1 Project Overview and Description

In this proposal, we aim to develop programming abstractions to encourage the best programming and the best performance of data-intensive applications on the Google/IBM cluster and other distributed infrastructures. To do this, we will use our experience with the design, development and deployment of SAGA [1] (a high level interface for creating distributed applications) and our experience with three selected data-intensive applications: BLAST [2], probably the most commonly used bioinformatics application; LIGO [3] data analysis, the work done to search for gravitational-wave signals in the data produced by the LIGO detectors in Louisiana and Washington; and Montage [4], a standard astronomical image mosaicking application also used in a wide variety of computer science projects.

SAGA was initially designed for and evolved with compute-intensive applications in mind; it provides many of the application level abstractions that are required to develop and deploy computationally intensive applications. Recently, we have implemented the MapReduce [5] (with support from Google) and All-Pairs [6] programming abstractions to provide frameworks – which in turn are being used to rewrite bioinformatics applications such as sequence alignment and searches in ways that are not conventional. We have also been involved in a theoretical study of distributed applications and the abstractions that can be used and are required to support commonly occurring patterns in distributed applications [7].

By integrating SAGA with Google's App Engine and by providing the relevant SAGA back-end interfaces to Google's BigTable, users will be able to use SAGA on the Google/IBM Cluster without any explicit change to the operating environment. Additionally we will extend our deployment to work with locally provisioned cloud-clusters using Eucalyptus [8], thus arguably providing for the first studies in Cloud-system interoperability from an application's perspective.

Our approach is to: 1) Extend our theoretical understanding of what commonly occurring data-access patterns for data-intensive distributed applications are; 2) Define the abstractions – both programming and system – that can be used to support these patterns; 3) Incorporate the necessary extensions in SAGA to enable it to support data-centric patterns and abstractions thereof; 4) Write SAGA adaptors for GFS, BigTable, and the Google/IBM cluster system, so that these programming patterns, become available to applications in that environment; 5) Re-architect and develop the three sample applications using the extended SAGA; and 6) Deploy and test them on the Google/IBM active storage cluster in conjunction with a locally operated cluster with active data management and policies.

1.1 Project Description

We have led the design and development of SAGA [9], a high-level interface for the effective development and deployment of distributed applications, and we have also been involved in the use of SAGA for novel application development [10, 11]. An important strand of our recent work with SAGA has been in the domain of data-intensive applications, such as our recent implementation of frameworks that provide MapReduce and All-Pairs programming abstractions. We have used the MapReduce framework to find the number of occurrences of a gene in a given genome and the All-Pairs framework to do multiple alignment [12]. The implementation of MapReduce using SAGA has been supported by Google (GSOC-08) [13]. The SAGA-based MapReduce framework supports general purpose use of the MapReduce programming model. We plan to investigate improvements in the efficiency of the framework over a range of applications, by using a general purpose "active data" framework such as Bitdew [14]. SAGA will be interfaced with frameworks that provide these "lower-level" APIs and the resulting system will be used for experimenting with various data-scheduling and placement approaches. As part of the Google project, we are also implementing the ability for SAGA to work using Hadoop [15] and BigTable [13, 16]

Our work as part of the funded research theme on Distributed Programming Abstractions [7], has lead to arguably the most comprehensive analysis of distributed applications [17], how they are programmatically constructed, what patterns commonly occur, and how they could possibly be supported. The work so far has focused on the general class of distributed scientific applications and not on data-intensive applications. Concurrently, we have been investigating theoretical ideas behind how clouds (and thus by generalization, clusters such as the Google/IBM cluster) differ from traditional distributed systems, by focusing on system semantics – importantly what the "restricted capabilities" of clouds imply for application usage modes and programming interface [18].

We propose to use three of the most challenging, important and commonly used data-intensive applications to address questions about programming abstractions and models, as well as scheduling algorithms and access patterns. Specifically, we will extend our theoretical understanding of what the commonly occurring data-access patterns for data-intensive distributed applications by continuing to analyzing common data-intensive applications as well as not so common data-intensive applications. We have carefully chosen our applications – BLAST, Montage and LIGO data analysis – from three distinct applications classes in an attempt to capture the critical characteristics of data-intensive applications. It is only by the analysis of real scientific applications that we will be able to discern possible other patterns. There are many patterns, akin to the Berkeley Dwarfs/Motifs [19], such as fan-in/fan-out [20], waiting to be discovered for data-intensive applications. Finding such patterns, those that are common across a range of applications, has not been done before. Once we have an understanding of the patterns, we will seek to understand the programming and system abstractions that can support them.

