A Framework for Coupled Multi-physics Simulations *****Jeff: some charming title?? emphasizing "it can cover various kinds of applications in different requirement, main components can be replaced by other similar softwares, etc."**

Soon-Heum Ko[1,2], Nayong Kim[2], Shantenu Jha[3,2]

[1]*National Supercomputing Centre, Linöping University, Linöping, Sweden*

[2]*Center for Computation & Technology, Louisiana State University, USA*

[3]*Dept. of Elec. and Comp. Eng., Rutgers University, Piscataway, New Jersey, USA*

Abstract

****Jeff: It is written considering that we will develop the hybrid simulation framework where multiple distinct application codes are coupled. We shall decide first* ***whether we will focus only on a framework for a computationally coupled multiple executions in parallel or we will also cover the workflow between computational and experimental tasks scheduled in tandem.*** *we design and develop a multi-physics framework for coupled simulations in which scientific components (codes) are logically separated. Designed framework provides the interface between scientific components, the compiling service for different system architectures, a runtime environment for scheduling of coupled tasks, and data management/code versioning support. The framework is built by developing the data exchange interface between coupled codes, providing the compilation templates for specific architectures, adopting a BigJob abstraction, and using the PetaShare service. We apply this framework into the hybrid computational fluid dynamics - particle dynamics application to demonstrate its capability.*

I. Introduction and Motivation

"Coupled scientific simulations" will refer to the scientific research procedure which involves the data exchange between different components operating either in parallel or in tandem. In both cases, whether scientific tools are synchronously exchanging the information during a single run or individual tools iteratively give the feedback to their counterparts, scheduling the overall procedure and providing the data stream under the distributed computing infrastructure is a headache to domain scientists. It motivates the development of a framework for coupled multi-component applications, which provides the following capabilities: (1) the framework provides the data interface between distinct tools, (2) embedded softwares should be adaptive to different users' implementations, and (3) they should be removable/replaceable by users' preferences.

Two different strategies can be applied to designing a coupled multi- component framework. One is to modularize all components and bind them into a single executable, and the other is to keep multiple standalone softwares and provide the interface between individual executable. The former is good for scheduling on most batch queue systems and effectively using allocated resources, while some restrictions on data structure or standard grammar may apply on users' codes. Also, the binary can be heavier since the executable might contain unused or non-optimized functions for solving specific target applications. The other gives much freedom in writing the individual component but can cause a computational headache in scheduling these components in remote production systems. We consider that the integration to a single binary is recommendable if all software packages are tightly coupled together for a single task or they share most memory allocations. It is preferred to couple multiple components through the communication interface and the scheduling function if those packages are sequentially loaded or they have different code structures.

A runtime environment for a coupled multi-physics application [6] has been developed previously. In this work, two coupled yet logically distributed scientific softwares have been effectively scheduled under the single batch queue allocation, by incorporating the load balancing capability on a BigJob abstraction [7]. It demonstrated that a Pilot-job formulation can ease the scheduling of a coupled application whose components are hardly packed into a single binary. However this runtime environment lacks the reusability because it only provides the scheduling functionality between already-coupled distinct application codes.

In this work, we design and develop a coupled multi-component framework. Along with the runtime environment between logically separated components, we also provide the standard interface between these coupled components and take care of the data management/versioning. The structure of a coupled simulation framework is introduced in Sec. II. Specific softwares implemented in this framework are described in Sec. III. A coupled multi-physics application and an experiment-computation integrated research procedure are presented in Sec. IV. Recommendation for future work and conclusions are presented in Sec. V and Sec. VI.

II. Design of a Coupled Simulation Framework

A. Requirements

***Jeff: In this section, we shall contain **what functionality the framework should provide** in easy words.

Packaging multiple components into a single binary is very tough if (1) each component uses very different computational kernels or (2) a manual processing/decision-making is included in one of solution procedures. A hybrid computational fluid dynamics (CFD) - molecular dynamics (MD) simulation is an example whose data structures are completely different so that it is fairly hard to incorporate application modules into a single binary. Another situation is occasionally observed when there is an iterative feedback between a numerical simulation and the experimental measurement. Most probably, the physical experiment involves the human labor of changing specimen/mockup after the numerical investigation, which cannot be digitalized. Figure 1 describes the solution procedure for these two multi-component simulations.

A coupled multi-component simulation framework should be capable of scheduling the overall procedure (workflow) and providing co-scheduled execution between multiple tasks (runtime environment). The workflow shall run middleware and scientific software packages according to the pre-described schedule. A runtime environment executes coupled multiple softwares in parallel, in the form of a virtually single executable under the batch queue system.

