Developing Adaptive Scientific Applications with Hard to Predict Runtime Resource Requirements with SAGA

Shantenu Jha*†, Yaakoub El Khamra*, Hartmut Kaiser*, Ole Weidner*, Andre Merzky*
* Center for Computation and Technology, Louisiana State University, Baton Rouge, 70803
† Department of Computer Science, Louisiana State University, Baton Rouge, 70803

***Abstract*—There exist a large class of applications that have irregular execution characteristics and highly variable resource requirements which are very difficult to predict in advance. Not only is the development of such applications difficult, but the effective deployment of such application remains a challenge. This paper discusses the design and development of a prototype ensemble Kalman filter application, which has varying resource requirements and execution characteristics. Even for relatively small input problem sizes at each stage, up to several hundred models are generated which require from one to sixteen processors to solve efficiently; however, the distribution of the number of jobs and the run time-to-completion for a given processor count varies by up to an order of magnitude. As a consequence of irregular execution characteristics and dynamic resource requirements, it is difficult to predict effective resource mapping *a priori* and to use static scheduling techniques. We demonstrate how the use of appropriate programming abstractions like SAGA and Cactus enable run-time resource selection based upon the application specific characteristics and the effective development of applications with non-trivial requirements. As proof of concept we deploy our application on the TeraGrid and show the effective and coupled utilization of several heterogeneous resources.**

***Index Terms*—eScience Application, Middleware Interoperation, Distributed Infrastructure Deployment, Distributed Application Programming, Cactus, Performance, SAGA, Programming Abstractions**

I. Introduction

A key impediment to accelerated development of Grid applications and consequently the uptake of Grids is the scarcity of high-level application programming abstractions that bridge the gap between existing Grid middle-ware and application-level needs. Application developers are daunted by the complexity of the vast array of low-level Grid and distributed computing software APIs that currently exist; APIs have traditionally been developed using a bottom-up approach that exposes the broadest possible functionality. Coding using these bottom-up APIs requires extremely verbose code to implement even the simplest of application-level capabilities. Many Grid computing projects [1], [2], [3] recognized the need for higher-level programming abstractions to simplify the use of distributed computing for application developers. The Simple API for Grid Applications (SAGA) attempts to consolidate community effort and make ends meet by employing a top-down approach to distributed computing software infrastructure. SAGA is the first comprehensive attempt to provide a programmatic approach for the development of applications so as to utilize distributed environments – either by design or by virtue of deployment. In addition to simplifying the programming environment for application developers, SAGA insulates applications from technological, version changes and other implementation specific details that regularly occur in the lower layers of the software stack.

By definition Grids are characterised as dynamic and heterogeneous environments. They are dynamic due to time-dependent resources loads, availability and access patterns; the aggregation of specialised resources in different administrative domains is one source of the heterogeneity. Most applications and support-tools for distributed applications assume that the individual "jobs" during the course of their run life-time will display a well-defined, static execution characteristics, i.e., a well-defined and fixed underlying resource requirement and access pattern. For example, there exist several well known examples of distributed applications which in turn have multiple pleasingly distributable jobs (or sub-applications), that are either typically uncoupled or the coupling is between the master and client application.

There exist a large class of applications for which the individual job run time characteristics are inherently difficult to predict and plan for. In addition to Kalman filters [4], [5], there are a class of "first principles" Grid applications such as GridSAT [6] and applications which are based upon resource aware "learning" algorithms [7], for which it is both difficult to estimate precisely the resource requirements while explicitly needing to marshal the distributed resources. For these applications the resource requirement is dynamic and unpredictable; interestingly, the resource requirements and utilization might possibly be dependent upon both the execution trajectory and underlying infrastructure. It is difficult to define a scheduling strategy that will be effective throughout the complete execution of an applications; hence static resource mapping is not an option. It can be argued that the logic for adapting to resource requirement changes from within the application are best addressed at the application level. Interestingly the variable execution pattern and resource requirement is accompanied by, though not casually related to, a horizontal coupling between the distributed jobs, say for example, a randomly occuring global (or even local) exchange points or global synchronization points. In general, such applications are hard to develop and deploy; but in spite of their importance, surprisingly they have traditionally not received the same level of infrastructural support for the development.

There has been impressive work in the area of application-level scheduling [**?**], [8]. It is important to point out that with earlier efforts such as that in the AppLeS project, the logic is encoded in the agent implemented by the AppLeS developers and that too just for a fixed set of distributed patterns (Master-Worker and Parameter-Sweep). However, there are a much

richer class of programming patterns that applications can be classified under and it is imperative to support the wider set. Also as pointed out in Ref [8] it is not easy to programmatically adapt the application performance characteristics. The approach using SAGA we adopt in this paper attempts to be the first to provide a programmatic ability for a general class of applications from a wide range of distributed patterns.

