Efficient Runtime Environment for Coupled Multi-Physics Simulations: Dynamic Resource Allocation and Load-Balancing

Soon-Heum Ko[1], Nayong Kim[1], Joohyun Kim[1],
Abhinav Thota[1], Yaakoub el-Khamra[3], Shantenu Jha[*1,2]
[1]*Center for Computation & Technology, Louisiana State University, USA*
[2]*Department of Computer Science, Louisiana State University, USA*
[3]*Texas Advanced Computing Center, Austin, Texas, USA*
[*]*Contact Author*

Abstract

Coupled Multi-Physics simulations, such as hybrid CFD-MD simulations, represent an increasingly important class of applications. Often the physical problems of interest demand the use of high-end computers, such as TeraGrid resources, which are often accessible only via batch-queues. We develop and demonstrate a novel approach to overcoming the co-scheduling requirements associated with coupled runs. Our solution which is developed using SAGA and a SAGA-based framework (BigJob), is a generalization of the Pilot-Job concept – which in of itself is not new, but is applied to coupled simulations for the first time. Our solution not only overcomes the initial co-scheduling problem, but provides a dynamic resource allocation mechanism. Such support for dynamic resources is critical for a load-balancing mechanism, which we develop and demonstrate to be effective at reducing the total time-to-solution of the problem. We also demonstrate for the first time that we are aware of, the use of multiple Pilot-Job mechanisms to solve the same problem; specifically, we use SAGA to access the SAGA-based Pilot-Job on high-end resources whilst using the native Condor Pilot-Job (Glide-in) on Condor resources, and importantly both are accessed via the same interface.

I. Introduction

Coupled Multi-Physics simulation techniques are being increasingly used to study many different physical phenomenon spanning time and length scales at different level of details [1] [2]. These techniques have been used to investigate phenomena from crack-propagation in materials, biological systems as well as understanding multi-phase fluid flow in constrained geometry.

In addition to the "physics challenges" of these Multi-Physics coupled simulations, there exist interesting "computational challenges". Probably the best known (and investigated) is the challenge of simulating large and complex systems, leading to simulations that require greater computational resources – often involving HPC resources, and no longer working on dedicated PCs. Parallelization helps address the individual codes, but incorporating two distinct codes under the umbrella of a single tightly-coupled application (say using MPI) is not without significant problems.

Here we will focus on the challenges arising from running tightly-coupled simulations on production systems with batch-queues, whereby it cannot be guaranteed that two separate jobs will execute concurrently. Specifically we will consider the case of coupling a Computational Fluid Dynamics (CFD) code and a Molecular Dynamics (MD) code, whereby the communication is via the exchange of files and not Unix pipes (see next section for details on the coupling). Although not exactly tightly-coupled in the sense of MPI, i.e., very frequent and extreme sensitivity to latency in communication delay, the CFD and MD codes have frequent communications, (e.g., the CFD code conducts data exchange in every iteration) they need to run concurrently. Thus, without explicit support for co-scheduling, it is unlikely that coupled CFD-MD simulations will run concurrently as inevitably the first job to run will have to wait for the other to follow.

Even if they could run concurrently, without explicit load-management/balancing support, there is likely to be inefficient utilization of compute resources due to load imbalance. As the performance of each tool changes with computing resource and problem size, re-adjustment of allocated resources to each task according to their performance is required during the simulation. However, if simulations have been submitted as independent jobs, changing CPU allocation to address these changes is challenging. Thus, the best way using conventional job submission system would be to find a site with sufficient resource pool and submit two jobs with the optimal number of processors according to the pre-test data on performance of each tool in that facility with the same problem size.

Another important challenge, especially for large-scale simulations is the need for efficient load-balancing, taking into account the individual simulation performance. Interestingly, as we will show, effective load-balancing of two independent but concurrently running codes introduces the need for dynamic resource allocation, and the same solution that we devise to overcome the co-scheduling requirement/constraint of concurrent jobs supports the feature of dynamic resource allocation. But given the lack of System or Service-level support to address the challenges outlined above, there is a need to address the solution at the application level. This paper aims to provide novel solutions to the above problem using frameworks that are in user (application) space.

We outline our approach and solution – which is not-tied to either a specific application set (or infrastructure) and is scalable and extensible. SAGA (the Simple API for Grid Applications) [3] is a high-level API which provides the basic functionality required to implement distributed functionally – both logically and physically, in an infrastructure and middleware independent fashion. SAGA enables the creation of higher-levels of abstractions, for example a container-job and pilot-job (which as we will discuss is referred to as the BigJob [4]). Although the container-Job/Pilot-Job concept is not novel *per se*, we believe this is the first documented utilization of these abstractions to perform coupled Multi-Physics simulations. Additionally, our approach employing a SAGA-based Pilot-Job is infrastructure neutral, unlike most other Pilot-Jobs. As our experiments will show, performance improvement arise from removing the need for scheduling the two-components separately and in providing a single job-requirement to the queuing system. Additional efficiency is provided via application scenario specific load balancing modules. We begin the next section with an outline of the basic motivation for coupled simulations and load balancing.