The functionality to handle the data flow (data management/archiving) and the standardised communicator for the coupled simulation (coupling interface) will provide more convenience in performing the coupled simulation and building coupled software packages. The framework should be capable of transerring data between distributed

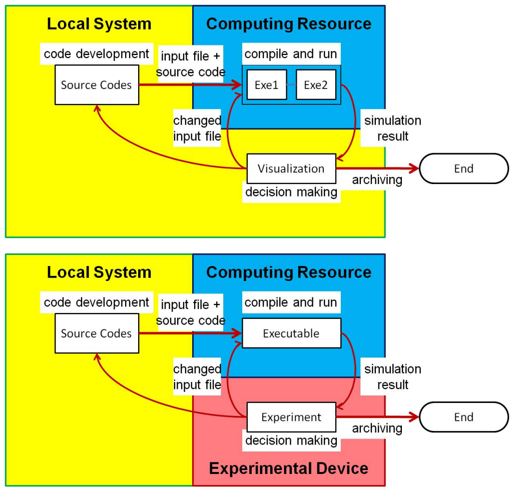

Fig. 1. Solution procedure of (1) coupled multi-physics simulations and (2) successive feedback between computations and experiments. In both cases, frequent data transfer takes place between local and remote systems and a manual decision-making intervenes (either by visualizing the numerical solution or by performing the experimental measurement). In addition, the synchronous communication takes place between co-scheduled computational components in case scietific softwares are coupled in parallel.

tools when each operation completes; Executables on remote computing facilities should be up-to-date when there is a change on the source code in the local server; Final solutions from multiple different parameter sets should be archived effectively. In addition, the capability of exchanging datasets between destinct softwares without the explicit and direct access between individual source code will ease the development of coupled scientific softwares. (or: the supprot of a standardised communicator between distinct codes will ease the development of coupled scientific softwares.)

B. Design

***Jeff: In this section, we address **what functionality each group/service will provide and how we can give these services** in more professional terms.

We design the concrete structure of a coupled multi-component simulation framework as presented in Fig. 2. Looking at the operation flow, the updated source code from the local machine replaces the old version in the remote computing resource and the individual source code is coupled together during the compilation, by the help of the external coupling interface. A runtime environment runs these distinct-yet-coupled executables concurrently. The solution is referenced by a scientist and the decision is made to accept/decline the numerical solution (either through the visualization or the experimental measurement). The acceptable solution is archived in the local server, or the new computation is performed by changing simulation parameters or source codes. In case the computation and the experimential measurement are coupled in sequence, compilation and job submission processes become simpler.

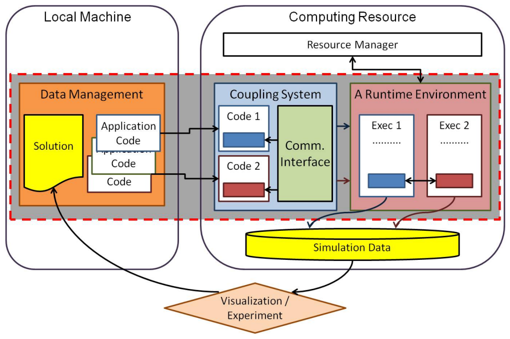

Fig. 2. Structure of a coupled multi-component application framework. The framework is composed of three main packages: a data archiving/versioning service, a coupler and a runtime environment. A workflow script is running these packages according to the coupled simulation procedure.

Data management service provides the data sharing among distributed computing facilities, updating the source codes on local and remote machines, and archiving final solutions. These functions can be implemented by running a couple of secure FTP commands; Installing a versioning service such as CVS (Concurrent Versioning System: [4]), SVN (Apache Subversion: [2]), or GIT (The fast version control system: [9]) on a central repository server is another way; Using a data Grid service such as PetaShare [1] can be helpful if sharing large-scale data on a distributed environment is of particular interest.

Coupling system provides the interface for the information exchange between distinct application codes and helps the compiling procedure in different system architectures. The coupling interface can be supported either in the form of a library which opens a file-based channel between distributed executables, or as a separate binary which binds all MPI communicators under a single MPI runtime environment. In either case, the interface function calls should be deployed in coupled source codes. Compilation system takes care of compiling individual scientific codes and coupling interface. It wraps up compiling options on users' makefiles to add the linkage with the coupling library and to apply optimal options for specific system architectures. A single wrap-up script along with templates which store optimal compilation options for known system architectures can provide these capabilities up to some extent; Some advanced compilation systems such as the Simulation Factory [10] can provide more functionality independent of different system architectures and configurations.