The TeraGrid has definitely provided a massive increase in the computing power available for scientific applications, but this is due to the individual parts, and has nothing to do with the sum of the parts, i.e., the increase is not nearly enough because applications are able to use distributed resources in a coupled or coordinated manner. The future of the TeraGrid [9] is at best uncertain; this is in part due to the fact that the TeraGrid has had limited success in engendering novel applications or usage modes. The problem is however not localised to the TeraGrid; there exists a general inability to provide the platform for any new distributed programming paradigm or novel application class. For example, it remains difficult to harness more than one resource at other high-end distributed infrastructure, such as DEISA for the purposes of a single application.

The aim of this paper is to provide a novel and rare example of an approach of developing programmatically an application that can utilize the individual resources of the TeraGrid in a coupled and coordination fashion, and not just a single piece of big-iron. We have captured the primary run-time characteristics of an interesting and common class of applications and implemented a solution to effectively manage this run-time complexity at the *application level*. Our solution is perfectly acceptable as a proof-of-concept, and modulo deployment and resource availability issues, our approach will scale to problem sizes encountered in the many science and engineering problems with hard-to-predict run-time characteristics.

The specific application that we investigate is a prototype implementation of an ensemble Kalman filter using SAGA and Cactus. It is a particularly interesting case of an application with *irregular, hard-to-predict run time characteristics*, in that while conceptually simple, it captures most of these features. The model needs to run to completion which is defined as convergence to within a certain value. The run-time of each model is unpredictable and is uncorrelated with the run-time of other models running on the same number of processors, i.e., a truly independent variable. Furthermore, at every stage, each model must converge to within a specified value before the next stage can begin, hence dynamically load-balancing so as to ensure that all models complete as close to each other as possible is both a desirable and required goal. The number of stages that will be required is not determinable *a priori*. In the general case the number of jobs required also varies between stages. Figure 1, shows a schematic of the irregular and hard to predict run time characteristics of the simulations.

The main highlights of this paper are:

Utilize the advantages of proper programming abstractions, by integrating SAGA and Cactus and demonstrate the usefulness for Grid application development: Cactus provides the support and features required to implement adaptivity e.g., checkpoint, restart and migrate thorns. SAGA provides the ability to implement these features in a specific distributed environment, for example, move files from location A to location B, start a job on resource X from resource Y. Thus the two programming abstractions that Cactus and SAGA provide are natural complements of each other. As alluded to, we do so by interfacing SAGA with Cactus and thus are able to draw on the many advantages of using SAGA function calls from within a Cactus application. The result is an application that uses these two important application-level frameworks and abstractions to create an explicitly distributed application. Importantly, although we focus on a prototype of a specific application – a Kalman filter – thanks to the architecture and abstractions used, similar functionality can be trivially incorporated in more sophisticated and complex applications. SAGA provides a high-level programming interface to Grid functionality, and thus presents arguably, for the first time ever, the ability to develop complete and sophisticated applications using simple Grid function calls. This paper demonstrates the utility of SAGA for creating general applications that can perform across dynamic and heterogeneous infrastructure.
The development of an application that is adaptive in multiple ways: A working definition of adaptivity can be given as following: an application that can respond to a change in the run-time environment and manage internally (i.e., without external user or control intervention) the associated change of state and control-flow arising from the change(s) in the run-time environment. For some applications, adaptivity as defined here would be just an *interesting* feature to have, (i.e., arguably of academic interest). As we will show, for applications that have irregular run-time characteristics, adaptivity is a *critical* feature, i.e., a necessity. For example, once a job – here representing one of the application models – has been mapped to a resource, it is likely that the same model will have to be mapped to another resource before the model reaches the desired convergence. Thus for such applications, it is difficult to statically (re-)schedule or manually control upfront the execution of the jobs through the entire life-time of the application.

Adaptivity may arise due to the number of processors required by the many jobs being a variable, as well as the fact that the number of stages (i.e., involving global synchronization and subsequent distribution) is unknown *a priori*. There comes a point when providing the mechanism (logic and control) to the applications to respond to these changes becomes more efficient and reliable than leaving it to a scheduler (or a meta-scheduler for that matter). The prototype application studied is well beyond the transition point where static schedulers are no longer useful and is a classic example of an adaptive application.

The ability of a Cactus application to migrate to appropriate resources, based upon network characteristics was

demonstrated in Ref [10]. We extend the sophistication of adaptivity and the logic used for resource selection to include: migration to better resources based upon both compute and network characteristics, as well as the ability to choose optimal resources based upon local queue characteristics. The correct abstractions and programming approach enable the simple codification of an application that can determine the best resource to migrate to based upon all of the above.

