**Noname manuscript No.**
(will be inserted by the editor)

# Distributed Application Runtime Environment (DARE): A Standards-based Middleware Framework For Science-Gateways

**Sharath Maddineni · Joohyun Kim · Yaakoub El-Khamra · Shantenu Jha**

Received: date / Accepted: date

**Abstract** Gateways have been able to provide efficient and simplified access to distributed and high-performance computing resources. Gateways have been shown to support many common and advanced requirements, as well as proving successful as a shared access mode to production cyberinfrastructure such as the TG/XSEDE. There are two primary challenges in the design of effective and broadly-usable gateways: the first revolves around the creation of interfaces that catpure existing and future usage modes so as to support desired scientific investigation. The second challenge and the focus of this paper, is concerned about the requirement to integrate the user-interfaces with computational resources and specialized cyberinfrastructure in an interoperable, extensible and scalable fashion. Currently, there does not exist a commonly usable middleware to that enables seamless integration of different gateways to a range of distributed and high-performance infrastructures. The development of multiple similar gateways that can work over a range of production cyberinfrastructures, usage modes and application requirements is not scalable without a effective and extensible middleware. Some of the challenges that make using production cyberinfrastructure as a collective resource difficult are also responsible for the absence of middleware that enables multiple gateways to utilize the collective capabilities. We introduce the SAGA-based, Distributed Application Runtime Environment (DARE) framework, using which gateways that seamlessly and effectively utilize scalable distributed infrastructure can be built. We discuss the architecture of DARE-based gateways, and show using several different prototypes – DARE-HTHP, DARE-NGS, how gateways can be constructed by utilizing the DARE middleware framework.

## 1 Introduction

Science-Gateways have witnessed impressive achievements in terms of the growth of the number of supporting researchers and computing time usages [**?** ]. According to the very recent report about TG/XSEDE science-gateway program [**?** ], in 2006 10 gateways consumed 100,000 CPU hours and in 2010, 22 gateways provided 40 million hours usage; by 2011 nearly 40% of users on TG/XSEDE came through TG/XSEDE gateways. Interestingly, the CIPRES phylogenetic gateway, which was notably absent a few years ago, accounted for 25 % of all TG/XSEDE users in 2011. Such rapid and broad uptake of gateways, is indicative of their potential of supporting existing and novel scientific computing practices and consequently for advancing associated scientific domains.

Based on the usage of the TG/XSEDE science-gateways in first quarter of 2011, gateways catering to biological/life sciences account for $\approx$ 35% of all cycles accessed via gateways. On the other hand, access to TG/XSEDE via gateways accounts for only a fraction of all cycles devoted to biological/life sciences, which by most accounts represented 25% of TG/XSEDE (nu) cycles used in Q1-2011[1]. Additionally, although a

Center for Computation and Technology, Louisiana State University, E-mail: jhkim@cct.lsu.edu · Center for Computation and Technology, Louisiana State University, E-mail: smaddineni@cct.lsu.edu · Center for Computation and Technology, Louisiana State University, · Texas Advanced Computing Center, TACC, The University of Texas At Austin, E-mail: yaakoub@tacc.utexas.edu · Center for Computation and Technology, Louisiana State University, Rutgers University, School of Informatics, University of Edinburgh, Department of Computer Science, Louisiana State University, E-mail: shantenu.jha@rutgers.edu

[1] it is difficult to provide such information in a consistent way as the total number of cycles available year-to-year varies, but also which discipline a proposal gets assigned to is somewhat random; thus many chemistry proposals, even

| Quarter | PIs | Current Users | Active Users | Allocations (NUs) $\times 10^6$ | Usage (NUs) $\times 10^6$ |
|---|---|---|---|---|---|
| Q3 2010 | 249 | 1,044 | 366 | 8,810 | 2,219 |
| Q4 2010 | 254 | 1,040 | 356 | 11,720 | 2,658, |
| Q1 2011 | 257 | 1,168 | 418 | 13,101 | 3,412 |

**Table 1** Life-Science usage of the TG/XSEDE over the recent quarters. The figures establish that both the allocation and the usage of life-science applications is increasing at least in proportion to the increasing resource availability on the TG/XSEDE, if not faster.

significant fraction of the biological/life sciences tg/xsede cycles are devoted to molecular biosciences, of which MD is the dominant component, only a few of the MD users actually use gateways to access tg/xsede resources. Thus, by most accounts at best only 10% of all life-science applications cycles are accessed via gateways. Given the overall success and uptake of the gateways approach, *prima facie* there are reasons to believe that if a scalable, effective and extensible capability were provided for life-science applications, this *gap* could be overcome.

The potential advantage of providing gateways for life-science applications presents a strategic opportunity. Reconciling the opportunity to support existing and emerging science gateways, with challenges inherent in utilizing distributed cyberinfrastructure [**?** ] in a scalable and extensible fashion provides the motivation for our work: to develop a simple, extensible middleware which can be used to stand-up gateways for the life-sciences so as bridge the missing capability to utilize distributed and high-performance cyberinfrastructure. As we will see, although initially motivated by life-sciences applications, the techniques and abstractions are widely applicable.

A fundamental design objective of science-gateways is to capture and support common requirements of the science scenarios, and to support them in a simple and extensible fashion by providing the right abstractions. Building effective science gateways has two major tracks: the first is the development of effective interfaces and tools that capture application requirements, e.g., expression of workflows etc, and making these accessible via community/shared accounts with proper security and credentials supported. These must be customizable and extensible to the end-user communities' needs and specific scientific requirements. The second is the integration of these interface/tools with a broad range of high-performance and distributed computational (HPDC) infrastructure to support the science requirement, but in a way that is scalable and flexible. It is important to separate these challenges, both conceptually and in practice. This objective imposes important requirements on the middleware, tools and frameworks used to develop science-gateways.

The contribution of this work is to address the former challenge via the development of middleware for Gateways so as to support the effective integration of these two different tracks. The role of effective middleware is to support maximum reusability in integrating with HPDC infrastructure. For example, it is always challenging to integrate new applications, to implement distributed data management with newly added infrastructure, whilst supporting existing and novel execution patterns; any framework that aims to facilitate integration with HPDC infrastructure must also address these challenges. Any framework used to build science gateways that provides simple and scalable integration with the HPDC infrastructure must support broad range of diverse execution patterns (i.e., commonly occurring modes of using HPDC infrastructure).

Many gateway development efforts use development frameworks such as the Open Grid Computing Environments[**?** ] (OGCE); however development frameworks such as OGCE do not support the integration with the underlying HPDC infrastructure, i.e., most gateways still enforce the tight-coupling between applications and a specific infrastructure (see table **??**), without support for flexible distributed execution. A primary reason is the additional and significant complexity in enabling applications to seamlessly utilize distributed cyberinfrastructure. Such requirements become harder to meet as the number of connected sites and heterogeneity grows. Interestingly, both the TG in its transition to eXtreme Science and Engineering Discovery Environment (XSEDE) [**?** ] and EGEE in its transition to EGI increased the number of middleware distributions they "support". XSEDE supports globus, UNICORE and GENESIS; EGI supports gLite, Globus, ARC and UNICORE.