Our next step will be to incorporate the required extensions in SAGA in order to provide the necessary interfaces to support the abstractions, which in turn will be designed to support the patterns we derive. We will create a suite of the most common patterns that we derive and will use SAGA and its extensions to actually implement these patterns. In order to understand better which programming models work for data-intensive scientific applications, it is also important to be able to conduct empirical studies with the ability to select programming models and abstractions to support them, along with, but independent of the ability to choose different underlying protocols/scheduling/cluster environments. Applications using our SAGA-based framework will find precisely such an opportunity.

MapReduce is simple and powerful, and clearly the best known example of such a pattern, but will be only one of the many needed programming models for data-intensive applications. Our stand-alone implementation [21] is a proof of concept of SAGA's ability to support commonly occurring patterns via *infrastructure independent implementation*; note, Google's MapReduce is tied to the Google File System. This forms the basis for our claim that SAGA can be used to implement abstractions other than MapReduce which are usable over a range of infrastructure.

We have insufficient detail on the Google/IBM cluster to surmise if it will be the right architecture for a range of scientific problems and if so, for which. By providing the mechanisms such that our research efforts can utilize a range of environments – Google/IBM, EC2, Eucalyptus-based – we will be able to determine the right programming model for a given environment. A simple conclusion such as, the Google/IBM environment is not good for LIGO is not very interesting or insightful. Thus, figuring out the right environment for a range of applications is part of the research agenda. Unfortunately, there are limitations on the extent of interoperability between different Cloud systems; current limitations in supporting interoperability arise at both the system level as well as at the application level. Although interoperability is not a primary goal – for the lack of interoperability with current set of applications is at best an irritant, not a performance or usage hindrance – interoperability emerges as natural and desirable by-product of our approach. This will be important for the next generation of applications, as well as those explicitly considered in this proposal.

SAGA provides the basis for implementing programming models for data-intensive applications, irrespective of where those applications actually run – Google/IBM, EC2 or a Eucalyptus-based environment. The application developer decides on the programming model to be experimented with – say MapReduce for a genomics- application and All-Pairs for a search-based application – codes it up using SAGA, or as is more likely, uses a SAGA-based MapReduce/All-Pairs framework, and tries to determine the best infrastructure to run on. SAGA enables the application developer to decouple the programming model from the environment.

Our experiments will span locally-provisioned cloud clusters running Eucalyptus, thus providing an in-house EC2 environment (and without the monetary cost of paying Amazon for cycles/data transfer), while also using the Google/IBM cluster. An additional advantage of our SAGA-based approach and the ability to have local environment that interoperates with the Google/IBM cluster is that among other things, this will provide for the first studies in cloud interoperability from an application perspective. Rich Wolski (UCSB) and the developers of Eucalyptus are committed to supporting this effort.

We will focus on one very commonly used application – BLAST, but in its distributed and parallel incarnation mpiBLAST, as outlined in the ParaMEDIC project [22][1] – and on two well known data-intensive applications – Montage and LIGO. Montage and LIGO have not been rewritten for or deployed on clusters with active storage; before we can aim to get the desired performance, we need to re-architect these application in order to take advantage of the underlying "architecture". All three applications will be re-architected and developed using the extended SAGA, and then will be deployed and tested on the Google/IBM active storage cluster in conjunction with a locally operated cluster with active data management and policies. These three applications contain differing computational features. mpiBLAST has the simplest structure, which can be thought of as distributing data, doing processing, and collecting results. LIGO data processing is somewhat more complicated, adding to the challenges of mpiBLAST additional issues related to pipelining and parameter sweeps. And finally, Montage is the most complex, as its structure really forms a general many-level DAG.

We recognize that many of the important challenges in computing involve data movement, where coordination is needed to ensure that the data are in the right place when the compute resources are ready to use them. This is true at all levels, from within a chip to across a network; however, it is the regime in between these two extremes – data access to/from local store by applications – that is of greatest challenge to scientific application programmer and which is both the focus of this proposal as well as beneficiary of better programming models and abstractions for data-access patterns.