Automated Resource Selection at Runtime: The Batch Queue Predictor (BQP) [11], [12], is traditionally used for determining the status of queues, resources availability as well as static resource matching. What is enticing about BQP is that it provides information that is of importance to the class of applications that we are interested in, namely, the ability to predict with certain probability the resource and the time a job – specified by number of processors and estimated runtime – is most likely to finish. This is typically harder to predict correctly than which resource a job with specified characteristics is most likely to run first. *This is the first application that we are aware of that uses BQP internally within an application to make resource selection decisions dynamically (at run-time, as opposed to static queries) and automatically (the logic of how to process BQP information is embedded in the application).*

Effective Deployment of a non-trivial coupled resources on the TeraGrid: Our application is able to use multiple TeraGrid resources. Based upon presentations by TeraGrid Directors (Catlett and Skow), the distribution of applications usage modes on the TeraGrid indicate approximately 10 applications (ie less than 1% of all applications) actually use the resources in any sense or coupled-fashion; this could be either as MPICH-G2 jobs, or explicitly marshalling more than one resource during the course of a simulation [1]. Our application is a high-end scientific application which uses SAGA to provide the ability to couple resources – either at the application level (via common namespace, logical files etc) or at the infrastructure level (ie middleware distribution specific adaptors enable the same functionality across different distributions).

The outline of the paper is as follows: In the next section we outline the scientific motivation and then describe the details of the components of the application that we have developed to use SAGA and Cactus. The next section provides details of the algorithm that the application uses to spawn itself onto a set of resources. In Section III, we discuss the architecture of the application. Section IV provides a brief description of the heterogeneous infrastructure (test-bed) that we use to deploy this application. Finally we present the data collected during real runs by this application and some ensuing simple analysis in Section V.

[1]It is important to note that this number has essentially remained level for several years, indicating the lack of new applications and usage modes. A significant fraction uses Gateways (portals) and a large number of applications use workflow tools

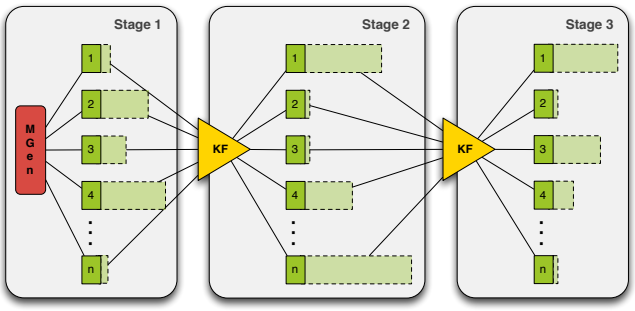

Fig. 1: Schematic illustrating the irregular execution or hard-to-predict run-time characteristics of a prototype implementation of an ensemble kalman filter. The end-to-end application consists of several stages; in general at each stage the number of models generated varies. In the specific case studied in this paper, the size and granularity of the models varied within a stage. Consequently for any given stage the resource requirements varied from 8 processors to 64 processors. The run time of each model was unpredictable and uncorrelated with the run-time of models on running on the same number of processors – a truly independent variable. At every stage, each model must converge to within a specified value before the next stage can begin, hence dynamically load-balancing so as to ensure that all models complete as close to each other as possible is the desired aim.

II. Application: Description and Motivation

A. Motivation: Ensemble Kalman Filters

Ensemble Kalman filters (EnKF) are widely used in science and engineering [4], [5], [**?**], [**?**], [**?**]. 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 Kalman filter to obtain the true physical state of the data [4], [5].

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 would estimate 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 of faster, cheaper or more powerful machines is even more advantageous [10].

B. Application Outline

The aim of our model application is to show the potential and ease of use of a SAGA-enabled Cactus framework application. Our application consists of a exemplary distributed simulation that uses the added SAGA functionality

to dynamically determine its ideal migration target based on ad-hoc and statistical network characteristics and to migrate itself in a heterogeneous Grid environment. Although this a model application it can be easily adapted to more complex scientific applications. Furthermore, our model application can be used as an autonomous benchmarking agent for Grid resources. In this section we briefly describe SAGA and the Cactus framework and discuss our motivation to use SAGA to incorporate high-level Grid functionality into Cactus.

1) SAGA: The Simple API for Grid Applications (SAGA) is an API standardization effort within the Open Grid Forum (OGF) [13] an international standards development body concerned primarily with standards for distributed computing.

The specification and implementations of the SAGA API has been guided by detailed examination of the requirements expressed by existing and emerging distributed computing applications in order to find common themes and motifs that can be reused across more than one use-case. In general, SAGA embodies the most commonly required features derived from a broad survey of the community and provides the most common grid programming abstractions that were identified by several use cases. They are intended to cover the most common application-level distributed computing programming constructs such as file transfer, and job management.