To address missing capabilities associated with middleware using which science-gateway can be developed, we introduce the Distributed Application Runtime Environment (DARE) framework; we illustrate and demonstrate the development of a runtime environment that supports as a primary design objective

physics proposals are likely to be biological in nature. also, although we are referring to life-science applications in general information is provided under bio-molecular bio-sciences which are understood to represent a large fraction of computational life-science applications.

| Gateway Name | Science Domain | Infrastructure Used | Execution Mode |
|---|---|---|---|
| CIPRES | phylogeny | single node | multiple independent tasks |
| GridChem | quantum chemistry, MD | single node or small cluster | multiple independent task |
| GridAMP | ASTEC coupled with a parallel GE | legacy domain scientific code | pipelined HPC programs |
| Nano Hub | portal for nano-technology | large set of tools | multiple independent tasks |

**Table 2** Examples of existing TG/XSEDE science-gateways. The list is not necessarily complete. More information including URLs can be found elsewhere [**?** ]. ASTEC means Aarhus Stellar Evolution Code.

the distributed execution of applications on HPDC infrastructure. Before we present details of the DARE framework, we discuss the "D" in DARE. Although the "D" is for Distributed, a fundamental attribute of many applications – distributed or otherwise, and infrastructure that DARE is designed to support is that of "dynamism". Dynamic aspects come from both an understanding of applications/analytical requirements, as well the runtime/infrastructure characteristics. The fact that application configurations can change as a consequence of the results produced, or the analytical requirements often means that minor changes in the application (runtime) properties/results often lead to large differences in the HPDC infrastructural requirements, duration of runs etc. The HPDC infrastructure utilized often varies, and thus runtime decisions related to scheduling and resource aggregation need to be made.

DARE provides fundamental support for dynamic execution via Pilot-Jobs, in particular the SAGA-based Pilot-Job (BigJob). Pilot-Jobs provide a simple approach for decoupling workload management and resource assignment/scheduling, via the concept of utilizing a placeholder job or a container for a set of compute tasks. They provide an effective abstraction for dynamic execution and resource utilization in a distributed context. Pilot-Jobs [**?** ] have been shown to be an excellent abstraction for supporting dynamic/runtime resource allocation and selection. In fact, there are reasons to expect Pilot-Jobs as an excellent abstraction to support the formulation of dynamic applications as well [**?** ]. Pilot-Jobs are particularly well suited for distributed environments. We have shown that a flexible Pilot-Job framework can provide the ability to run many MD simulations effectively across multiple resources on the TG/XSEDE [**? ?** ].

It is important to establish the relationship between DARE, BigJob and SAGA. In order to do so, we first examine the relationship between Pilot-Jobs and SAGA: BigJob is a Pilot-Job framework that uses the SAGA API to manage files and job submission to distributed resources. DARE uses BigJob – the SAGA based Pilot-Job as the primary execution environment for tasks assigned to it. Specifically, DARE extends and provides BigJob capability (and Pilot-Jobs in general) for pre-determined and customized applications using two possible routes: (i) via template support for execution patterns (such as abstract pipelines specifying dependencies and the order of execution), or (ii) via pre-defined interactions between tasks.

As we will discuss in the next section, the DARE framework makes extensive use of Pilot-Jobs and is also designed to support distributed and dynamic applications as a primary/fundamental attribute. In this paper we establish how these capabilities can be generalized to a range of applications via an extensible and interoperable framework and provided via science-gateways, to support a range of life-science applications. The validation of DARE lies in establishing its effectiveness as an integral component for multiple life-science gateways, and by demonstrating that it can enable the utilization of the collective capabilities of distributed cyberinfrastructure such as the TG/XSEDE and LONI in a simple and extensible fashion for a range of life-science applications. In general, DARE supports the general goals of distributed applications: interoperability, dynamic execution extensibility, adaptivity, scalability (scale-up, scale-out and scale-across) – referred to as IDEAS [**?** ]. *Scale-up* – or the most common type of scaling behavior, refers to the need of a single simulations of using many cores efficiently; *Scale-out* is a measure of the ability to support the concurrent execution and management of multiple tasks/jobs respectively (either on a single resource, or across multiple physically distributed resources); *Scale-across* is raised by the need to efficiently distribute

across multiple distinct and possibly heterogeneous resources. *Adaptivity* addresses the requirement that distributed applications may have to adapt to resource fluctuation across platforms, and to the availability of dynamic data. DARE builds upon and utilizes SAGA to achieve the IDEAS objective. In Ref [**?** ], we've shown how a SAGA-based implementation of the Pilot-Job supports IDEAS.

In §2 we provide background information on SAGA, SAGA-based Pilot-Jobs and the need/role for a standards-based building block that can be used by a range of science-gateways. A primary contribution of this work is to address the latter need via design and implementation of the DARE framework. It is important to distinguish DARE from DARE-based Gateways: DARE is SAGA-based framework that supports different abstractions such as Pilot-Jobs which in turn are packaged and integrated with support for other capabilities, such as execution pattern (e.g. MapReduce, M-W etc). A DARE-based Gateway is one that uses the DARE framework to implement the coupling to HPDC infrastructure for a scientific gateway. §3 discuss DARE and the four layered architecture of DARE-based Gateways. In §4 we present examples of DARE-based Gateways that we have been developing, followed by a discussion of the novel deployment models for DARE-based Gateways on XSEDE. We also discuss data-management support within the DARE framework.

## 2 DARE: Standard-based Middleware Framework for Science Gateways

In the introduction we outlined the first of two major tracks for building effective science gateways: the development of effective user interfaces (UI) and associated tools that capture application requirements (e.g. expression of workflows). The UI must be customizable and extensible to meet the specific scientific requirements of different science and application scenarios (SAS). The second is the use of HPDC infrastructure in a scalable and flexible fashion to support as many different SAS as possible. Combining the two, it becomes the role of science gateway middleware to support effective integration of different application-specific UI tools with underlying HPDC infrastructure. Successful integration would result in end-user communities customising base-capabilities to utilize a range of HPDC infrastructure with low overhead.

Currently science gateway development requires scientists and developers to spend non-trivial amounts of time resolving everything from job submission script errors to data management problems. Simple as it sounds, to the best of our knowledge, there does not exist middleware capabilities that provide the ability to utilize different HPDC infrastructure, using a single, simple API, whilst supporting a range of application development frameworks. To address these issues, and to provide a basis for rapid science gateway development that is capable of utilizing HPDC resource, the DARE [**?** ] middleware framework, hereafter referred to as the DARE, has been developed.

Many of the critical features of the DARE framework are provided by SAGA and the Pilot-Job capability: SAGA-BigJob [**?** ]. The SAGA components provide the distributed management capabilities such as remote job submission and data movement across machines (including data archiving). A integrated layer above the resource management layer, provides the ability to support both abstract and specific instances of dynamic execution patterns. DARE allows gateway developers and scientists to focus on the unique requirements of their scientific applications, as opposed to focus on the "plumbing" of using HPDC infrastructure as well as making it easier to support application-level constructs, e.g., how to submit ensembles of simulations to several supercomputers concurrently and archive their results.

### 2.1 SAGA: Standards-based Access to the Resources Layer

The Simple API for Grid Applications (SAGA) [**?** ] is an implementation of an Open Grid Forum (OGF) Technical Specification that provides a common and consistent high-level API for the most commonly required functionality to construct distributed applications. It also provides a high-level API **??** to construct tools and frameworks to support distributed applications. The functional areas that are supported by SAGA include job-submission, file transfer and access, as well as support for data streaming and distributed coordination. SAGA provides both a syntactic and semantic unification via a single interface to access multiple, semantically distinct middleware distributions.