A significant research outcome of this project will be a general purpose tool (extended SAGA) for addressing many of the questions related to programming models and abstractions for data-oriented computing that arise in the context of this solicitation. The second major research impact will be that three important applications will have been re-architected and programmed so as to be able to use the Google/IBM cluster. That in itself will address "how can old programs use these clusters". Additionally it will provide insight into how efficiently "old programs can, or cannot" utilize these cluster. Answers learned will be readily generalizable to a wide range of application classes/types. The usage of active storage [23] will have been shown to be a valid paradigm for a new range of scientific applications, which so far have been run in a more conventional, compute-centered environment.

2 Background Work

2.1 SAGA: Basic Concepts

The Simple API for Grid Applications (SAGA) [1, 9] is a high-level programming interface being developed as an approach to solve the fundamental challenge in distributed computing: reducing the barrier for the development of truly distributed applications. To achieve this, it is critical to provide the right abstractions at the applications level, to enable applications to be developed independent of the specifics of the underlying deployment infrastructure (such as middleware distributions, cloud environments, etc.,) and applications, once developed, must remain portable and immune to the evolution and dynamics of their environments. Additionally, the programming interface should cover a broad range of different programming paradigms and usage scenarios and therefore, it should not be restrictive.

We chose SAGA as a vehicle for our development in this work for three reasons. 1) We are familiar with SAGA, as we have been driving its specification and implementation [9, 24, 25]. 2) SAGA is gaining attention in the distributed community, and with the availability of C++, Python and Java implementations [9, 26], is suited to target a wide user community. 3) Most importantly, SAGA has a number of design features that make it an exceptionally well fitting choice for the work proposed in this project:

- Data representations in SAGA are abstracted in a data format and data model agnostic manner. Data models and formats can, however, be added for specific application domains, without breaking the vertical stack.
- At the same time, SAGA also has the means to abstract *algorithmic* elements of applications. We intent to leverage this, as it is, in our opinion, a crucial element in data locality aware distributed computing.
- SAGA is *designed* to accommodate additional programming models and to support new programming

[1] winner of ISC2008 Outstanding Paper Award, of which Jha and Katz are co-authors