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. Version 1.0 [14] of the specification has been submitted to the OGF editorial pipeline. It 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
- Replica package - provides methods for replica management such as browsing logical filesystems, moving, copying, deleting logical entries, adding and removing physical files from a logical file entry, and search logical files based on attribute sets.
- 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 [15] specification.
- 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 GGF GridRPC API [16] definition and provides methods for unified remote procedure calls.

The two critical aspects of SAGA are its *simplicity* of use and the fact that it is well on the road to becoming a community *standard*. It is important to note, that these two properties provide the added value of using SAGA for Grid 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.

The SAGA C++ reference implementation [17] was incorporated into the Cactus Code Framework in Ref [10] to provide the needed Grid programming functionality. We believe this was an important step in merging two programming abstractions to achieve an effect that is greater that sum of the parts. The SAGA C++ reference implementation is being developed in close conjunction with the OGF standard. Advert-service package which will most-likely be incorporated into a future version of the OGF standard. The SAGA C++ reference implementation comprise a complete set of local adaptors, an SQlite3 and PostgreSQL advert-service adaptor, and Globus pre-WS adaptors for file (GridFTP) and job (GRAM2) packages. We will go onto show how the application uses these features.

2) The Cactus Code [18]: Cactus [19] is a framework for high performance scientific computing designed for scientists and engineers to develop and run codes for solving complex problems. Developing code for high performance parallel machines has many challenges including scalability, efficiency (for computation, communication and input/output), portability and flexibility. Frameworks such as Cactus allow scientists and engineers to develop modules which can then be used together with modules written by other researchers to solve complex computational problems. The framework provides tools ranging from basic computational building blocks to complete toolkits that can be used to solve complex problems in astrophysics, computational fluid dynamics or other disciplines. Tools developed in the Cactus Code framework run on a wide range of architectures including desktop PC's, supercomputers and computational Grids. Cactus and its associated toolkits are publicly available for download from the Cactus Code website.

From an architectural standpoint, the Cactus Code framework consists of a central part (the "flesh") and code modules ("thorns"). The flesh serves as a module manager, scheduling the routines of the thorns and passing data between thorns. Thorns perform tasks ranging from setting up the computational grid, decomposing the computational grid for parallel processing, providing boundary and initial conditions.

3) Why SAGA and Cactus?: Because of the modular structure of Cactus, any functionality provided by a specific thorn is immediately available to any of the other thorns in the configuration. For this reason we implemented a set of new Cactus thorns providing an extensible set of functions allowing the collection of netperf [20] based network performance metrics. Additionally, the extensible nature of this set of thorns permits additional metrics for any Cactus based application in the future. To integrate SAGA functionality into Cactus, a SAGA thorn was developed that provides the basic SAGA installation information (header files, libraries etc.) to the thorns that require SAGA capabilities. SAGA provides different packages

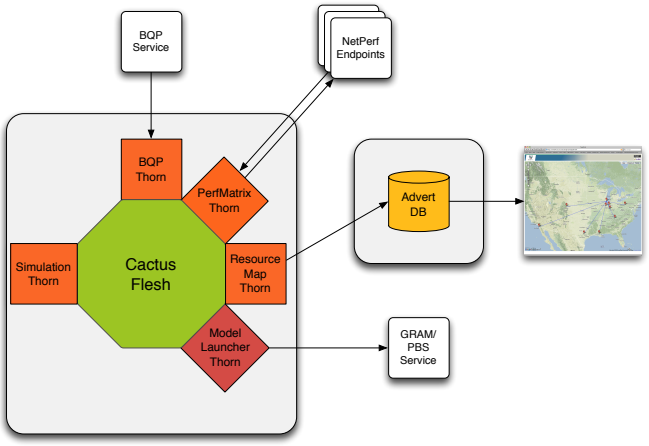

Fig. 2: The main components of the prototype implementation of Cactus based ensemble Kalman filter application. The central orchestrating component is the Cactus Flesh; each thorn – of which there are several – implements a specific, well encapsulated functionality. SAGA functionality is provided via a library and is accessed through several thorns (such as the Submit thorn). The thorns are also the interface to 'services' such as the Advert service, which in turn uses the SAGA API to interface with Google Maps. We believe this is an interesting case where an application at run-time is able to provide information dynamically to Google Maps.

with a consistent and uniform flavor, thus implementing thorns that have different functionality (performance measurement and migrate thorns) using different SAGA packages is preferable. Last but not least, using SAGA and Cactus enables applications to specify and customize the network performance characteristics that it needs; as we shall see later, the ability to do so is a very useful feature.

