Developing Scientific Applications with Loosely-Coupled Sub-tasks

Shantenu Jha, Yaakoub El-Khamra, and Joohyun Kim

Center for Computation and Technology, Louisiana State University, Baton Rouge LA 70803, USA

Abstract. The Simple API for Grid Applications (SAGA) can be used to develop a range of applications which are in turn composed of multiple sub-tasks. In particular SAGA is an effective tool for coordinating and orchestrating the many sub-tasks of such applications, whilst keeping the application agnostic to the details of the infrastructure used. Although developed primarily in the context of distributed applications, SAGA provides an equally valid approach for applications with many sub-tasks on single high-end supercomputers, such as emerging peta-scale computers. Specifically, in this paper we describe how SAGA has been used to develop applications from two types of applications: the first with loosely-coupled homogeneous sub-tasks and, applications with loosely-coupled heterogeneous sub-tasks. We also analyse and contrast the coupling and scheduling requirements of the sub-tasks for these two applications. We find that applications with multiple sub-tasks often have dynamic characteristics, and thus require support for both infrastructure-independent programming models and agile execution models. Hence attention must be paid to the practical deployment challenges along with the theoretical advances in the development of infrastructure-independent applications.

1 Introduction

There exist many scientific problems that are solved by the collective analysis of many independent tasks, e.g., Monte-Carlo simulations, or parameter sweeps. There also exists a large class of scientific problems that involve applications that can either be decomposed into smaller coupled sub-tasks via the of choice an appropriate algorithm [1], or are naturally composed of coupled sub-tasks. The decomposition of an otherwise monolithic application into smaller components of computation, in principle makes them amenable to efficient distribution.

In this paper, we discuss Replica-Exchange (RE) and Ensemble Kalman-Filter (EnKF) based applications, as representative prototypes of applications with coupled sub-tasks. Although similar at some levels, they possess important differences. For RE based simulations, the sub-tasks are identical (ie. replicas), whereas for the EnKF the sub-tasks are heterogeneous. Additionally the nature of coupling between the sub-tasks in the former (regular intervals and pair-wise) is very different from the latter (irregular and a global-synchronisation point).

It is important to appreciate the difference between loosely-coupled when typically used in the context of parallel applications versus when used in the

G. Allen et al. (Eds.): ICCS 2009, Part I, LNCS 5544, pp. 641–650, 2009.

context of distributed applications. For the former, loosely-coupling is most often a reference to the application's tolerance of latency in message passing. For distributed applications loose (or tight) coupling has more context: it could be a reference to the flexibility in scheduling and placing the sub-tasks or even a flexibility in choice of resources the different sub-tasks are mapped to. Although both applications we investigate are classified as loosely-coupled, the nature of the coupling between their sub-tasks varies. It is important to appreciate The nature of the coupling of the sub-tasks, in addition to imposing constraints on scheduling and resource mapping strategies, also determines the feasibility of any speculative computing. Thus, along with the size and number of sub-tasks, the nature of coupling determines the overall development and deployment strategy.

In addition to some similarity between application characteristics, what binds the two together, is our adopted approach of developing distributed applications. The Simple API for Grid Applications (SAGA) [2] provides a simple, standard, programmatic approach to codify distributed applications such that they can be seamlessly run on any underlying infrastructure. Critically, this allows the application developer to focus on supporting the application characteristics and exploiting the relative strengths of different infrastructure whilst not worrying about adapting to the details of the infrastructure. Not being coupled to the details of the underlying infrastructure is a necessary condition for applications whose resource requirement might increase or those that want to make opportunistic use of newly available resources. In other words independence from specific infrastructure, is a necessary condition for dynamic applications to achieve the desired agile-execution models and thus be adaptive. The aim of this paper is to discuss how SAGA has been used to develop two applications with multiple sub-tasks in a way such that these applications can be deployed and executed on both distributed as well high-end machines, with a minimal, if not no-changes.