II. Hybrid CFD-MD Approach: Understanding the Coupling, Communication and Load-Balancing Requirements

The hybrid CFD/MD approach [5], [6], [7] is a simulation method which adopts the continuum hypothesis in capturing macroscopic features of a flow-field and details atomistic intermolecular interactions on interfaces of different materials by using the MD approach. CFD can accurately predict flow properties on conventional moderate/large size fluid domains, but is intrinsically impossible to reflect characteristics of surrounding solid materials. While MD can provide atomistic level resolution of interactions between particles, it becomes computationally demanding as the size of simulated system grows. The hybrid approach provides a good balance between computational cost and atomistic details/resolution.

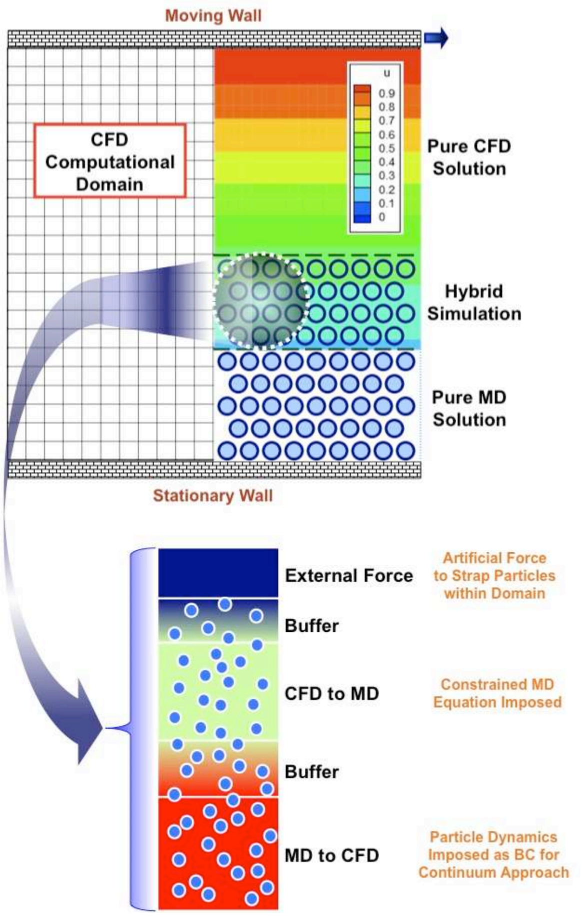

Fig. 1. Schematic of CFD/MD Coupled Simulation of Channel Flow

A scientific problem that can be effectively tackled by the hybrid CFD/MD approach would be, for example, the simulation of a flow-field where the viscous effect of solid boundary is so dominant that the length scale required becomes significantly larger than the typical size of a system conducted by the conventional MD. Additionally, understanding of this fluid system near the wall is profoundly important but can be achieved only through atomistic molecular dynamics. One solution is, as is seen in Fig. 1, to carry out the hybrid CFD/MD approaches with which atomistic interactions between solid elements and fluid particles near the wall is simulated by MD and the far field flow is calculated by CFD.

In this study, we used the MPI version of the in-house incompressible CFD code [8] for CFD and the MPI version of the in-house modified version of LAMMPS [9] for MD. As illustrated in Fig. 1, the coupling mechanism is the key component for successful hybrid CFD/MD simulations and our implementation follows the literature [6], [7]. In brief, the hybrid region where the coupling mechanism between MD region and CFD region is executed comprises five sub-regions. In the CFD boundary grid region positioned near the pure MD region, velocities of particles obtained with MD are averaged and used for boundary condition for the corresponding CFD computational cells. The MD boundary zone is placed above the CFD boundary zone and here, information on velocities from the CFD grid are imposed on particles in the zone through dynamically constrained equation of motion for MD. Between these zones, a buffer layer exists to avoid any harmful direct influences from one zone to another zone. The truncated and shifted Lennard-Jones potential is used for interactions of particles in MD simulation.