4) BQP: The Batch Queue Predictor [11] is a tool that is available on a number of TeraGrid resources that allows users to make bound predictions of how much time a job of a giving size and duration will spend in the queue. This prediction is given with a degree of confidence (probability) that the job will finish before a certain deadline (i.e. the time in the queue). This information is vital when submitting jobs to various machines as the longer a job sits in the queue, the longer the delay for the entire stage. For our application we use the C-API to BQP, and interface it to a BQP thorn that retrieves the predicted time in queue for jobs of given sizes and estimated durations.

III. Application Implementation and Control Flow

A schematic of the application architecture is given in Figure 2. We discuss the components that comprise the application:

A. Simulation Thorn

This thorn implements the simulation routines for the various models. Our models are mostly non-linear partial differential equations that are discretized using finite difference methods and solved with a Krylov subspace solver using PETSc. [21]. For a prototype implementation we chose a multiphase, multicomponent fluid flow through porous media simulator and implemented it in the Simulation thorn in Cactus.

B. BQP Thorn

The BQP thorn is a module in Cactus responsible for retrieving the BQP information using the BQP API. The thorn iterates over all resources in a given resource list, finds the resource and queue where if submitted, a simulation (or model) of a given size and expected run time, will have a good chance (or high confidence) of getting through the queue and start running the earliest. Confidence is in essence the probability that the job will run, and to obtain a confidence of 1.0, a high queue time is required. For our application, we found that a confidence of 0.75 works well enough.

The information from the BQP thorn is retrieved for all simulations that need to be run, particularly the time in queue. The time in queue is added to the time required to run the simulator (which is a function of the number of nodes and resource used and is retrieved from regular application benchmarks) and the time required to transfer data (that is obtained from the PerfMatrix thorn). That time would be the estimated time to completion a given models in a given Kalman filter stage.

C. ResourceMap Thorn

The ResourceMap thorn maps the various models to available resources. Each model has its own parameters that influence the size of the model and the wall time it will need to finish. The ResourceMap thorn queries the BQP thorn for the best machine and queue combination that will start the model earliest at a high enough confidence (i.e above a specified threshold). Once a machine/queue combination is selected, the ResourceMap thorn calls functions in the Submit thorn to submit the model.

Advert-service database: SAGA's advert-service package describes a key-value based hierarchical attribute interface for storing arbitrary application level information. The currently available adaptor is capable to map these structures to an relational SQL schema and store them in local (SQLite) and remote (PostgreSQL) RDBMS.

SAGA's advert-service offers a convenient way to simplify the difficult task of centralized data collection and logging within distributed applications. The location of all the simulations, the start and finish time as well as the paths to the output files are all stored in the advert-service. The advert-service also contains the data collected from BQP and from NetPerf.

D. Submit Thorn

The Submit thorn, as its name suggests, submits a job described by the ResourceMap thorn to a given resource. A parameter file for the simulator and submission script are created at the resource using SAGA. A SAGA job is then launched that performs the actual submission (e.g qsub job_1701.pbs).

E. PerfMatrix

The PerfMatrix thorn takes care of the intrinsic network performance measurement and persistent storage of the results. Currently, the implementation uses netperf [20] to measure unidirectional end-to-end throughput. Netperf is implemented as a simple client-server model consisting of two executables:

- netserver: the measurement endpoint waiting for incoming connections
- netperf: the initiator of a measurement connecting to a netserver endpoint

The PerfMatrix algorithm uses a list of computational resources which are potential migration targets for the application. After starting up, the initial application spawns itself onto all available hosts. [2] Once all jobs have been launched, the original spawning application first establishes netperf connections with all the spawned applications; this is followed by the spawned applications establishing netperf connections amongst each other, following the scheme shown in Fig. 3. The job spawning, control, and I/O redirection is done entirely using SAGA's job management package. Once a netperf process returns a throughput result, the PerfMatrix thorn uses the SAGA advert-service package to announce the result to a central PostgreSQL database which is also used as a centralized logging facility. After all netperf processes have finished and published their results, the database contains a host-to-host throughput performance matrix along with a timestamp which is available to other thorns as well as other applications.

F. Control Flow

The application starts by getting BQP data for a list of candidate machines, assigns simulations to machines and submits the jobs to the respective machine scheduler. Once all the simulations are submitted, the Kalman filter submitted to run. The Kalman filter is run on the machine that will run 50% or more of the entire simulations or the machine that has the highest bandwidth amongst the machines used for the simulations. Once the simulations start, they notify the advert service of their starting time and location; upon successful completion, they also notify the advert service of the path of the files they output. Meanwhile the kalman filter is notified of the completion of the simulations and modifies the models used in the simulation. The modified models are then submitted again as a second stage and the entire processes is repeated.