Lingering problems associated with deployment on production Grids has made the uptake of Grids challenging and unattractive to the end-scientist. In a nutshell, our experience is consistent with and indicates that one of the reasons deployment on general-purpose Grids is difficult, because Grids are comprised of many "isolated" components. We believe that this contributes to a currently unmanageable number degrees-of-freedom and failure modes. Although programming models and conceptual frameworks exists to unify the uptake of "grids or supercomputers" as required, practical considerations make this currently unrealistic and motivate the end-scientist to settle for the the solution that is often simpler to deploy. Whereas this has consequences for all distributed applications, it influences the development and uptake of *dynamic* distributed applications. Although the focus here is on utilising distributed machines, the same approach can be used for monolithic large machines. We demonstrate the validity of the SAGA approach for high-performance computing on large single machines.

2 SAGA: A Standard Programming Interface

The Simple API for Grid Applications (SAGA) is an API standardization effort within the Open Grid Forum (OGF) [3] an international standards development

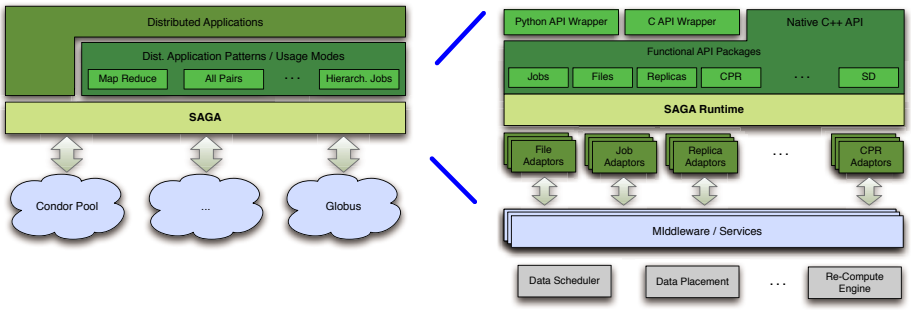

Fig. 1. Schematic diagram showing how SAGA supports the development of three simple, but important ways of developing distributed applications. Layered schematic of the different components of the SAGA landscape. The core API supports the main functionality required by distributed applications. Middleware specific adaptors make applications developed using SAGA grid portable.

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 able 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. The SAGA Version 1.0 specification defines the following packages:

- 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. The replica package support the same functionality for logical files.
- Job package - provides methods for describing, submitting, monitoring, and controlling local and remote jobs.
- Stream package - provides methods for authenticated local and remote socket connections with hooks to support authorization and encryption schemes.
- RPC package - is an implementation of the OGF GridRPC API [4] definition and provides methods for unified remote procedure calls.

The SAGA Runtime Engine can dynamically load environment specific adaptor (see Fig. 1). The two critical aspects of SAGA are its *simplicity* of use and the fact that it is a proposed standard. It is important to note, that these two properties provide the added value of using SAGA for distributed application development. Simplicity arises from being able to limit the scope to only the most common and important grid-functionality required by applications. Standardization represents the fact that the interface is derived from a wide-range of applications using a collaborative approach and the output of which is endorsed by the broader community.

2.1 Developing Distributed Applications with SAGA

SAGA can be used to develop and support distributed applications in many different ways; the exact way in which it is used, in addition to the application

characteristics, depends upon factors such as how the application needs to be used. For simplicity, in this paper, we will discuss only three different approaches for distributed application development (schematically summarized on the left side of Fig. 1). First, applications can use SAGA directly for standardised and simple distributed function calls that work on nearly all middleware systems. Typically, applications developed using direct SAGA calls are *explicitly* distributed. Secondly, SAGA can be used to create infrastructure independent frameworks (that support patterns such as MapReduce), which provide distributed capability and which can be used by applications to be *implicitly* distributed. Thirdly, SAGA can be used to support usage modes that provide access to distributed infrastructure, such as bulk job-submission or hierarchical job-submission over different machines. For this case too, applications are typically implicitly distributed, and the knowledge/control of utilizing distributed infrastructure is left to the SAGA-based framework that supports the usage-mode. The RE application that we will discuss in the paper belongs to the third category, whilst the EnKF based application is of the first type.

3 Applications with Loosely-Coupled Homogeneous Sub-tasks: Replica-Exchange

RE [5] simulations can be used to understand important physical phenomena – ranging from protein folding dynamics to binding affinity calculations required for computational drug discovery. For RE simulations utilizing as many distributed resources as possible, is critical for the effective solution of the scientific problem [6]. Distributed RE simulations must be able to orchestrate different resources in a complex and dynamic environment. Writing such an applications is a complex task for a myriad number of reasons [7]. In the following a SAGA-based RE framework developed for molecular dynamics simulations is described.

