usually (though not always) a problem can be immediately diagnosed simply by retrieving the events (through SIEGE) and inspecting them. It is only when these Java-based events are insufficient (such as for more arcane system-, network- or protocol-level errors) that the user is constrained to go directly to the remote resource and inspect additional log files for potentially revelatory information.

To a large degree the quantity of logging/debugging information made available as remote events can be configured for any given workflow by setting the event level attribute. When turned down, the debugging information goes only to local log files and is not transmitted as events over the bus. This is crucial, in fact, when long-running, complicated or many-membered workflows are run, for traffic may otherwise exceed what the event bus can actually handle.

2.5 *SIEGE*

The user interacts with our services through this desktop client, which is a standalone RCP (Eclipse) application. Siege runs on Linux, Windows and Macintosh systems. Siege currently has two main perspectives, or views: the PWE perspective for launching and monitoring workflows, and VIZIER perspective for configuring Host Information entries, inspecting, deleting or adding tuples to the Tuple Space service, or for retrieving events from the Event Repository. In addition, there is a Repository view associated with both perspectives; this view is keyed to a local directory (e.g. on a user's laptop) designated as the repository, where XML conforming to well-defined models (most importantly, the workflow-builder XML) can be stored and edited for eventual submission to the services. The next section demonstrates some typical usage scenarios for these perspectives.

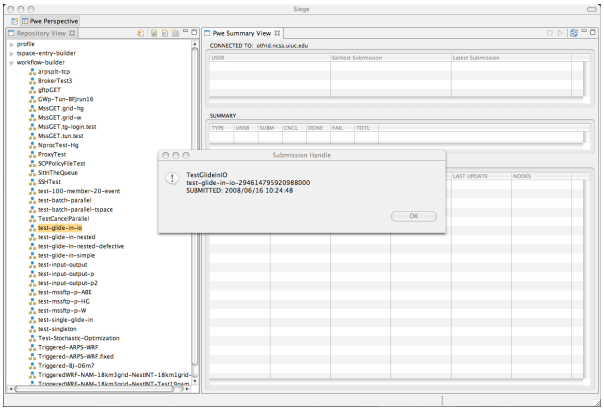

Figure 2. Siege: Submitting Workflow

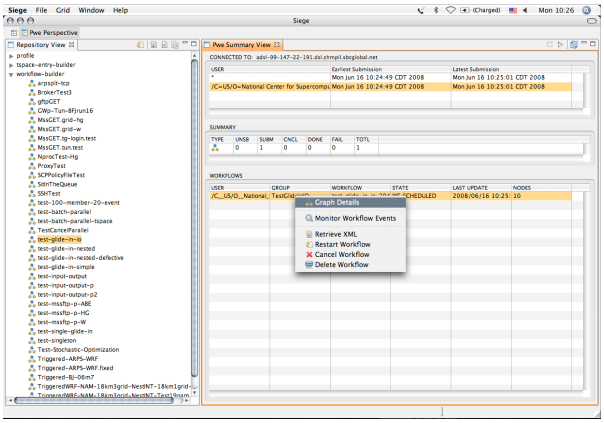

Figure 3. Siege: Viewing Workflow Summary

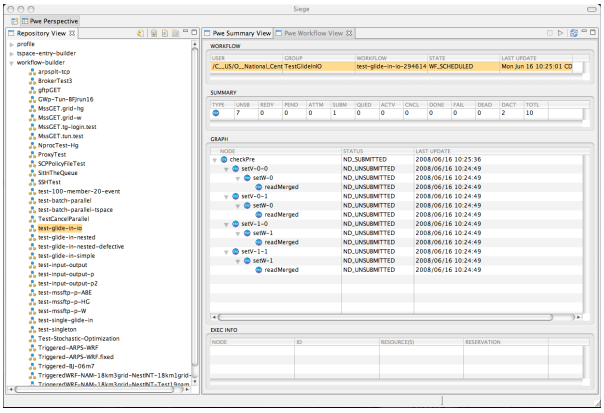

Figure 4. Siege: Viewing Workflow Details

3. Summary and Future Work

The PWE system is currently used or planned for use in atmospheric sciences, astrophysics and other fields. Scientific workflows may include nodes executed serially or in parallel, with dependencies or children, and with parameterization possible over text (e.g. file names), integers (e.g. namelist physics options), real numbers (such as runtime parameters), or a combination thereof.

Parameterized "glide-in" submission is now in alpha-testing, and will hopefully prove useful on a number of different machine architectures, including the AIX/Load Leveller -based Blue Waters project at NCSA. In the upcoming months we intend to address the following:

1. The addition of logic-programming type rules for pruning a parametric expansion. In many cases, a dense matrix of values is not necessary, and if the user can express the exceptions, we can avoid producing and running those members unnecessarily.
2. The implementation of a metadata agent and a corresponding perspective in SIEGE for querying over stored metadata from experiments or runs.
3. The addition of a namelist perspective in Siege in which the programmatic manipulation now available through Ogrescript can be harnessed manually by the user to create namelist files.
4. The addition of a workflow composition perspective which will allow the user to assemble all but the Ogrescript part of the workflow description via widgets or wizards.
5. The integration of real scheduling modules into PWE. We would like to be able to submit to an external scheduler a set of open-ended requests based on resource requirements, and get back resource names on which those requirement can be met inside a determinate time window. Batch-Queue Predictor and the MOAB scheduler are candidates for further exploration in this regard.

4. Acknowledgments

Workflow broker development has been supported by LEAD, a Large Information Technology Research (ITR) grant funded by NSF under the following Cooperative Agreements: ATM-0331594 (Univ. of Oklahoma), ATM-0331591 (Colorado State Univ.), ATM-0331574 (Millersville Univ.), ATM-0331480 (Indiana Univ.), ATM-0331579 (Univ. of Alabama in Huntsville), ATM03-31586 (Howard Univ.), ATM-0331587 (Univ. Corporation for Atmospheric Research), and ATM-0331578 (Univ. of Illinois at Urbana-Champaign, with a sub-contract to the Univ. of North Carolina). This work has also been supported by SCI03-30554, SCI04-38712, and SCI96-19019.