G. Interfacing with Google Maps

The application level information about the collected BQP and netperf data is stored in a central location using the SAGA advert service facilities. The stored information is usable not only by the application itself, but is accessible from separate tools allowing the progress of the application to be monitored.

[2]This approach is different from the original non-centralized spawning scheme shown in Fig. 3 The reason for our implementation using centralized spawning is the Globus Toolkit's (4.0.5) inability of full credential forwarding.

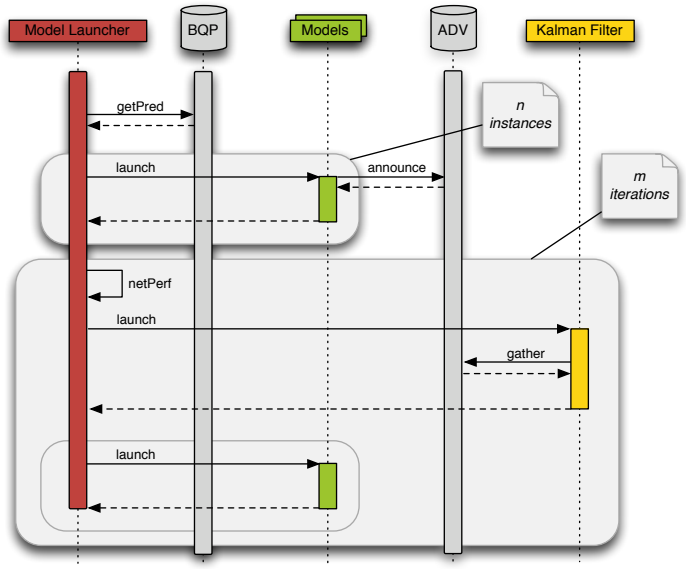

Fig. 3: A high-level overview of the control flow for the SAGA-Cactus based ensemble Kalman filter application. Models are generated and these are mapped to appropriate resources. This is done based upon the number of grid points in the model (which in turn determines the number of processors and memory) and the projected run-time (which is based upon estimated number of iterations). To a first approximation the data from BQP is utilized to determine the optimal resource – defined as that resource with the highest chances of finishing a job with the prescribed characteristics (number of processors, duration) first. Obviously this is just a best effort guesstimate, as the actual run-time (number of iterations) is difficult to get correct (as it depends upon convergence to a value). After all models (n instances) in a given stage have finished there is a global synchronization point which in turn is the basis for providing models for the next stage (m iterations).

To make this information readily available to the user, we developed a Google Maps [22] based user interface presenting live BQP data for each of the resources as shown on the map; Figure 4 shows a screen shot of this web interface displaying BQP data for the `datastar` machine at SDSC. In the near future, we will add support for displaying the netperf-based network performance information for each of the network interconnects as shown on the map, as well as extend to show the geographical distribution of sites where jobs are running and the number of jobs at any given site.

Our Google Maps application architecturally consists of two parts. The client (browser based) side is implemented using Java scripts, which is the standard way of using the Google Maps API. The server side (generating the html as requested by the client) is implemented as a Python based CGI application. We use the SAGA Python API bindings to access the BQP data stored in the advert service and to generate the html to display on the client side. Whenever the user clicks on one of the markers or lines in the map we request the corresponding information from the server by invoking a server side Python CGI script. The returned html is displayed in the Google Maps info window pointing to the marker or

Fig. 4: Google maps application showing the live BQP data retrieved from the central advert service using the SAGA Python bindings (http://fortytwo.cct.lsu.edu/teragrid/teragrid.html).

line the user clicked on.

IV. Infrastructure Used & Deployment Details

A. Infrastructure

For this trial run we used a pool of TeraGrid machines: Cobalt, SDSC IA-64 and Queen Bee (QB). The main application starts on Cobalt, runs the model generator and retrieves the BQP data for all three machines. Since QB is a new and big machine and the least loaded machine amongst the three, it was not surprising when almost all the jobs were submitted there. Initially, the application code based the decision on where to run the Kalman filter on netperf data gathered by the thorn PerfMatrix. After our initial runs showed that most jobs will be mapped to QB, the Kalman filter launch code was modified to take into account where most of the jobs are being run, i.e. running the Kalman filter on a different machine than 75% of the jobs would add more bandwidth cost to the entire stage. Once the jobs are finished running on QB, the Kalman filter reads in the output data and modifies the model parameters. These new model parameters form the basis of the Stage II, where the entire process is repeated as in Stage I.