3.1 Application Description

Even with the most powerful computing resources at the moment, straightforward Molecular Dynamics (MD) simulations are unable to reach the relevant time-scales required to study conformational changes and searches. This is partly due to the inherent limitations in the MD algorithm – a global synchronization is required at the end of each time step. This limitation provides an important motivation for research into finding ways to accelerate sampling and enhance "effective" time-scales studied. Generalized ensemble approaches – of which Replica-Exchange Molecular Dynamics (REMD) [5] are a prominent example – represent an important and promising attempt to overcome the general limitations of insufficient time-scales, as well as specific limitations of inadequate conformational sampling arising from kinetic trappings. In the simplest formulation, RE is an algorithm whereby one single long-running simulation is be substituted for an ensemble of shorter-running similar simulations, but which are very loosely-coupled, ie, the interval between exchange attempts is much larger than

the interval over which the simulations run; this also make the RE formulation of physical problems excellent candidates for distributed environments.

3.2 Application Architecture

There are many architectural aspects of the framework used to implement RE simulations. However, we will focus on the abstractions that we create using SAGA that enable efficient job-submission on any underlying infrastructure. Details of the architecture and abstractions can be found in Ref. [6, 7]. Here we present the architecture in the context of an application consisting of loosely-coupled multiple sub-tasks. RE simulations can be thought of as consisting of two distinct components: the simulation engine/mechanism used for each replica process, and the orchestration-coupling mechanism between the individual replicas. Our current RE framework uses NAMD for the former and a SAGA-based framework for orchestration and coordination of the replica sub-tasks.

The developed RE framework [7] comprises of the *RE-Manager* – the central master deployed on the user's desktop, and the *Replica-Agents*, that reside on the machines where RE simulations are carried out. The RE-Manager orchestrates all replicas, i. e. it is responsible for the parameterization of replica tasks, file staging, job spawning and the conduction of the replica-exchange itself. The Replica-Agent is responsible for spawning and monitoring the sub-tasks.

In particular, queueing delays can represent a major bottleneck: a single crowded resource can slowdown the simulation arbitrary. Thus, to achieve an optimal time to solution, RE sub-tasks need to be dispatched efficiently. A common principle to prevent this is the usage of Glide-In jobs, which represent a placeholder for a set of sub-tasks (see Ref. [8]). For a Glide-In job, a sufficiently large chunk of resources is requested. Smaller sub-tasks can then rapidly be executed through the Glide-In job. Figure 2 summarizes the abstractions used within the RE framework.

While the implementation of the enhanced job model is entirely based on SAGA we can utilise other frameworks, such as the original Condor Glide-In [8]. Currently, we are actively working on a Condor adaptor for SAGA [9], which will also support native Glide-In functionality for Condor Jobs; our enhanced job model will then serve as abstraction, while the Condor level Glide-In will be used where appropriate. Irrespective of that, however, the strengths of our approach are the following: A general purpose Glide-in mechanism that does not require either Condor, or Globus and in which sub-tasks are part of a Glide-In meta-job, can be controlled at the application-level using simple *ssh* if needed. Secondly, the same mechanisms can be used to exploit distributed resources [6], as well as single high-end resources, without any changes in application code. This represents the basis of our claim of independence from underlying-infrastructure.

3.3 Application Deployment

We have shown how using the SAGA Glide-In infrastructure on multiple TeraGrid/LONI resources, the time-to-solution can be reduced [10]. Continuing with