As clearly indicated in Fig. 1, the most prominent computational challenge is how to run efficiently two separate stand alone applications while efficiently conducting information exchange. In other words, the time-to-solution is heavily dependent upon whether a runtime environment can provide, (i) low collective-waiting times (arising from batch queue wait times), and (ii) prevent an imbalances in time to reach information exchange steps between the two codes. The imbalance arises due to different performance between two distinctly heterogeneous applications, CFD and MD, resulting in an unavoidable time gap between arrival times of CFD and MD for the exchange step. We propose to address this using dynamical resource allocation mechanism with a load-balancing mechanism. In that sense, our SAGA-based framework provides a single efficient runtime environment for the coupled simulation.

III. SAGA and SAGA-based Frameworks for Large-Scale and Distributed Computation

The Simple API for Grid Applications (SAGA) is an API standardization effort within the Open Grid Forum (OGF) [10], an international standards development body concerned primarily with standards for distributed computing. SAGA provides a simple, POSIX-style API to the most common Grid functions at a sufficiently high-level of abstraction so as to be independent of the diverse and dynamic Grid environments. The SAGA specification defines interfaces for the most common Grid-programming functions grouped as a set of functional packages (Fig. 2). Some key packages are:

- File package - provides methods for accessing local and remote filesystems, browsing directories, moving, copying, and deleting files, setting access permissions, as well as zero-copy reading and writing

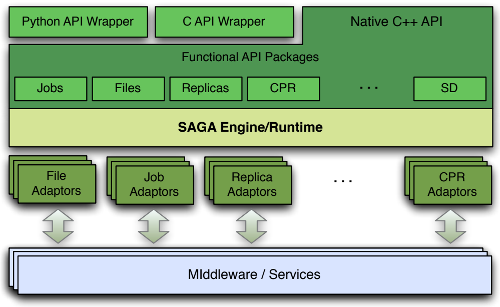

Fig. 2. Layered schematic of the different components of the SAGA landscape. At the topmost level is the simple integrated API which provides the basic functionality for distributed computing. Our BigJob abstraction is built upon this SAGA layer using Python API wrapper

- Job package - provides methods for describing, submitting, monitoring, and controlling local and remote jobs. Many parts of this package were derived from the largely adopted DRMAA specification.
- Stream package - provides methods for authenticated local and remote socket connections with hooks to support authorization and encryption schemes.
- Other Packages, such as the RPC (remote procedure call) and Replica package

Fig. 3 shows the structure of BigJob and its operation flow. When a BigJob is submitted to the remote resource, the application manager monitors the status of this pilot-job through the advert service. When resources are allocated to the BigJob, the application manager divides obtained resources to its sub-jobs and a coupled simulation starts under the control of a multi-physics agent in the remote resource. Advert service keeps on getting the status of a pilot-job from the queuing system and the status of sub-jobs from multi-physics agent, also delivering these information to the application manager by a push-pull mechanism. The application manager watches the status of sub-jobs and decides the next event when the coupled simulation is finished. When one default BigJob is launched, subtasks keeps running until final solution is achieved and the manager quits the pilot-job at that time. In cases multiple BigJobs are submitted for the same simulation or a load balancing function is included, sub-jobs experience several restarts from their checkpointing data, reflecting changed processor allocation to each application. In former case, resource allocation to each sub-job follows a pre-defined map according to the number of BigJobs allotted to this simulation: In latter case, resource allocation to each sub-job becomes dynamic according to its performance, to be discussed in the next section.

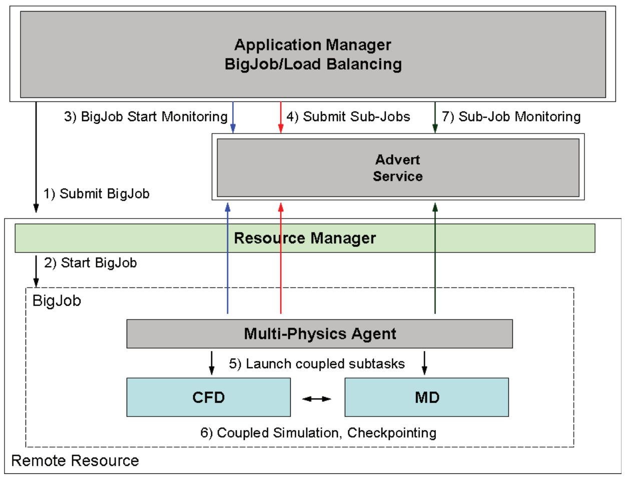

Fig. 3. Architecture of the Controller/Manager and Control Flow: Application manager is responsible for job management including BigJob and sub-job submission, their status monitoring functions. We implement a load balancing module, and migration service based on job information. Application agent system resides on each HPC resource and conducts job information gathering as well as communication with application manager via the advert service

IV. Load Balancing for Coupled Multi-Physics Simulation

SAGA and its pilot-job framework (BigJob) enable coupled, yet distinct simulations to be submitted to queuing system. This is done by submitting one container job, and re-distributing its processors to each task. Also, total execution time can further be reduced by assigning more BigJobs and dynamically reallocating resources to each sub-task with increased number of processors. However, the flexibility to re-distribute resources (processors) to the individual task, does not imply efficient utilization. This is the responsibility of a load-balancer (LB) We will discuss the implementation and algorithm of this LB; it is important to mention that the LB functions in the context of the SAGA-BigJob framework.