With changing resource requirements during a simulation (as is the case with adaptive mesh refinement) a mechanism which can take advantage of faster, cheaper or more powerful machines is even more advantageous than blind migration. As the primary Cactus simulation starts, it progresses through the schedule of the routines in the configuration. One of these routines regularly checks for the time left for the simulation as well as the the time left for other simulations, as this information is kept at the advert service. If a simulation finds itself lagging from the pack, it attempts to migrate to a resource where it has a better chance of catching up. While most of this infrastructure is available in the code, it has not been enabled for these trial runs as it adds more complexity and possible points of failure. [3]

[3] `http://fortytwo.cct.lsu.edu/teragrid/teragrid.html`

B. Deployment

The deployment of applications across a pool of heterogeneous machines belonging to different Grids and organizations can be a difficult task. Different versions and availability of libraries and compilers makes every single machine a unique environment. Although it is technically possible to stage-in all required applications and libraries and even to compile the application sources on the fly using SAGA, we decided to deploy pre-build binaries on all machines since this exercise is not part of this work's focus.

SAGA uses a `autoconf/automake`-based build and installation system which explicitly tests for available compilers and external library requirements. Although the SAGA engine itself only depends on the *Boost C++ Libraries* the available middleware adaptors may require additional prerequisites which can range from *PostgreSQL* to the *Globus Toolkit* libraries. The initial attempt to build and deploy SAGA binaries on TeraGrid resources greatly helped us to realize that our initial requirements were too progressive and did not work well with the rather conservative software deployment strategies in HPC environments. Based on this experience we started to modify both, the build-system and the C++ code to allow for more relaxed prerequisites which now comprise:

- gcc $\geq$ 3.4 (32 & 64 bit on i386, x86_64, and IA64)
- Boost C++ Libraries $\geq$ 1.33
- Globus Toolkit $\geq$ 3.2 (optional)
- PostgreSQL $\geq$ 8.0 (optional)

Currently SAGA is deployed in user space, but should ideally be supported at the system level due to the limited persistent disk space in the home directories and to make SAGA transparently available to every user. Due to SAGA's latest improvements a system-wide deployment should be a trivial task which has already been started on the joint LONI/TeraGrid resources.

V. Results and Discussion

To validate our approach we present our results for a three stage simulation for a *reduced input system*, i.e., a system with limited granularity. The model generator produces 200 models of varying size, which are really models of different requirements – memory and run-time duration. These models are mapped to jobs of varying node counts. For example, in Stage I of the pipeline, the 200 models are mapped such that 173 of them are run on 2 nodes (which is 8 processors), 21 models are mapped to 4 nodes and 6 models are mapped to 8 nodes. Interestingly, the 173 simulations running on say 2 nodes have a wide variation in the duration that they need to run for – as determined by a convergence criterion. For Stage I, the fluctuation of 2 node jobs was more than a factor of 2: the minimum run time was 1.68hrs, the maximum run time was 3.5hours. The fluctuations in run-time are not limited to a set of models for a given processor count within a single stage; there are significant fluctuations between the maximum time that models need between different stages too. For example, the longest running 4 node job in Stage 1 was 5.99 hours, for

stage 2, it was 6.18 hours and for Stage 3, it was 3.99 hours; which represents a factor of 2 difference (almost) in time to convergence between different stages.

Simplified as the above may seem there are a few important salient points that need to be highlighted: The models above were mapped onto jobs with different resource requirements; here the lower bound and upper bound are 2 and 8 nodes respectively. However as the granularity of the input system (problem being studied) is increased, the variation will get much larger, i.e., the upper bound would be of several hundred processors, if not thousands. These in turn will lead to greater variation between individual jobs for a given processor count within a given Stage, as well as between different Stages! The overall fluctuation would also be greater if the difference between the shortest running job (which might be say a 2 node job) and the longest run job (which might be a say a 8 node job) were considered; for this reduced model we encountered differences by factors of 4, if not more. We estimate [**?**], that we would have to scale-up the size of input system and thus simulation by factors of 10 or more; Our current *toy problem* is a proof-of-concept, yet took greater than 20,000 CPU hours; any *real problem* would require several orders of magnitude greater resources

VI. Future Work and Conclusion

Developing both applications and tools using SAGA is an effective mechanism for ensuring (application-level) interoperability across different middleware distributions — something that is arguably missing in current Grid Interoperation efforts [23]. The main challenge that needs to be addressed is application environment heterogeneity, i.e., the development, deployment and run-time variation across resources. With Grids there is the additional complications of cross-administrative or virtual organizations (VO); as our paper exclusively uses the TeraGrid, which can for all practical purposes and intent be considered a single virtual organization, we are not encumbered by any additional burden arising from using more than one VO. These are serious challenges for all applications, even those that would like to utilize more than one resource in a decoupled fashion (for example parameter sweep). However, the seriousness of the heterogeneity problem is highly aggravated when an application needs to utilize resources in a coupled manner.