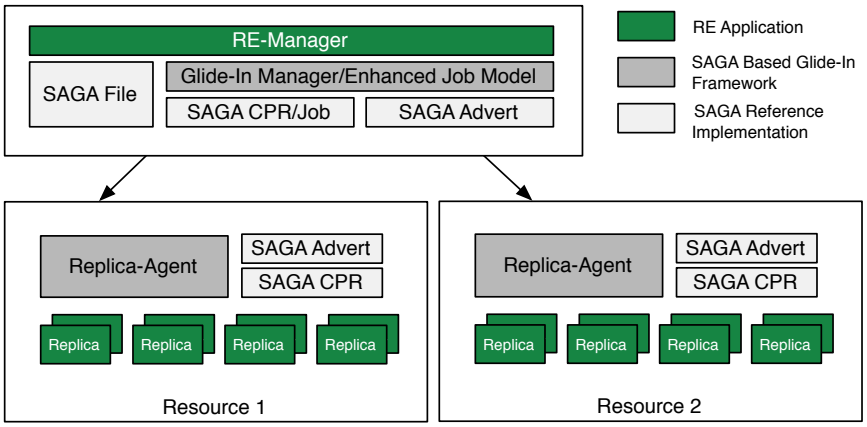

Fig. 2. Replica Exchange Framework Abstractions: The Replica-Agent is used as placeholder job for all sub-tasks running on a single cluster. The RE-Manager can control both the Replica-Agents and the replica jobs using a SAGA-based user-level job API. By using this efficient way to allocate resources, queuing times are minimized and the time to completion can be dramatically reduced when using multiple and single resources.

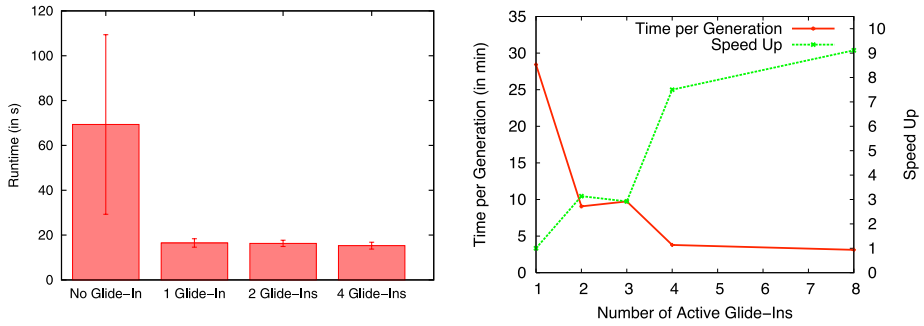

Fig. 3. SAGA Glide-In Performance: The figure shows the average runtime of a RE simulation with 16 RE processes running on 16 cores each on QueenBee. The Glide-In framework provides the possibility to effectively cluster RE jobs to receive a significant reduced time to solution (upto 80 %) even on a single machine. The plot on the right shows the number of active Glide-Ins over a six-hour run on the TeraGrid. The plot in red (using the left-hand y axis) illustrates how the average time between exchange attempts (inverse of physical efficiency) decreases as the number of Glide-Ins increases. The plot in green shows the speedup.

the theme that well developed abstractions can serve across the spectrum – distributed HPC machines (such as the TeraGrid) to single high-end supercomputers (such as Abe or QueenBee) to many smaller machines flocked together (Condor style high-throughput), we focus on using the same infrastructure to

reduce the time-to-solution on a single machine. This is also a test of the scalability of the SAGA-based Glide-In framework on a single machines.

Figure 3(a) shows that the Glide-In framework is especially beneficial if there are fluctuations in the queue-time for the sub-tasks (which is almost always!). The more sub-tasks are spawned, the more likely such delays become. While with the SAGA Glide-In framework the runtime only modestly increase with more than 8 replicas, the runtime rapidly rises when using regular Globus job for spawning NAMD tasks. The unpredictable nature of these queueing times becomes obvious by the high standard deviation found in the measurements.

Figure 3(a) shows that the SAGA Glide-In framework can provide a reduced time to solution even on a single machine by avoiding queuing time delays and fluctuations for every sub-task and allowing the efficient dispatching of RE tasks solely through the Replica-Agent. During our experiments we were able to measure speedups of up to 80 % compared to the non Glide-In approach. Fig. 3(b) shows several measures of how the use of SAGA framework results in efficient and effective deployment.

4 Loosely-Coupled Heterogeneous Sub-tasks: Kalman-Filter Simulations

Ensemble Kalman filters (EnKF) are widely used in science and engineering [11]. EnKF are recursive filters that can be used to handle large, noisy data. The data can be the set of results and parameters of ensembles of different models of a particular physical system. The ensembles are run through the KF to obtain the true physical state of the data [11], to effectively solve the inverse problem.