Key advantages to development using SAGA include, but are not limited to: (i) providing a general-purpose, commonly used yet standardized functionality, while hiding complexity and heterogeneity of back-end resources, (ii) providing building blocks for constructing higher-level functionality and abstractions, and (iii) providing the means for developing a broad range of distributed applications, and tools to support distributed execution, such as workflows, application management systems, and runtime environments.

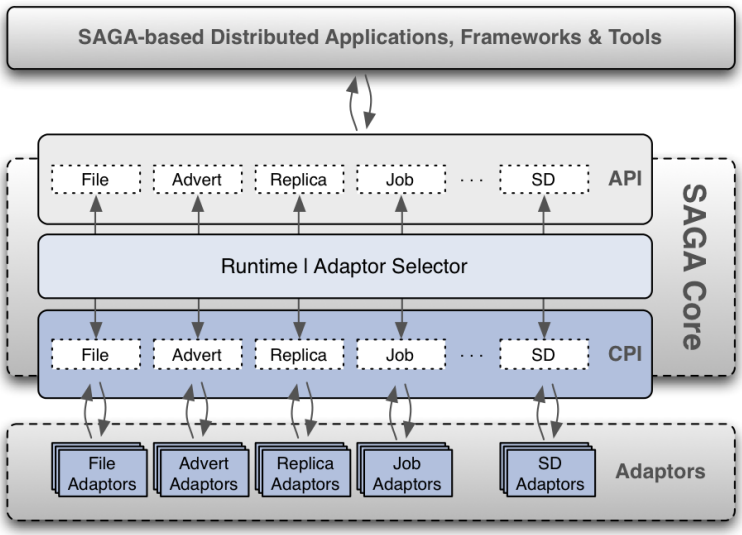


**Fig. 1 SAGA Overview:** SAGA is an OGF technical specification that provides a common interface to heterogeneous DCI – hitherto typically Grid systems. The implementation of the SAGA[**?** ] specification consists of a high-level *API*, the SAGA *Engine* providing that API, and backend, system-specific *Adaptors*. The engine is a lightweight, highly-configurable dynamic library that manages call dispatching and the dynamic runtime loading of the middleware adaptors. Each of these adaptors implements the functionality of a specific functional package (e.g., job adaptors, file adaptors) for a specific middleware system. Adaptors are also realized as dynamic libraries.

Different aspects of SAGA appeal to different groups. The standardization of SAGA as an OGF Standard is important because it makes it more likely that production infrastructures, like NSF XSEDE and Open Science Grid, will support SAGA. Having SAGA deployed and tested on these systems makes it readily available for users and developers of national Cyberinfrastructure projects. The fact that SAGA is an OGF technical specification also makes SAGA highly appealing to application frameworks, services and tool developers, which is quite understandable, as it not only simplifies their development but also makes for scalable, extensible and sustainable development. Users find the simple and extensible interface providing the basic abstractions required for distributed computing is very appealing to add their own "functionality" to a core base of functionality. Furthermore, SAGA is now part of the "official" access layer for the $121M NSF TG/XSEDE project [**?** ], as well as for the world largest distributed infrastructure EGI [**?** ].

The SAGA API provides the base abstractions upon which tools and frameworks that provide higher-level functionality can be implemented. Ref. [**?** ] discusses distributed application frameworks and runtime systems that SAGA has been used to develop successfully.

A SAGA implementation consists of a high-level API, the SAGA Engine providing that API, and backend, middleware/systems specific adaptors. Each of these adaptors implements the functionality of a functional package (e.g., job adaptors, file adaptors) for a specific middleware system. The engine is a dynamic library that manages call dispatching and the runtime loading of the middleware adaptors. Adaptors are also realized as dynamic libraries. The SAGA API has been used (in C++, Python and Java versions) to provide almost complete coverage over nearly all grid and distributed computing middleware/systems, including but not limited to Condor, Genesis, Globus, UNICORE, LSF/PBS/Torque and Amazon EC2.

SAGA is currently used on production Cyberinfrastructure in several ways. Admittedly the number of first-principle distributed applications developed is low, but SAGA has been used to develop glue-code and tools that are used to submit and marshal jobs & data across and between heterogeneous resources. Specifically, it has been used to support multiple computational biology applications that use high-performance and high-throughput molecular dynamics (MD) simulations.

### 2.2 Support for Dynamic Execution: SAGA-based Pilot-Job

HPDC infrastructure is by definition comprised of a set of resources that is fluctuating – growing, shrinking, changing in load and capability. This is in contrast to a static resource utilization model traditionally a characteristic of parallel and cluster computing. The ability to utilize a dynamic resource pool is thus an

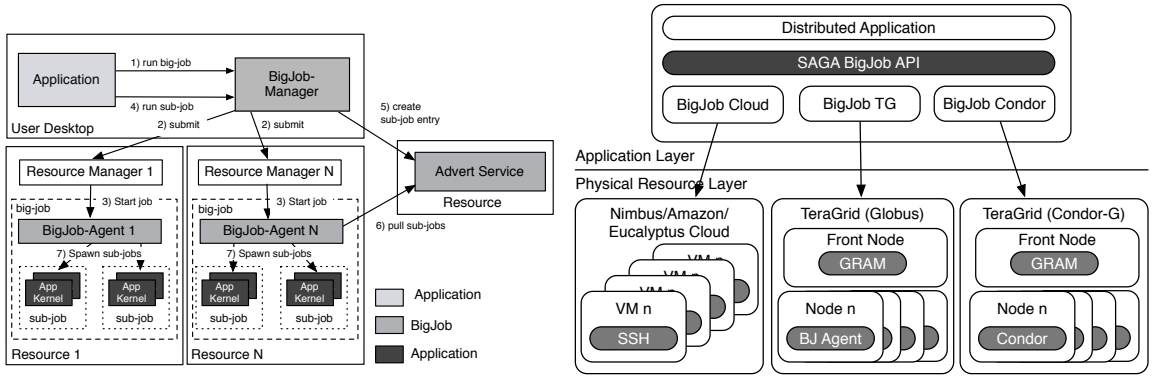


**Fig. 2 BigJob Architecture:** The core of the framework, the BigJob-Manager orchestrates a set of pilots. Pilots are started using the SAGA Job API. The application submits work units, the so-called sub-jobs via the BigJob-Manager. Communication between the BigJob-Manager and BigJob-Agent is done via a shared data space, the Advert Service. The BigJob-Agent is responsible for managing and monitoring sub-jobs. From Ref. [**?** ]

important attribute of any application that needs to utilize HPDC infrastructure efficiently. Applications that support dynamic execution have the ability to respond to a fluctuating resource pool, i. e., the set of resources utilized at time (T), $T = 0$ is not the same as $T > 0$. However, even for traditional high-performance/parallel applications, the evolution or internal dynamics of an application may vary, thereby changing the resource requirements. For example, different solvers, adaptive algorithms and/or implementations, can also require applications to utilize different set/amounts of resources. Thus, the need to support dynamic execution is widespread for computational science applications.