The widespread availability of SAGA, and adaptors for different middleware distributions, is an important step towards the creation of distributed applications that can be universally deployed (i.e. independent of the details of the resource's middleware and configuration detail). Our experience should serve as useful input to the community – resource providers and middleware developers - to support the development and deployment of SAGA. We hope to motivate resource providers and middleware developers to support the SAGA programming abstractions and thereby help engender applications, as well as contribute to the development and deployment of SAGA adaptors.

We will in the future compare the performance of the execution of such applications using a meta-scheduler such as GridWay and the conversion of static workflows into dynamic workflows. Although the former approach may possibly be quicker for some instances, it will not be as scalable and extensible. Motivated by the scalability and general-purpose solution we have discussed in this paper, we will go on to generalise to other applications that exhibit similar characteristics [24].

VII. Acknowledgements

This work has been made possible thanks to the internal resources of the Center for Computation & Technology (CCT) at Louisiana State University. Important funding for SAGA specification and development has been provided by the UK EPSRC grant number GR/D0766171/1. SJ acknowledges the e-Science Institute, Edinburgh for supporting the theme, "Distributed Programming Abstractions".

We would also like to thank Carola Kaiser for assistance with interfacing our output to google maps. Additionally, we would like to acknowledge help from the LONI support team and CyberInfrastructure Develompent group. Y. El-Khamra is supported in part by the U.S. Department of Energy (DOE) under Award Number DE-FG02-04ER46136 and by the Board of Regents, State of Louisiana, under Contract Number DOE/LEQSF(2004-07)-ULL. Y. El-khamra would like to acknowledge Prof. Chris White, Prof. Mayank Tyagi and Dr. Xin.

References

[1] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. van Nieuwpoort, A. Reinefeld, F. Schintke, T. Schott, E. Seidel, and B. Ullmer, "The grid application toolkit: Toward generic and easy application programming interfaces for the grid," *Proceedings of the IEEE*, vol. 93, no. 3, pp. 534–550, March 2005.

[2] [Online]. Available: http://wiki.cogkit.org/index.php/Main_Page

[3] [Online]. Available: http://www.realitygrid.org/

[4] G. Evensen, *Data Assimilation: The Ensemble Kalman Filter*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[5] R. E. Kalman, "A new approach to linear filtering and prediction problems." [Online]. Available: http://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf

[6] W. Chrabakh and R. Wolski, "GridSAT: A Chaff-based Distributed SAT Solver for the Grid," *SuperComputing*, vol. 00, p. 37, 2003.

[7] F. J. Seinstra and J. M. Geusebroek, Color-Based Object Recognition on a Grid, Proceedings of the 9th European Conference on Computer Vision (ECCV 2006) , Graz, Austria, May 7-13, 2006, Springer-Verlag.

[8] F. B. et al, "Adaptive Computing on the Grid Using AppLeS," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, p. 369, 2003.

[9] [Online]. Available: http://www.teragridfuture.org/,http://www.teragridfuture.org/positionpapers

[10] S. Jha, H. Kaiser, Y. El Khamra, and O. Weidner, "Design and implementation of network performance aware applications using saga and cactus," in *Accepted for 3rd IEEE Conference on eScience2007 and Grid Computing, Bangalore, India.*, 2007. [Online]. Available: http://saga.cct.lsu.edu/publications/saga_cactus_escience.pdf

[11] [Online]. Available: http://www.sdsc.edu/us/tools/bqp/

[12] Http://nws.cs.ucsb.edu/ewiki/nws.php?id=Batch+Queue+Prediction.

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

[14] Tom Goodale et al, "A Simple API for Grid Applications," To be published as Grid Forum Document GFD.90, 2007, Open Grid Forum.

[15] Distributed Resource Management Application API, http://drmaa.org.

[16] http://forge.ogf.org/sf/projects/gridrpc-wg.

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

[18] Cactus Framework. [Online]. Available: http://www.cactuscode.org

[19] Tom Goodale et al, The Cactus Framework and Toolkit: Design and Applications, Vector and Parallel Processing — VECPAR'2002, 5th International Conference, Springer, (2003). .
[20] NetPerf Homepage, http://www.netperf.org/netperf/NetperfPage.html.
[21] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc Web page," 2001, http://www.mcs.anl.gov/petsc.
[22] Google Maps API. [Online]. Available: http://code.google.com/apis/maps/
[23] Interoperation Scenarios of Production e-Science Infrastructures, Morris Reidel et al, submitted to e-Science 2007.
[24] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky, vol. 400, no. 6740, pp. 133–137, 1999. [Online]. Available: http://dx.doi.org/10.1038/22055