4.1 Application Description

In EnKF, an ensemble of forward models are run with different parameters. The data they produce is assimilated at the end of each stage, the parameters are corrected, and the models are run again. This process is repeated several times until a pre-determined criteria has been met. The ensemble of forward models are run as sub-tasks on possibly different machines, launched by a master filter task using SAGA. SAGA is also used to control the flow of data between the filter and the ensemble of models.

The variation in model parameters often has a direct and sizable influence on the complexity of solving the underlying equations, thus varying the required runtime of different models. Since we need both parameters and results for the EnKF, a mechanism to assign models to available resources based on their expected time to completion and resource requirement is useful. Such a mechanism estimates the time a model will spend in the queue of a resource, the time it needs to run, and the time required to migrate the data it requires/produces back and forth, and based on that attempt to minimize the time required to perform each Kalman filter iteration. In fact, with changing resource simulation requirements (as is the case with models that find themselves lagging behind

the rest of the model pack), a mechanism which can take advantage of faster, cheaper or more powerful machines is even more advantageous [12].

We have developed a mechanism whereby EnKF can be solved using multiple-resources, using application-level scheduling applied dynamically [13], ie mapping the sub-tasks requirement to the resources available at the instant the sub-tasks become available and ready to run, as opposed to *a priori* static method of job submission. For the problem size studied, the sub-tasks required mostly less than 32 processors. For this paper we used the earlier developed frameworks and deployed it on a single large machine – NCSA's Abe [1]. A mechanism (multiple, distributed versus single machine) that is more efficient for physical models with sub-tasks that have typically low processor counts, will not necessarily be the more efficient as the typical sub-task size increases. Therefore it is crucial that any general-purpose solution be usable on both single large machines to multiple machines. We can enhance throughput further by applying the Glide-In mechanisms discussed in the earlier section, which facilitate dynamic tasks being aggregated from similar sub-tasks. We will report on the results of this and whether the framework can be used on high-end petascale supercomputers in future work.

While concurrently running on various machines is advantageous by simple virtue of the fact that more resources would be available for running the forward models, it is also more technically challenging than running on a single machine. Authentication, job launching, multiple executables in correct paths for different architectures and file systems, and of course file transfer across the different machines are all possible points of failure. These are just some of the additional reasons why a high-level interface such as SAGA is required to hide the heterogeneity of different distributed systems. In spite of that, several challenges remain – technical, sociological as well as policy level, some specific examples of relevance we discuss in the next section.

5 Deploying on Distributed Resources

As mentioned in the opening section, using SAGA we have developed programming and execution models, whereby applications are independent of the underlying infrastructure, i.e., either use a monolithic mammoth machine, or multiple-distributed machines, depending upon the physical problem being investigated, without any modification at the application-level code. In spite of these theoretical advances, in practise end-users often find it more convenient to use single resources, even if not optimally-efficient. The smooth and effective deployment of distributed applications on heterogeneous resources remains is a difficult task. To highlight just some of the challenges of deploying advanced application on general-purpose distributed infrastructure, we mention the fact that at best 33%

[1] We wanted to use Ranger, but BQP was not available on Ranger, and would not have been before the submission of this paper.

of the resources we tried were usable (ie two in three were not usable) . We mention two problems that we encountered and led to a high amount of complexity: different library versions and broken Globus installations. Also, Globus installations on TeraGrid machines proved to be quite different. For example, the Globus GRAM2 versions on Abe and QueenBee map the RSL count element different: While on QB the count element is mapped to the number of cores, on Abe this element describes the number of nodes. Further standardization of this aspect is required in the future. The GRAM2 on Ranger (in particular the Sun GridEngine adaptor) was completely unusable due to the lack of support for MPI jobs.

Deploying our applications on ranger was not a straightforward task. SAGA requires a recent installation of the BOOST library which we had to compile ourselves. When we were finished compiling our applications, we ran into a job submission problem on a particular login node. Moving past the firewall and GRAM2 issues, getting the right certificates that are recognized on the machine, we discovered there were even more issues that need to be resolved: GridFTP was not working, the Globus/SGE script had a small error in it that had to be corrected. These issues are outlined in tickets 4957, 5111, 5130, 5145, 5172 and 5174. It is important to note that we encountered excellent response time and expert system administrators who resolved all of these issues promptly, but reiterates the complexity of utilising multiple resources from general-purpose grids. The aim here is not to criticise any provider – resource or software product, but to simply highlight the practical challenges of deploying distributed applications.