The SAGA-based Pilot-Job (BigJob) extends and generalizes the commonly used Pilot-Job concept. BigJob [**? ?** ] has enabled applications to use the same Pilot-Job API over disparate resources. It has also provided advantages including scalability, simplicity and extensibility. Similar advantages of scalability, simplicity and extensibility are now available to Gateway developers as a consequence of DARE – a SAGA-based library for developing science-gateway. BigJob has been used to support various execution patterns and execution workflows [**? ?** ]. For example, BigJob was used to execute scientific applications categorized as embarrassingly parallel applications and loosely coupled applications on scalable distributed resources [**? ?** ]. Pilot-Jobs are a common approach for decoupling workload management and resource scheduling, and thereby the dynamic fluctuations inherent in workloads and resources. The Pilot-Job abstraction is also a promising route to address additional requirements of distributed scientific applications [**? ?** ], such as application-level scheduling, as well as supporting dynamic application execution: some decisions can be made at runtime that affect the behavior of the simulation, the resource used, data moved and so on. The use of Pilot-Jobs as the fundamental execution unit lies at the heart of the dynamic execution capabilities that DARE support.

Figure **??** illustrates the architecture of BigJob. BigJob utilizes a Master-Worker coordination model. The BigJob-Manager is responsible for the orchestration of pilots, for the binding of sub-jobs. For submission of the pilots, SAGA relies on the SAGA Job API, and thus can be used in conjunction with different SAGA adaptors, e. g. the Globus, the PBS, the Condor and the Amazon Web Service adaptor. Each pilot initializes a so called BigJob-agent. The agent is responsible for gathering local information and for executing tasks on its local resource. The SAGA Advert Service API is used for communication between manager and agent. The Advert Service (AS) exposes a shared data space that can be accessed by manager and agent, which use the AS to realize a push/pull communication pattern. The manager pushes a sub-job to the AS while the agents periodically pull for new sub-jobs. Results and state updates are similarly pushed back from the agent to the manager. Furthermore, BigJob provides a pluggable communication & coordination layer and also supports alternative c&c systems, e. g. Redis [**?** ] and ZeroMQ [**?** ].

In many scenarios it is beneficial to utilize multiple resources, e. g. to accelerate the time-to-completion or to provide resilience to resource failures and/or unexpected delays (queue delays for example). BigJob supports a wide range of application types, and is usable over a broad range of infrastructures, i. e. it is general-purpose and extensible (Figure **??**). In addition there are specific BigJob flavors for cloud resources such as Amazon EC2 and Microsoft Azure that are capable of managing set of VMs, as well as a BigJob with a Condor-G based backend. BigJob supports dynamic resource additions/removals as well as late binding. The support of this feature depends on the backend used. To support this feature on top of various

BigJob implementations that are by default restricted to single resource use (e. g. BigJob), the concept of a BigJob pool is introduced. A BigJob pool consists of multiple BigJobs (each BigJob managing one particular resource). An extensible scheduler is used for dispatching work units to one of the BigJobs of the pool (late binding). By default a FIFO scheduler is provided.

### 2.3 DARE: An Extensible Middleware Framework

Two features make DARE unique. The first is the support for a range of (common) application execution patterns and their easy integration both with applications (above) and range of HPDC infrastructure (below). The second distinguishing feature of DARE is its extensibility along two dimensions: along the horizontal dimension, new features and support for new execution patterns can be added. In the vertical direction, support for new infrastructure can be added.

There are several common execution scenarios required for multiple life-sciences applications. These scenarios include launching ensembles of simulations or pipelines consisting of application execution followed by analysis. These can repeat any number of times and include data-management or analysis stages. We refer to these common execution scenarios as *execution patterns*. They provide general templates frequently used in life sciences applications, e.g., well known patterns such as MapReduce and Scatter-Gather. DARE also supports launching of large ensembles of loosely-coupled or uncoupled tasks. DARE's support for these common patterns is generic, and thus easily adaptable to many different applications (not necessarily life sciences). In section §3 we will discuss some of the common patterns available in DARE. The support for a patterns-based execution is a useful and novel feature of DARE.

Figure **??** shows a layered architecture of a DARE based science gateway in levels L2 and L3. Level L2 provides the application and application workflow support along with templates for common pre-configured usage scenarios: stand-alone, pipeline of multiple tools, dynamic pipeline and general workflow support. Level L3 provides the the runtime mechanisms that manage the execution of specific implementations of these patterns. In the next section, we explore this architecture in greater detail.

## 3 DARE-based Science-Gateways: Inside the Layered Architecture

A typical gateway needs capabilties for everything from authentication and authorization to the ability to submit user jobs, and data management. Ideally these capabilities are implemented without being tied to specific resources, i.e., there is a role for a middleware framework that integrates the usage of HPDC infrastructure with the capabilities required to build user interface for science-gateways. As illustrated in Fig. **??**, DARE based science gateways are typically organized into four levels: Levels L2 and L3, are common to all gateways, and are decoupled from access level (L1) and resource layer (L4). Development is therefore modular and layered. Adding support for a new resource, for example OpenStack based clouds in L4, is as simple as adding the required SAGA adaptors in L3. No changes need to be made at L1 or L2. Similarly, adding more execution patterns and workflows in L2 is decoupled from the runtime system in L3 and the resources involved in L4. We will not focus on L1, other than to say that a combination of DARE-interfaces and Pylons [**?** ] (but could equally as well be Django) are used as the building blocks to build user-interfaces that seamlessly integrate with the rest of DARE and resulting functionality.

### 3.1 Applications, Capabilities and Services Level ($L2$)

The application layer (L2) contains the necessary infrastructure to capture/express the logic necessary to create jobs of single standalone applications, pipelines and the execution of non-linear task dependencies. These dependencies are expressed using the DARE-interface alluded to, which is a customization of the Pilot-API [**?** ]. The Pilot-API provides a common access layer to Pilot Job (Compute) and Pilot Data (File transfers) frameworks over distributed production cyberinfrastructure, such as XSEDE, OSG, EGI and FutureGrid. The Pilot-API hides the complexity involved in the execution of tasks and file transfers on distributed resources; the DARE-layer provides a means for capturing the dependency and the scheduling of different tasks. The same scheduling can be achieved *directly* with just the Pilot-API, but DARE enables the auto-generation of scripts that implement these capabilities, i.e., in other words, DARE provides a higher-level capability on top of the Pilot-API. This requires the simple modification of necessary parameters in pre-exisiting templatized configuration files. This results in the creation of objects that are in turn used by the Pilot-API by combining the user customized configuration files.