Each applications load is determined by its elapsed time to run the evolution loop. Here, time for initialization or inter-domain data exchange are excluded from the counting, because they are one-time event or irrelevant to application's performance. As the result of load balancing is reflected in the next launch of simulation codes, applications should be able to restart from their checkpointing data. Also, they should be equipped with generalized domain partitioning routine to run simulation with any number of processors, without harming their parallel efficiency a lot. If above conditions are satisfied, BigJob application manager, can be interfaced with the LB to implement the dynamic resource allocation between tasks. The idea of current load balancing algorithm is to assign more processors to a sub-task with more runtime until two codes show the same runtime. We have adopted some assumptions to simplify load balancing procedure and apply the algorithm without considering applications' characteristics and individual code's scalability. Assumptions are, (1) Each application code follows the ideal parallel efficiency. (2) All processors in one node is assigned to one sub-job.

Let the computation time of two applications as t_{CFD} and t_{MD}, processors to each domain as PE_{CFD} and PE_{MD}, respectively. Based on assumption (1), workload on each application remains the same after the reallocation of resources,

$$\begin{aligned} W_{CFD} &= PE_{CFD} \times t_{CFD} = PE'_{CFD} \times t'_{CFD}, \\ W_{MD} &= PE_{MD} \times t_{MD} = PE'_{MD} \times t'_{MD} \end{aligned} \tag{1}$$

Still, total number of processor remains the same:

$$PE_{TOT} = PE_{CFD} + PE_{MD} = PE'_{CFD} + PE'_{MD} \tag{2}$$

Our objective is to reduce the computation time of an over-loaded subtask until two simulations show the same computation time, $t'_{CFD} = t'_{MD}$. Then, Equation 1 and Equation 2 concludes the optimal processors for one sub-task would be

$$PE'_{CFD} = \frac{W_{CFD}}{(W_{CFD} + W_{MD})} \times PE_{TOT} \tag{3}$$

The other sub-job will follow the similar expression. The optimal processor distribution from above equation will return a float value. Also, considering the second assumption, which is the policy of many supercomputing center, the above distribution needs to be asymptotic. If CPU cores in one node is N_{UNIT}, optimal condition can be gained by finding $S = int(PE'_{CFD}/N_{UNIT})$ and comparing $MAX(t'_{CFD}, t'_{MD})$ between processor allocations on CFD task with $N_{UNIT} \times S$ and $N_{UNIT} \times (S+1)$.

V. Dynamic Resource Allocation for Coupled Simulations: Three Scenarios

In Section 2, we have argued the challenges of running a coupled multi-physical simulation on a conventional queuing system as (i) Hardness to start multiple applications concurrently; (ii) Inability to balance the load among domains, and (iii) Fixed number of allocated resources throughout the simulation. To resolve the co-scheduling and load-imbalance issues, and with the ultimate aim of reducing the time to solution, we have used the BigJob abstraction with load balancing module to the coupled simulation. Three different real scenarios arise. In the first

Use Case, a single BigJob is utilized to run the coupled-simulations, first without, and then with the LB redistributing resources based upon their individual performance. One component maybe relieved of its resources for the greater good. In the second Use Case, two BigJobs are started together, but often one BigJob starts before the other; to increase efficiency both coupled simulations start with whatever resources are available as part of the first (to run) BigJob. When the other BigJob becomes active, there is a dynamic redistribution of tasks, such that the larger of the two simulations is assigned the bigger BigJob. The variant of the above when the two BigJobs are on different machines forms Use Case 3. In the remainder of this section we discuss the details of these three different Use Cases.