6 Conclusions and Discussions

SAGA provides the abstractions and the ability to create applications with multiple sub-tasks that can exploit multiple and different infrastructure types. We have demonstrated this via the implementation of two distinct, specific applications but both representative of a broader class of applications and running them in two different execution environments without changing the application in any way! The specific applications differed not only in the types of sub-tasks (homogeneous versus heterogeneous) but also in the nature of the coupling between the sub-tasks.

It is interesting to note that *simple, naive* implementations of these applications are possible; these would require these applications to be "grid-unaware" (or implicitly distributed). Although we don't provide details here, the real power of these applications arise from their ability to have an agile execution model, i.e., by being adaptive to dynamic resource requirements or availability. In other words, in order to develop applications that have agile execution models, more often than not, applications need to explicitly control the distributed aspects, i.e., be *grid-aware* We posit that SAGA provides an important mechanism to develop explicitly distributed applications.

Optimal scheduling of sub-tasks remains a challenge of distributed computing; however as demonstrated, for adaptive applications, scheduling can often be

done effectively at the application level. This is possible because, as shown, adaptive applications don't necessarily need tight co-scheduling, but often just lightweight-coupling between resources. This is yet another advantage of an agile-execution model.

Acknowledgement

Important funding for SAGA specification and development has been provided by the UK EPSRC grant number GR/D0766171/1. SJ also acknowledges the e-Science Institute, Edinburgh for supporting the research theme, "Distributed Programming Abstractions". This work would not have been possible without the efforts and support of the wider SAGA team. This work has also been made possible thanks to the internal resources of the Center for Computation & Technology (CCT) at Louisiana State University and computer resources provided by LONI.

References

1. Jha, S., Coveney, P., Harvey, M.: Spice: Simulated pore interactive computing environment. In: SC 2005: Proceedings of the ACM/IEEE conference on Supercomputing, p. 70. IEEE Computer Society, Los Alamitos (2005)
2. SAGA, `http://saga.cct.lsu.edu`
3. Open Grid Forum, `http://www.ogf.org/`
4. GridRPC, `http://forge.ogf.org/sf/projects/gridrpc-wg`
5. Sugita, Y., Okamoto, Y.: Replica-Exchange Molecular Dynamics Method for Protein Folding. Chemical Physics Letters 314, 141–151 (1999)
6. Luckow, A., Jha, S., Kim, J., Merzky, A., Schnor, B.: Adaptive Replica-Exchange Simulations. Royal Society Philosophical Transactions A (to appear, 2009)
7. Luckow, A., Jha, S., Kim, J., Merzky, A., Schnor, B.: Distributed replica-exchange simulations on production environments using saga and migol. Accepted for 4th IEEE International Conference on e-Science (2008)
8. Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S.: Condor-G: A Computation Management Agent for Multi-Institutional Grids. Cluster Computing 5(3), 237–246 (2002)
9. SAGA/Condor, `http://fortytwo.cct.lsu.edu:8000/SAGA/wiki/CondorAdaptor`
10. Luckow, A., Jha, S., Kim, J., Merzky, A., Schnor, B.: Distributed replica-exchange simulations on production environments using saga and migol. In: 4th IEEE International Conference on e-Science, Indianapolis, IN, USA (2008)
11. Kalman, R.E.: A new approach to linear filtering and prediction problems
12. Jha, S., Kaiser, H., Khamra, Y.E., Weidner, O.: Design and Implementation of Network Performance Aware Applications Using SAGA and Cactus. In: E-SCIENCE 2007: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, pp. 143–150 (2007)
13. Jha, S., Khamra, Y.E., Kaiser, H., Merzky, A., Weidner, O.: Developing large-scale adaptive scientific applications with hard to predict runtime resource requirements. In: Proceedings of TeraGrid 2008 (2008), `http://tinyurl.com/5du32j`