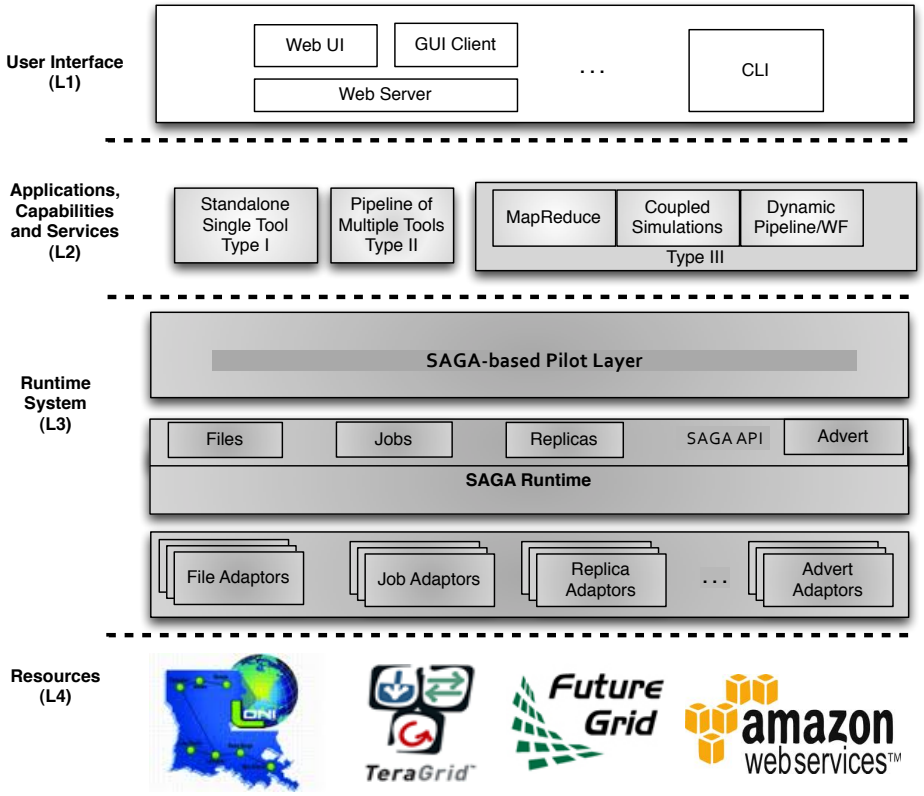


**Fig. 3** Overall architecture of a DARE-based gateway. L2 and L3 – SAGA, SAGA-based Pilot-Jobs and support for application-level execution patterns, which are shaded in grey constitute the primary layers of DARE.

These configuration files are carefully categorized into three different types for simplicity and reusablity: (i) ConfType I: Holds details about resources/machines for example job submission url, file transfer url, authentication type, (ii) ConfType II: Provides information about a tool, such as the location of a tool on different resources and arguments that should be used for tools, and (iii) ConfType III: Application instance specific information like location of input files, which series of tools (ConfType II) to use and resources as well as their size, wall time (ConfType I) etc. Using the information provided in the configuration files, DARE-level parses the scheduling of tasks, via the creation of objects. Furthermore, the tasks and input/output files will be distributed over different machines and need to be tracked. DARE supports simple dependencies between tasks as managed via the Pilot-API.

These configuration file (ConfType II, ConfType III) are combined to provide three categories of patterns available to users: Types I, II and III. Type I is the simple, standalone tool. Type II is a linear pipeline of tools and type III includes non-linear tasks dependencies, as well as complex interactions between them. The Type I/II/III classification provides a natural and convenient breakdown of available services, and was validated by its early applicability and usability for life-sciences applications. Table **??** shows some of the common life sciences applications currently being used and their respective pattern types.

| | Type I | Type II | Type III |
|---|---|---|---|
| Description | Standalone Single Tool | Pipeline or Multiple Tools | Dynamic Workflows |
| NGS Example | BFast, BWA, Bowtie, ABySS | MACS, TopHat-Fusion, TRANS-ABySS, Hydra | Novel Service for RNA-Seq |

**Table 3** Comparison of three kinds of services provided by the DARE-based gateways. The lower row provides specific examples of services implemented for NGS analytics as part of DARE-NGS.

In general, a program developed as a single standalone application needs less effort. These are typically provided as Type I service. In contrast, Type II services are built by incorporating different individual tools/applications, and linked so that the output of one is pre-determined as the input of another, i.e.,

provided as pipelines. Making a pipeline available as Type II services provides two major advantages: availability of multiple tools that are used in conjunction with each other in one location, and the enhanced capability to utilize scalable resources such as TG/XSEDE and other HPC and Cloud resources.

On the other hand, Type III services aim to support more complex service, both in the type of workflows supported as well as more sophisticated execution like the input file changes dynamically from job to job. In contrast Type I and Type II services aim to serve stand-alone and pipelined tools with advanced cyberinfrastructure capabilities, scalable production infrastructure and integrative efficient pipelines. They are however limited to static execution: once the job submission scripts are in, nothing can be changed. Type III workflows are fully customizable and extensible whereas Type II workflows are predefined and used as is. The specific resources that are utilized in the execution phase are determined dynamically – either through delayed binding or are refined/optimized after the initial resource assignment.

There are two interfaces to access DARE functionality: DARE-Wrapper and DARE-Gateways. DARE-Wrapper provides a simple command line tool named "`dare-run`" , which takes a configuration file as an argument and initiates function calls to DARE passing out the given configuration file. DARE-Gateway is a web interface, which provides a simple user interaction for the services applications level L2.

DARE uses SAGA to support job-state monitoring and notification; this capability is used to update status of submitted jobs. After submission these parameters will be stored in a database. Furthermore users can look-up the status of submitted jobs and download the output files once jobs terminate. A job monitoring thread is an important component inside DARE-Gateways. Its function is to continuously poll the database for new tasks, and whenever it finds a new task it spawns it after creating the necessary configuration files. Connecting the web interface to the server-side DARE services is made simple with the job monitoring thread. DARE uses Pylons [**?** ], and in the process out-sources authentication, database creation/management, and data storage/retrieval. The database inside DARE-Gateways is defined in such a way it could hold information about the job submitted by the user as well as the user authentication information for login and registration.

### 3.2 Runtime System Level ($L3$) and its Integration with ($L4$)

The DARE framework provides the two main features required for science gateway development: task management and data management. These features are critical for distributed heterogeneous resource utilization. As mentioned earlier, the pilot job abstraction, SAGA-BigJob, has been successfully developed for distributed task management and integrated into the current DARE framework as a key component. L3 represents the core of the DARE framework. In a nutshell, it integrates the application-level capabilities (L2, support for Type I/II/III services) with the HPDC infrastructure (resources/L4) layer. The DARE runtime level in turn achieves this via a critical dependence on SAGA. The challenges in building a scalable distributed gateway are greatly mitigated by SAGA and/or SAGA-based frameworks. The SAGA-runtime system is responsible for making middleware specific dispatches. The SAGA-BigJob (Pilot Job) supports dynamic configuration changes, thus it is relatively easy to optimize the usage of target resources for a given computational workload and their execution characteristics. Examples include embarrassingly parallel, loosely-coupled, or even multiple instances of tightly-coupled applications. Once the desired configuration has been determined, a gateway developer can easily construct an optimized runtime environment for a given target application capable of utilizing distributed resources without any application or infrastructure refactoring.

SAGA and SAGA-based frameworks enable distributed scale-out across multiple production grade resources. These production grade resources include grids of large systems such as Ranger, Kraken and Lonestar as well as HPC clouds on FutureGrid and vendor solutions such as Amazon's EC2. A basic design principle that the runtime system adheres to is future-proofing through extensibility and abstraction. When adding a new computing platform, only a small amount of code has to be written as a SAGA adaptor to enable access to that platform.