A. One BigJob with Load Balancing

Fig. 4 shows the flow of a coupled simulation with the conventional job submission and through one BigJob. In a conventional way, individual tasks with resource requirements of PE_{CFD} and PE_{MD}, respectively, are independently submitted to the conventional queuing system and job scheduler recognizes these coupled tasks as two distinct jobs. Thus, they are going to start at a different time, except when enough resources are available. In this case, both tasks wait on the queue when no job is allocated, the first allocated job idles to do data exchange with its counterpart, and run actual simulation after both jobs are allocated. On the other hand, in a BigJob simulation, a pilot job with the size of $PE_{CFD} + PE_{MD}$ is submitted to the queue and coupled simulation directly starts when the resource is assigned on this BigJob. By using a BigJob, a user can save his/her account consumption on inactive mode in conventional job submission, while total runtime is the same if the resource distribution to sub-jobs is identical. However, eliminating inactive mode does not guarantee total simulation time is reduced, because waiting to get one bigger allocation may happen to take more than getting two allocations with smaller chunks. The same situation can happen to the load-balanced case with one BigJob, which satisfies the reduction of total runtime compared to other examples.

Table I shows elapsed time on running above test cases: conventional job submission, one default BigJob launch and one BigJob with load balancing. These tests were conducted on supercomputing resources in LONI (Louisiana Optical Network Initiative) [11] and tested resources of eric, oliver and louie have the same specification with 4 processors in each node. A coupled simulation has been conducted by using 32 processors with two different processor allocations to subtasks. Application codes experience four restarting from their checkpointing data in

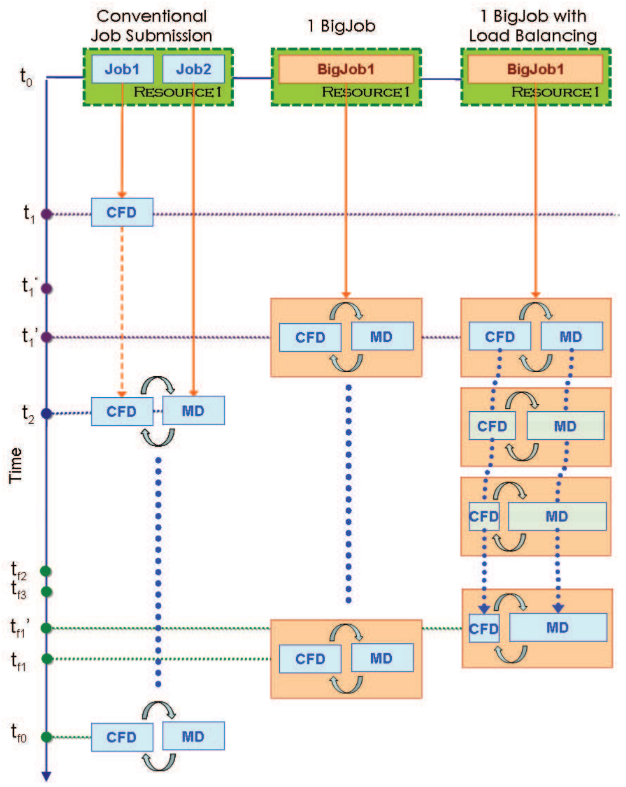

Fig. 4. Comparison of dependencies encountered for coupled-simulations submitted as conventional Jobs, to the scenario when using Pilot-Jobs (BigJob). Here we use only 1 BigJob. The conventional mode of submission experiences three phases of queue waiting (all jobs are waiting: $t_1 - t_0$), inactive mode (one job is waiting for another: $t_2 - t_1$), and active running (running a coupled simulation: $t_f - t_2$). On the other hand, inactive mode disappears when coupled simulation runs within one BigJob, as an allocated BigJob distributes its resource to sub-jobs.

load balancing test, while simulation keeps running until it completes in baseline and default BigJob tests. When the same number of processors are allocated to CFD and MD tasks, one BigJob simulation with LB shows better performance on runtime compared to other two cases. Load balancing function changes the resource distribution to the best condition after the first simulation loop and 8 processors from CFD task are moved to MD simulation from the next start. As a result, total runtime is saved by about 20 percent compared to other test cases. On the other hand, a load-balanced BigJob shows worse performance in the latter test when initial processor distribution is an optimal condition, because running a load balancing function and trying checkpointing and restarting subtasks becomes additional costs. Comparing results between conventional job allocation and one BigJob simulation (with or without LB), it is guaranteed that a lot of computational cost on idling is saved. However, it does not guarantee that one BigJob allocation satisfies the reduction of total simulation time: waiting time on the queue purely depends on the condition of remote resource.