A runtime environment is designed to co-schedule and load-balance between coupled distinct tasks. The above issue is easily resolved by allocating a pilot-job in which coupled subtasks are effectively scheduled. A number of pilot-job implementations such as a Condor Glidein Job [5], a BigJob abstraction [7], or a Pilot Factory [3], are present. Also, some supercomputing centers provide the capability of concurrently starting multiple jobs in the basic level.

III. Implementation

***Jeff: In this section, we shall explain **how each service is implemented in our specific framework**.

A. Data Management and Archiving

B. Coupling System

Coupling system services (1) the information exchange interface between coupled distinct codes, and (2) the Makefile-wrapper for specific system architectures. The former helps in writing the application code in a generic form and the latter enables compiling the program independent of the system architecture.

The coupling interface is provided in the form of a library. It acts to exchange registered variables between two codes. A "separate" library is linked to individual executable and the information exchange takes place between the library interfaces through the file-based communicator. Initially, parameters from users' codes, i.e., the communicator ID (any strings which define a set of coupled application) and the element ID (either 1 or 2), are registered to the library. Upon the calling of a *send* function, the library stores passed data to the file communicator whose unique name is provided by the communicator ID and the element ID. In time of loading the *recv* function, the library scans (and waits for) the data file from the counterpart and returns this dataset to the scientific code. Compared with the MPI-based communicator, the file I/O has the low latency/bandwidth and can have the stability issue in cases the resource experiences the unstability in file system. On the other hand, the file communicator library form does not require the dedicated resources compared with the separate MPI-based coupler and it is basically free to cooperate on distributed resources.

The Makefile-wrapper overwrites the user's Makefile so as to link the coupling interface and to apply optimal compiling options. It consists of a wrapper script and a number of templates which specify optimal compiling options for well-known system architectures. The script replaces user's compiling flag with the recommended ones for that system architecture and adds the linking flag of the coupling interface. The wrapper gets the location and the name of user's Makefile, as well as the location of a coupling library. By default, they are set to be *./*, *Makefile*, and *../Lib*. The wrapper also accepts user's flags from the input file, which prioritize over the recommended configuration for that system architecture.

C. Runtime Environment

A BigJob is a pilot-job implementation under the simple API for grid applications (SAGA) [8]. SAGA provides a simple, POSIX-style API to the most common grid functions at a sufficiently high-level of abstraction in order to be independent of diverse and dynamic grid environments. The SAGA specification defines interfaces for the most common grid-programming functions grouped as a set of functional packages. Thanks to various adaptors in SAGA, the BigJob application is infrastructure-neutral, which is the strength over other Pilot-job implementations.

We apply the previously-established BigJob runtime environment for coupled multi-physics applications [6]. The load balancing function is turned off since it requires relevant changes (time checking function and checkpointing capability) in the application code. We directly submit a job to the remote batch queue system in case the sequential feedback process between a computation and an experiment is of interest.

D. Workflow Engine

The workflow engine consists of a number of hierarchical scripts. The main script *hybrid* loads each relevant module according to the procedure of coupled simulations. (introduce each service, along with how it operates. Especially, about the first procedure which copies local files to petashare location) shall provide the hierarchical structure where modules will run.

The first characteristic is the simplicity. (minimize variables to edit, while yours can still handle many parameters if not set default.)

The second and more important feature is the controllability. (you can start from the middle, skip some procedures, restart from where it ended.)

IV. Numerical Experiments

V. Further Achievements

VI. Conclusions

Acknowledgment

References

[1] A Distributed Data Archival, Analysis and Visualization Cyberinfrastructure for Data-intensive Collaborative Research. http://www.petashare.org/.
[2] Apache Subversion. http://subversion.apache.org/.
[3] P.-H. Chiu and M. Potekhin. Pilot factory - a condor-based system for scalable pilot job generation in the panda wms framework. *Journal fo Physics: Conference Series*, 219:062041:1–7, 2010.
[4] Concurrent Versioning System. http://www.cvshome.org/eng/.
[5] Condor Project. http://www.cs.wisc.edu/condor/.
[6] S.-H. Ko, N. Kim, J. Kim, A. Thota, and S. Jha. Efficient runtime environment for coupled multi-physics simulations: Dynamic resource allocation and load-balancing. In *The 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2010)*, 2010.
[7] A. Luckow, S. Jha, J. Kim, A. Merzky, and B. Schnor. Adaptive Replica-Exchange Simulations. *Royal Society Philosophical Transactions A*, pages 2595–2606, June 2009.
[8] SAGA. http://saga.cct.lsu.edu, 2011.
[9] The fast version control system. http://git-scm.com/.
[10] The Simulation Factory. http://simfactory.org/.