To cement support for different resources in L4, SAGA was deployed publicly on XSEDE. The installations are available for anyone and everyone to use: no private or shared accounts, no special permissions. Ranger, Lonestar, Kraken, Trestles, Steele as well as most FutureGrid machines have SAGA installations that are maintained and kept up to date by the SAGA team and supporting XSEDE personnel. Furthermore, several virtual machine images with SAGA installations are available publicly on EC2. Any science gateway developer can in principle and practice use these installations for their DARE based science gateway.

It is important to consider data movement requirements, since the movement of large data sets (input, output, and temporary files) is a major challenge. For data management, SAGA-BigData (Pilot Data) [**?** ]

| Genome Type | Index File (GB) | Resource | # of Cores | # of nodes | # of VMs | TTC (sec) |
|---|---|---|---|---|---|---|
| BG | 0.435 | R | 64 | 4 | - | 1067 |
| BG | 0.435 | QB | 64 | 8 | - | 719 |
| BG | 0.435 | R/QB | 32/32 | 2/4 | - | 919 |
| BG | 0.435 | FG | 64 | - | 8 | 712 |
| BG | 0.435 | R/QB/ FG | 24/24/ 24 | 2/3 | 3 | 1022 |
| Chr | 1.9 | R | 64 | 4 | - | 1145 |
| Chr | 1.9 | QB | 64 | 8 | - | 924 |
| Chr | 1.9 | R/QB | 32/32 | 4/2 | - | 1170 |
| HG | 127 | R | 256 | 16 | - | 9586 |
| HG | 127 | R/QB | 128/128 | 8/16 | - | 7582 |

**Table 4** Comparison of Time to Completion (TTC) required for the NGS data mapping calculations using BFAST (match step) using DARE-NGS. Three genome types, HG18 (HG), HG18-Chr21(Chr), B. Glumae(BG) represent a human genome, chromosome 21 of HG18, and the genome of a microbe Burkerholderia Glumae. Three compute resources are Ranger (R), QueenBee (QB), and FutureGrid Eucalyptus Cloud on Sierra (FG), respectively. The number of tasks for carrying out the required mapping calculation is 30 (135MB) for B. Glumae and 32 (169MB)for HG18 and HG18-Chr21. VM stands for virtual machines

is being actively developed in parallel to BigJob which is used in DARE as well. We have recently analyzed the date-volumes that need to be moved around, and demonstrated some of the challenges and solutions for BFAST [**?** ], an NGS application prototype. As we will discuss in §4, typical data-volumes for NGS applications can easily exceed 100GB; data-movement services need to and can managed these volumes. The current implementation of DARE relies on data movement solutions provided by BigData(Pilot Data). It is worth noting that SAGA supports different transfer protocols along with data-intensive programming models such as MapReduce. At the moment, GridFTP, scp and Globus-online are used as the primary protocols for data transfer. Thanks to the dynamic adaptor loading mechanism, SAGA can provide these execution patterns for emerging data intensive computing infrastructure such as clouds [**? ?** ].

## 4 EXAMPLES OF DARE-BASED GATEWAYS FOR LIFE SCIENCES

In spite of fundamental limitations on the precision of the data, amongst the many indicators of the importance of computing in the life sciences, is the fact that for most production distributed cyberinfrastructure, the proportion of resources devoted to the life sciences is already more than any discipline and the usage seems to be increasing. This is valid whether measured by number of cycles consumed, users or allocations. However, this rise has been dominated by a few well-established domains, such as large-scale Molecular Dynamics (MD) simulations, and phylogenetics. The former has benefited from the availability of multiple community codes, tailored to effectively utilize the computing infrastructure, thus enabling easy uptake/usage amongst the scientific community. The need for large-scale distributed parallel execution for other areas of life-science applications is growing. This need is highlighted by Next-Generation DNA Sequencing (NGS) which requires significant levels of computing coupled with large-scale management and storage. To reinforce the importance of this point, the first DARE-based gateway we present is DARE-NGS, and discuss scientific advantages that arise from the DARE-based gateways, as a form of validation of concept and implementation.

### 4.1 DARE-NGS

The DARE-NGS gateway is one of our signature gateways built upon the DARE framework and focusing on Next-Generation Sequencing data analytics and downstream analysis; it currently supports essential computational tasks for NGS data analytics, including multiple mapping tools such as BFAST, BWA, and Bowtie and pipelines for RNA-Seq and ChIP-Seq[**? ?** ]. Mapping (alignment) is the first step, along with de novo assembly, in scientific discovery employing NGS sequencing-based protocols, i.e., the whole genome resequencing, RNA-Seq, and ChIP-Seq. The number of available services and the capacity to handle large volume of data are still growing.

In the previous work [**?** ], using DARE-NGS, we have analyzed the performance of BFAST, a mapping program representing the latest generation of mappers. The tool stands out with its development goal for providing high sensitivity in analysis, whereas the design objective of most other popular mappers has been to achieve computational efficiency, i.e. less memory, computing time, computing resource, etc. In fact,

| | LW to QB (s) | LW to Ranger(s) |
|---|---|---|
| Avg. Rate MB/s | 112.04 ±2 | 24.7 ±2 |
| Index File FTT | 1133 | 5131.3 |
| Raw Exome FTT | 80.3 | 363.6 |
| Processed Short-Reads FTT | 34 | 153.9 |

**Table 5** Estimated File transfer times in seconds (FTT) with GridFTP between a local workstation (LW; machine named cyder located at LSU), QueenBee (QB), and Ranger for the Human Genome index files(127 GB), exome data (9 GB) and processed Short Read files (3.8 GB).

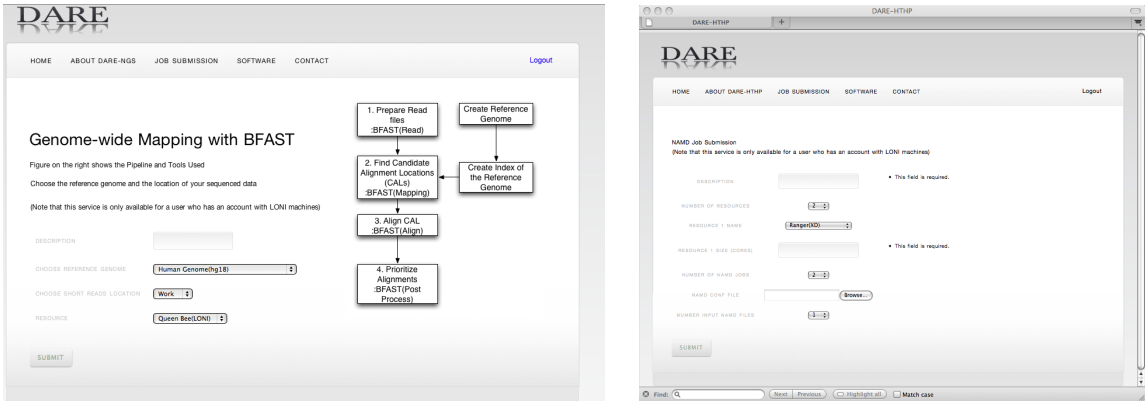


**Fig. 4** (L) The User Interface of the DARE-NGS Gateway that utilizes DARE to execute BFAST (NGS) analytics. (R) NAMD job submission web page form through DARE-HTHP. Here, the form indicates the usage mode with multiple resources. The simple case with a single resource is default, and a user can expands the form by adding more resources as shown. DARE-HTHP and DARE-NGS can be accessed at: `http://dare.cct.lsu.edu`

the computational complexity of the analysis (e.g. mapping) depends upon various elements such as the size and complexity of the reference genome and the data-size of short reads, but mostly algorithms. We characterized the computational requirements, I/O bottlenecks and degree of concurrency with BFAST in Ref. [**?** ], and concluded that an efficient, scalable and extensible analytical approach must be supported by any framework supporting NGS- analytics.