TABLE I. Results for performance data using BigJob with and without LB. The Baseline Simulation represents the case when BigJob is not used; in this case, the time to start is dominated by the Inactive Mode of the longer waiting task. The use of BigJob resolves, this as can be seen by the reduced inactive mode time in columns 3 and 4. The starting configuration assigns resources equally between CFD-MD – 16px each. However, after a few iterations of the LB the configuration is 8-24. It is with this configuration that the performance is better with LB, than without the LB 1370 ±98 vs 1641 ± 162. The starting configuration of the second set (8-24) is a "balanced configuration" which is why there is no performance gains on using the LB. Values in table shows averaged time in seconds, values within a parenthesis are standard deviation of time elapsed; the average is taken over 5 distinct experiments

16-16	Baseline Simulation	BigJob (no LB)	BigJob (LB)
Waiting on the Queue	3000 (4136)	10834 (4593)	15201 (10968)
Inactive Mode	10300 (15327)	4 (0)	4 (0)
Active Runtime	1672 (246)	1641 (162)	1370 (98)
Total Time	14973 (19213)	12480 (4430)	16577 (11004)
8-24	Baseline Simulation	BigJob (no LB)	BigJob (LB)
Waiting on the Queue	1900 (1873)	3372 (6450)	10344 (6767)
Inactive Mode	5486 (10318)	4 (0)	4 (0)
Active Runtime	1171 (175)	1169 (72)	1280 (81)
Total Time	8557 (11882)	4545 (6457)	11629 (6720)

A number of experiments have been conducted to validate the performance of a load balancing function with different BigJob size. Fig. 5 shows the change of processor allocation to sub-jobs and their runtime at every start. A BigJob with 24 processors is tested with two kinds of initial processor distribution between CFD and MD, 8 to 16 or 16 to 8. In all simulations, the load balancing function detects the ratio of 4 and 20 between CFD and MD tasks are optimal and converges to this ratio at the next restart, except one case when CFD solution experiences unexpected overload initially. Even in this case, load balancing function traces the optimal load allocation during the iteration and this demonstrates the stability of current load balancing function. After the balancing, runtimes of CFD and MD jobs become very close, 219 seconds for CFD and 236 for MD task.

Though the above results show the large performance gain by employing a load balancing algorithm, some limitations are also observed. First, as the function only refers

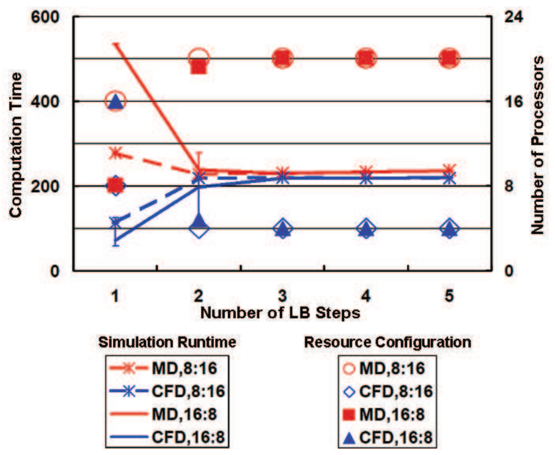

Fig. 5. Plot showing the convergence in resource configuration and runtime with load-balancing steps. Lines show runtime change of each sub-job, symbols represent the history of load balancing at every start/restart. When the starting distribution between CFD and MD tasks are either 8-16 or 16-8, load balancer converges to a distribution of 4 and 20 processors for CFD and MD respectively. The coupled simulation time reduces to around 235 seconds. All tests were repeated 5 times; the bars in the graph represent minimum and maximum values from the 5 tests.

to the computation cost at given processor distribution and moves to the optimal condition iteratively, it takes time to achieve a converged resource allocation if initial processor allocation starts from other extreme. Second and furthermore, the algorithm itself should be refined to distinguish the computational cost from inter-processor communication within one application code and control both factors. Also, the load balancing module needs to be expanded to cover multiple BigJob allocations for one coupled simulation, when running a subtask across BigJobs becomes possible.

B. Two BigJobs from a Single Site or Multiple Sites

As illustrated in Figure 6, runtime environment for coupled multi-physics simulations would benefit from dynamic resource availability exploited by BigJob implementation. In brief, it is possible to assign more CPUs for a slowing application when another BigJob becomes available. Compared with one BigJob allocation for a coupled simulation in Figure 4, two smaller BigJob submissions with sizes of CFD and MD will enable faster start of coupled subtasks. Also, when next BigJob is also allocated to this coupled simulation, total job size will be the same as one BigJob submission. In a word, resource allocation and simulation flow is identical to the conventional job submission, but inactive idling in a conventional case disappears as a BigJob framework starts coupled subtasks with smaller