An examination of various execution scenarios with multiple grid and cloud resources was conducted and a part of obtained results are presented in Table **??**. Not surprisingly, challenges are more involved than effective distribution of computational tasks across multiple, heterogeneous resources; there is the need to manage large data movement and storage. Specifically, managing large-data volumes on FutureGrid clouds required the Walrus data storage system[**?** ], however access to Walrus through multiple VM's often failed. Such failures need to be managed, as human genome mapping requires greater disk space than is often available in many environments, including FutureGrid. Interestingly, we overcame some disk space limitations, by implementing and exploiting task-level concurrency.

Based upon the types of capabilities summarized in Table **??** and their seamless utilization of HPC grids such as TG/XSEDE and Clouds as part of FutureGrid[**?** ], DARE framework has thus been shown to provide the required data and compute management capabilities for NGS analytics and is thus an effective building block for science gateway for sequencing data analytics and downstream analysis. For data-intensive computation such as NGS analytics, file transfer processes are important for the total time-to-completion; Table **??** shows the measured transfer time. Currently, DARE-NGS employ GridFTP as a default protocol on the grids and SCP for FutureGrid Sierra Cloud. Table **??** highlights some data volumes that typically need to be transferred with data analytics for DARE-NGS with a model system (human genome).

### 4.2 DARE-HTHP

In addition to long time simulations of a single large system, an important challenge is the ability to run a large number of identical copies (ensembles) of the same system. Ensemble-based simulations are important for effective sampling and due to the low-level of coupling between them, ensemble-based simulations are good candidates to utilize distributed cyberinfrastructure. The problem for the practitioner is thus

| Number of tasks | Resource | # of Cores | # of nodes | # of VMs | TTC (sec) |
|---|---|---|---|---|---|
| 8 | R | 64 | 4 | - | 141 |
| 8 | QB | 64 | 8 | - | 176 |
| 4/4 | R/QB | 32/32 | 2/4 | - | 151 |
| 8 | FG | 64(8) | - | 8 | 414 |
| 2/3/3 | R/QB/FG | 32/24/24 | 2/3 | 3 | 384 |

**Table 6** Time to completion (TTC) for 8 tasks of NAMD (each running on 8 cores, irrespective of the underlying resources), run over different resources. The three compute resources are Ranger (R), QueenBee (QB), and FutureGrid Eucalyptus Cloud on Sierra(FG), respectively. The data shows that DARE-HTHC has been run successively on Grids and Clouds concurrently.

effectively marshaling thousands if not millions of high-performance simulations on distributed cyberinfrastructure.

Thus fundamental support for an ensemble of simulations with the following characters is required: (i) Usable on a range of underlying distributed resources and independent of resource-specific middleware and services (i.e. scale-across), (ii) Efficiently manage both scale-up and scale-out of ensembles – possibly scaling up to tens-of-thousands, if not millions of ensembles, with varying degrees of coupling, (iii) Effective for a range of physical model sizes – from thousands of atoms to hundreds of thousands of atoms, (iv) All of the above without being tied to a specific underlying MD kernel, (v) Extensible and Interoperable with emerging computational platforms such as clouds.

It is important to establish that the challenges of supporting ensemble-based MD simulations are different from traditional workflows (e.g, SCEC [**?** ]), where the emphasis is on coordination of pre-ordered tasks; by contrast ensemble-based simulations need to be designed to support the optimal scale-out and scale-up of tasks across multiple machines. In other words, in the latter, the makespan has to be reduced by executing as many tasks concurrently as possible, whereas in traditional workflows the makespan has to be reduced by the efficient scheduling and placement of ordered tasks.

In Ref. [**?** ] the ability of an *interoperable* and *extensible* Pilot-Job tool (BigJob), to support high-throughput simulations of high-performance molecular dynamics simulations across distributed supercomputing infrastructure was assessed. Specifically, a nucleosome positioning problem as an exemplar was used, wherein we computed 336 independent trajectories of 20 ns each. Each trajectory was further divided into twenty 1 ns long simulation tasks. A single task required $\approx$ 42 MB of input, 9 hours of compute time on 32 cores, and generated 3.8 GB of data. In total 6,720 tasks (6.7 $\mu$s ) and approximately 25 TB were managed. There is natural task-level concurrency, as these 6,720 can be executed with 336-way task concurrency. A scalable Pilot-Job approach was used to successfully complete this workload on the TeraGrid/XSEDE.

Building upon these results, DARE-HTHP provides an extensible middleware wrapper around SAGA-BigJob, as evidenced by results presented in Table **??** (from Ref. [**?** ]), which establish that that DARE-HTHP support ensembles of simulations across a range of HPDC infrastructure.

In discussing DARE-NGS and DARE-HTHP, we have address how DARE has been used to construct gateways, focusing on, (i) specific science capabilities supported, (ii) demonstrating the use of a middleware framework — which serves as a common-layer to efficiently integrate interface layer of a gateway with HPDC infrastructure, (iii) specific problems solved, demonstrating the unique advantages of DARE-based gateways for large scale compute-intensive as well as data-intensive services are highlighted. For example in DARE-NGS (one of the gateway based on DARE-Gateways) a web form for CHiP-SEQ application where user can choose parameters like reference genome, input files like control and treat data and the mapping tools like BWA, Bowtie, and peak calling tools like MACS, PeakSeq and submits the form.

Although the main uptake of DARE-based gateways has been for the common scientific problems of NGS analytics and high-throughput MD simulations, DARE is being used to construct gateways to address other science requirements, viz. To study RNA folding and for large-scale docking studies. What is common amongst the "family" of gateways is that they use the same DARE middleware framework, but each is customized to support different science/application scenarios.

## 5 Deploying DARE-based Gateways on TG/XSEDE

Indiana's Quarry [**?** ] has been used as a novel agile virtual hosting service for DARE-based gateways. Quarry is a gateway hosting resource consisting of geographically distributed Dell AMD systems that provide persistent, secure virtualization services. Typically gateways would run statically on dedicated hardware that is not directly connected (or affiliated) with XSEDE systems. With Quarry, gateways can be

| Science Domain | Supported Applications | Conventional Usage Mode | Distributed Usage Mode |
|---|---|---|---|
| Molecular Dynamics | `NAMD` | MPI-based single simulation run | ensemble-based multiple simulation runs |
| NGS data analytics | `BFAST` | memory-intensive single-node execution | data-intensive distributed computing |

**Table 7** Examples of life-science applications of interest and their usage modes. Gateways were developed for these applications using the DARE framework

run from gateway-developer run images hosted on an XSEDE resource with high bandwidth connections to XSEDE HPC resources as well as data repositories at TACC (Ranch), NCSA (Forge) and PSC (Albedo).