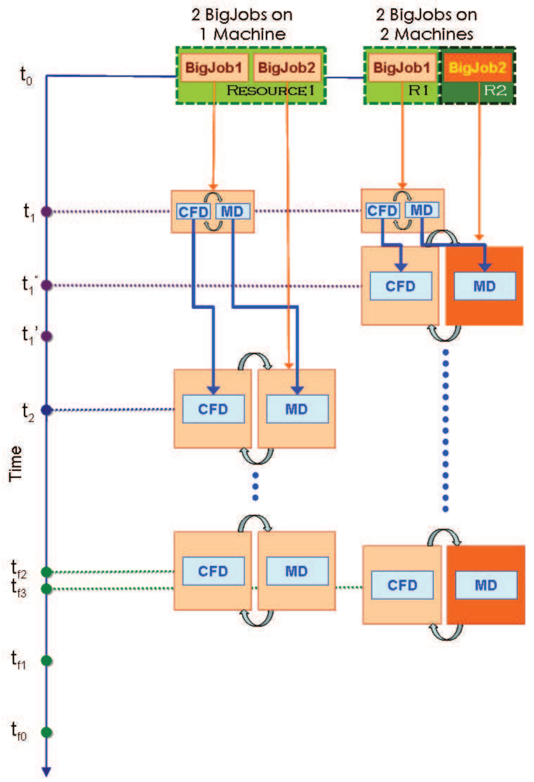

Fig. 6. Schematic comparing the distribution of simulations when 2 BigJobs are used, first on the same physical machine, then on different (distributed) machines. In both cases, coupled sub-jobs start running when the first BigJob is allocated at $t = t_1$ and experience resource reallocation with increased amount when the next BigJob becomes available. When two BigJobs are allocated, each sub-job occupies one BigJob and data exchange between jobs takes place across BigJobs.

number of processors when first job is allocated. So, submitting two BigJobs guarantees the reduction of total simulation time compared to conventional job submission, if all conditions are identical. If this two BigJob submission is applied on multiple sites, it is more likely to get two allocations faster than requesting two jobs in the same site. However, faster launch of two BigJobs does not guaranteed save of total simulation time, because time for data exchange between applications will also increase if distributed resources cooperate.

In Table II, test measurements for performance gains are summarized. The scenario for this testing is simple for clarifying benefits. Initially, two BigJobs are submitted to one HPC resource or two HPC resources. Once the first bigjob becomes available, two applications are submitted as two subtasks with pre-defined cpus to the bigjob, and then when another bigjob becomes available later, two subtasks for each application is reconfigured by assigning the entire cpus of a bigjob to one application. Table II shows performance gains with such a scenario in terms of MD run time. In this simple scenario, other complicated aspects such as file transfer, in particular between two distributed resources are not considered but in the future study, the overall performance will be optimized by considering them.

TABLE II. Performance measures for Use Cases 2 and 3. We take the time of the MD simulation as a measure of performance. For Test 1 and 2, we compare the time taken (for the MD simulation) before and after the availability of the second BigJob. As can be seen, the time to solution is significantly lower. This is now because the entire second BigJob is available to the MD simulation. For Test 3 and 4, the second BigJob (i.e the one to start later) is launched on a different machine from the first BigJob. For similar reasons to Test 1&2, the time to solution is lower when the second BigJob starts. This situation will win over the single BigJob scenario, despite of transfer/set-up times. For Test 1-3, the size of the first BigJob is 8px, whilst the size of the second (later available) BigJob is 16px. For Test the first BigJob is 16px and the second BigJob is 32px.

	Test 1	Test 2	Test 3	Test 4
number of sites	1	1	2	2
site names	P	L	L + P	L + P
(one bigjob available)				
number of cpus assigned	4	4	4	8
MD CPU time (sec)	1037.2	1045.2	1049.0	534.2
(two BigJobs available)				
number of cpus assigned	16	16	16	32
MD CPU time (sec)	277.21	277.63	276.8	150.4

In summary, use of a BigJob will eliminate idling operation of a subtask, which waits for another application to start running. Also, employing a load balancing function will let coupled codes use allocated resources more efficiently. Furthermore, using two BigJobs promotes the reduction of total simulation time, compared to conventional job submission. Meanwhile, launch of two BigJobs does not intend to use resources more efficiently than conventional job allocation, but intend to save total simulation time more than one BigJob test case, by utilizing more available resources. Remember that having multiple BigJob allocations is to increase the computing power for a specific simulation, not to use optimally within given circumstances.

VI. BigJob: A General Purpose Pilot-Job

We are able to submit a SAGA-based Pilot-Job to Condor in addition to TeraGrid resources – to both a Condor-G interface and a Condor resource pool. To that end we use Condor's native pilot-job (Glide-In) for Condor resources, through the SAGA Condor adaptor. This allows us to concurrently make use of high performance resources on the TeraGrid and high-throughput resources such as the Condor pools at Purdue and LONI. Having the capability to run pilot jobs on both types of resources through a unified abstract interface with varying underlying mechanisms presents new vistas for infrastructure interoperability, such as the TeraGrid and the Open Science Grid.

To demonstrate that a single SAGA-based BigJob interface works for both, we started a BigJob on Queen Bee (shared TeraGrid LONI machine), to which we submitted a MD simulation. Once the MD simulation started, the coupled CFD simulation is submitted to the Condor Pool at Purdue through the Condor Glide-In. The total time spent running on QueenBee was 5328 seconds, and the total run time on the Condor poll was 1015 seconds for a reduced number of iterations. The reduction in problem size was necessary due to the lack of a working MPI Condor universe on an accessible resource.

VII. Conclusions

Increasing accessibility to High Performance Computing and advances in software development utilizing parallelisms implemented in different granularity are, perhaps, two critical components for recent numerous successful outcomes with large scale scientific calculations in various domains. Nonetheless, it is not trivial to integrate a targeted scientific application with a resource management system that is aware of the challenges arising from the distributed computing as well as the local scheduler implemented with the local resource management policy. In this work, we report our development for efficient runtime environment targeting the coupled multi-physics application comprising MD and CFD as two coupled standalone applications. Overcoming the co-scheduling requirements and implementing dynamic resource allocation mechanism were two main goals motivating a novel development and our test runs demonstrated its potential for large scale scientific simulations benefiting scientific problems that are only tackled by a coupled hybrid MD-CFD calculation. Our development is built upon the BigJob framework enabled by the SAGA. The cycle of the development for core BigJob management system is significantly helped by using the SAGA with which the simple and consistent interface for managing HPC resources is easily achieved resulting the agile and flexible development. We tested our development for three cases and demonstrated its capability. In addition, the use cases includes the usage of the Condor-glide-in as our BigJob framework is closely related. In spite of simple nature of our test problems, already our development showed many promises for a successful large scale simulation. Some of those are i) employment of load balancing mechanism ii) advantages from implementation of dynamic allocation in heterogeneous distributed computing resources iii) simple solution for the co-scheduling requirements for the coupled tasks.

Acknowledgment

This work is part of the Cybertools (http://cybertools .loni.org) project and primarily funded by NSF/LEQSF (2007-10)-CyberRII-01. Important funding for SAGA has been provided by the UK EPSRC grant number GR/D0766171/1 (via OMII-UK). This work has also been made possible thanks to computer resources provided by LONI. We thank Andre Luckow for initial work on BigJob, Lukasz Lacinski for help with SAGA deployment (via HPCOPS NSF-OCI 0710874) and Joao Abecasis for his work on the SAGA Condor adaptors.

References

[1] C. H. Tai, K. M. Liew, and Y. Zhao, "Numerical simulation of 3d fluidstructure interaction flow using an immersed object method with overlapping grids," *Computers and Structures*, vol. 85, pp. 749–762, 2007.

[2] H. Watanabe, S. Sugiura, H. Kafuku, and T. Hisada, "Multiphysics simulation of left ventricular filling dynamics using fluid-structure interaction finite element method," *Biophysical Journal*, vol. 87, pp. 2074–2085, 2004.

[3] SAGA - A Simple API for Grid Applications, http://saga.cct.lsu.edu/.

[4] A. Luckow, S. Jha, J. Kim, A. Merzky, and B. Schnor, "Adaptive distributed replica-exchange simulations," *Philosophical Transactions of the Royal Society A: Crossing Boundaries: Computational Science, E-Science and Global E-Infrastructure Proceedings of the UK e-Science All Hands Meeting 2008*, vol. 367, pp. 2595–2606, 2009.

[5] S. O'Connell and P. Thompson, "Molecular dynamics continuum hybrid computations: a tool for studying complex fuid flows," *Phys. Rev. E*, vol. 52, pp. R5792–R5795, 1995.

[6] X. B. Nie, W. N. E. S. Y. Chen, and M. O. Robbins, "A continuum and molecular dynamics hybrid method for micro- and nano-fluid flow," *J. Fluid Mech.*, vol. 500, pp. 55–64, 2004.

[7] T. H. Yen, C. Y. Soong, and P. Y. Tzeng, "Hybrid molecular dynamics-continuum simulation for nano/mesoscale channel flow," *Microfluid Nanofluid*, vol. 3, pp. 665–675, 2007.

[8] J.-S. Lee, C. Kim, and K. H. Kim, "Design of flapping airfoil for optimal aerodynamic performance in low-reynolds number flows," *AIAA Journal*, vol. 44, pp. 1960–1972, 2006.

[9] [Online]. Available: http://lammps.sandia.gov

[10] Open Grid Forum. [Online]. Available: http://www.ogf.org/

[11] Louisiana Optical Network Initiative, http://www.loni.org/.