There are many advantages to agile virtualized gateway systems including automatic nightly backups, quick restores (if re-deploying after failure), reduced maintenance and so on. Furthermore, the virtualized solution has low operational costs, high bandwidth and excellent support. There is however a unique advantage to gateway framework developers: the ability to bundle a VM image containing the gateway framework for immediate, user-ready deployment. For DARE, the bundled image will contain the basic framework along with demonstration web interfaces: DARE-NGS and DARE-HTHP. If a third party user (or gateway developer) wants to use DARE, all users have to do is run the DARE image on Quarry and setup their credentials.

By contrast, most XSEDE (and formerly TeraGrid) science gateways ran statically on hardware with low fault tolerance and custom web and machine interfaces [**?** ]. In a sense, this is a one-off custom solution that every group needing a science gateway had to struggle with. Each of these gateways is a unique installation running on user side resources with custom scripts and more importantly custom interfaces to TeraGrid/XSEDE. This complicates the deployment and utilization model of science gateways: each gateway has to be a full ground-up development project and the entire development and deployment burden is placed on the gateway developers.

The deployment and utilization model for DARE shifts the development and deployment effort so that regular users (not necessarily gateway developers) can get a basic science gateway running in quick order without much difficulty. With a basic understanding of gateways, a user can start their own DARE VM on Quarry and customize the existing building blocks and interfaces to their satisfaction. This reduces XSEDE integration overhead, gateway development and deployment overhead and unifies gateway infrastructure for many applications. It is also important to note that other XSEDE sites are in the process of deploying gateway hosting services similar to Quarry, and with VM portability, DARE VM's can be ubiquitous on these resources as well. Our expectation is to see science gateways as a generic but customizable service available to users, complete with building blocks in the form of "canned" virtual machine images.

## 6 Discussion

The fact that DARE has been used to support four very different science applications is testimony to the correct abstractions and design objectives. In each case, DARE has facilitated the quick, effective and scalable integration of the scientific logic associated with the of a use-case/usage-mode (often implemented with python scripts) with the underlying HPDC infrastructure/resources. Each of these science applications and their usage-modes have been supported via both *command-line* as well as *science gateway* interfaces.

DARE supports the primary distributed application design objectives in IDEAS [**?** ] – Interoperability, Dynamic execution, Extensibility, Adaptivity and Scalability. One of these objectives, scalability (in particular, scale-across), represents the ability to utilize multiple resources; using these resources collectively has many benefits, not least of which is the decrease in the time-to-completion. As illustrated in Table **??**, DARE has been used to demonstrate such performance improvements; specifically, where two large TG/XSEDE systems – QB and Ranger, were used for the match step in BFAST pipeline. For the HTHP mode [**?** ], up to 24,192 cores were utilized at the same time, 128 concurrent simulations were managed (scale-out) and up to four different TeraGrid/XSEDE resources used. Given the difference in middleware distributions, this also demonstrates DARE's ability to support interoperability.

Functional and infrastructural extensibility are important requirements for gateways; indeed, as we have shown, many levels of extensibility are provided by DARE-based gateways. For example, the SAGA-

based approach which provide numerous adaptors to different middleware, supports interoperability across different middleware types; it is limited only by the availability of adaptors (SAGA has relevant adaptors to multiple existing infrastructure including clouds). The integration between L2 and L3 via different types of services supported via a range of execution patterns and SAGA-based pilot-abstractions, provide the flexibility to extend scientific capability of a gateway, e.g., support of ensemble-based simulations as a primitive "execution unit". Building upon the Pilot-abstractions, DARE-based gateways also support extended data-access patterns and coupled data-compute placement. Functional extensibility is required for scientific inquiry; for example, with DARE-RFOLD, we have provided a pipeline whereby the output from the RNA secondary structure prediction was utilized for Riboswitch gene annotation using structural information.

The experience of using DARE to support NGS analytics over a range of input problem sizes, infrastructure and analytical problems, (as illustrated by Table **??**), indicate that science-gateways for distributed data-intensive applications have a set of unique challenges. For example, for NGS data alignment using BFAST, not only does one need to determine the right distribution of computational resources for the given tasks/problem at hand, but these tasks need to be placed "optimally" so that data movement does not become the bottleneck. DARE supports such *dynamic* resource utilization and execution capabilities.

In conjunction with the ability to distribute tasks effectively, NGS data alignment using BFAST, requires the ability to determine the optimal task-level concurrency; this is often a function of how the reference genome indexes are utilized for chunks of short read data[**?** ]. The design of the DARE framework supports such application-level adaptivity, via support for flexible composition of L2 services, and flexible execution patterns and dynamic execution modes. As alluded to in § 2.1, application-level requirements must be matched by the runtime systems; here the runtime systems such as SAGA-based Pilot-Jobs support application-level adaptivity, e.g., change in task-level granularity, usage modes etc. Other forms of adaptivity supported by the runtime systems that DARE-based Gateways build upon, include different data-transfer mechanisms – varying from gridFTP, to reliable protocols and variants thereof (such as GlobusOnline).

It is important to reiterate, that all of the objectives as captured by IDEAS are provided by DARE using distributed computing community *standards*. The lack of need to change interfaces and development systems imparts a simplicity to the development process, i.e., SAGA is the basis for the simplicity ensuing from the well defined interfaces that supports multiple *standards* distributed functionality, as well as higher-level abstractions, e.g., based on the underlying SAGA-based API, via SAGA-BigJob a general purpose Pilot-Job abstraction.

Finally, the new virtual machine image based deployment model on Indiana's Data Quarry system enables rapid and hassle free deployment. No hardware needs to be purchased, no backups arranged, and web development. In due time, we expect to have popular content management systems with DARE infrastructure available in images for the entire community to use.

In summary, we have demonstrated via working examples, how providing the right abstractions and easy integration with other components, enables DARE-based gateways to support the effective and easy development of science gateways for life-science applications that support distributed applications, whilst adhering to the IDEAS design-objectives. This includes advanced deployment modes and novel data-intensive life-science applications that utilize emerging distributed computing/data resources such as TG/XSEDE and cloud environment contributes.

## Acknowledgments

This work is funded by NSF CHE-1125332 (Cyber-enabled Discovery and Innovation), HPCOPS NSF-OCI 0710874 award, NSF-ExTENCI (OCI-1007115). Important funding for SAGA has been provided by the UK EPSRC grant number GR/D0766171/1 (via OMII-UK) and the Cybertools project (PI Jha) NSF/LEQSF (2007-10)-CyberRII-01, NSF EPSCoR Cooperative Agreement No. EPS-1003897 (with additional support from the Louisiana Board of Regents) and Grant Number P20RR016456 from the NIH National Center For Research Resources. SJ acknowledges the e-Science Institute, Edinburgh for supporting the research theme. "Distributed Programming Abstractions" & 3DPAS. This work has also been made possible thanks to computer resources provided by TeraGrid TRAC award TG-MCB090174 (Jha). This document was developed with support from the US NSF under Grant No. 0910812 to Indiana University for "FutureGrid: An Experimental, High-Performance Grid Test-bed". We thank FutureGrid support, in particular Fugang Wang for exceptional support. We thank our fellow Radical members for their support and input, and in particular Ashley Zebrowski for providing detailed feedback, Ole Weidner for a constructive criticism of

this manuscript and Andre Merzky for the information about SAGA CSA deployment. Last, but not least, we would like to thank one of the anonymous reviewers for exceptionally useful and insightful comments, which have greatly improved the paper.