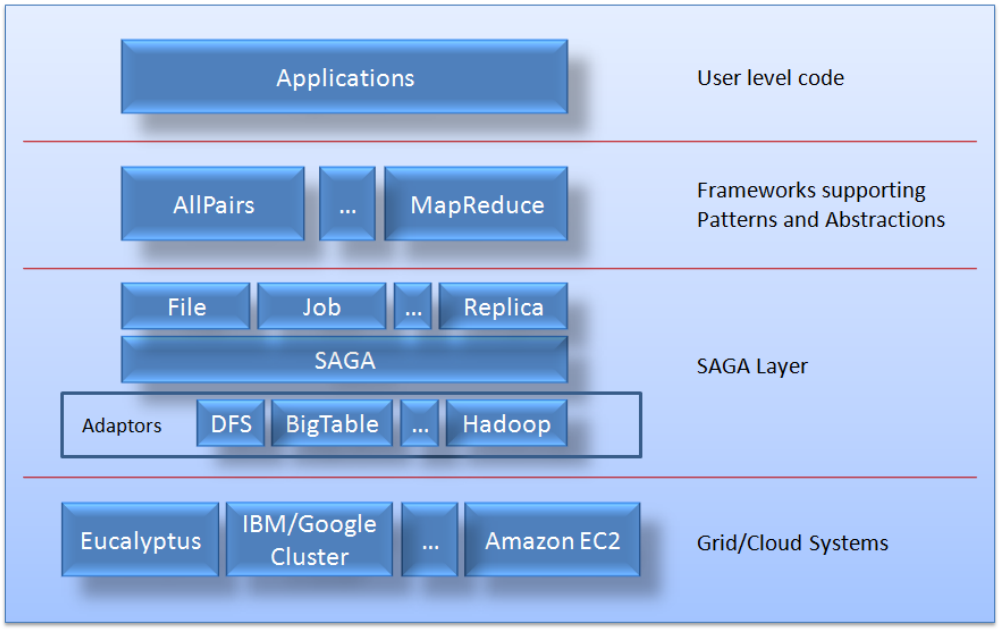

Figure 1: Layered ordering of the various components to be developed and/or used in this proposal. At the very top level, applications such as mpiBLAST, will be re-architected to utilize active storage systems. Some applications will be built using frameworks that support commonly occurring patterns. The frameworks in turn will use SAGA which will be extended to provide abstractions for data-intensive scientific applications. SAGA also provides homogeneous interface to underlying runtime systems as Grids or Clouds. Through the development of appropriate adaptors for SAGA – such as BigTable and Hadoop, applications can utilize different systems, thus enabling empirical testing of a variety of different programming models over a range of environments.

patterns. It is extensible in multiple dimensions, again without breaking the vertical stack, and with some guarantees for backward compatibility.

- SAGA has a strong focus on compute-centric distributed applications. Extending SAGA with data-centric distributed programming models will allow SAGA to serve a wider array of application classes.

Our recent work has shown that SAGA is not only a programming interface on the brink of being a OGF technical recommendation [1] that conforms to the above requirements, but that it also provides sufficient abstractions allowing the creation of frameworks for different application models, such as MapReduce, All-Pairs, various producer/consumer patterns, and truly adaptive applications (in terms of their resource and data usage patterns). These models all use SAGA at different levels [12, 13, 27, 10].

SAGA had been designed with the fundamental aim of enabling compute-intensive applications to utilize distributed environments, by providing a high-level, semantically consistent programming abstraction and a uniform interface to distinct flavors and versions of distributed runtime environments. The abstractions addressed by SAGA are currently file and replica management, job submission and control, remote procedure calls, and streaming, are mainly oriented towards compute-intensive applications, and being added are service discovery, information management, checkpoint management and application recovery.

Effective application development doesn't just require simple interfaces to allow uniform access to the different functionality provided by the deployment and runtime environments, as provided by SAGA. It also requires support for higher level application models and patterns. SAGA addresses these challenges by providing a programming interface that integrates common distributed programming abstractions while respecting critical application level requirements: simplicity, stability, portability, uniformity, and support for higher level programming models and abstractions. Figure 1 shows the overall application architecture of different classes of applications built on top of SAGA-enabled, distributed programming abstractions.

We propose to develop more extensions to SAGA to support programming models and patterns for applications that focus on data-intense program flow, such as support for distributed file systems, data scheduling

and placement, data streaming, exploitation of active storage, and data analysis pipelines. Toward that, we will validate and extend SAGA so that it can serve as the basis for the implementation of different application specific data access models and patterns. We will implement SAGA adaptors to enable access to different underlying runtime environments, such as Google's BigTable, Hadoop, etc.

Our current implementation of the SAGA API is based on a very modular and dynamic architecture. Since this implementation must cope with a multitude of different dynamic requirements, the main goal has been to maximize the decoupling of different components of the developed library to provide as much extensibility, adaptability, and modularity as possible. This implementation is built based on a layered architecture: a thin API layer is bound by the application and exposes the SAGA API as specified by OGF. This API layer dispatches the API calls to dynamically selected and loaded components, called adaptors, connecting the application to specific services and thus permitting dynamical binding to different runtime environments. Consequently, distributed applications using SAGA can be transparently executed on environments for which the respective set of adaptors exists. The complex task of updating runtime bindings is moved from the application level to the provider level.

These architectural decisions are the fundamental preconditions to be able to extend the existing framework for data-centric distributed applications, which will permit us to reuse most of the existing infrastructure.

Although SAGA remains a vibrant research project, it has matured sufficiently to be used on production-grade cyberinfrastructure such as the TeraGrid. A recent paper [11] describing how SAGA was used to develop a *first principles* distributed application and deploy it to utilize several distinct resources concurrently was awarded a Performance Challenge Award (TeraGrid 2008 Conference). The award is testimony to the fact that SAGA provides the correct level of abstractions for scientific applications and to the maturity of the SAGA software environment to provide these abstraction for computer science research into access patterns as well as production science computation.

2.2 SAGA: Recent Developments

As mentioned previously, the SAGA interface was designed with compute-centric scientific applications in mind: those where data is not a first-class citizen of the computational life-cycle, but is dependent on the compute requirements. While we will outline our plans to address this original design limitation in Section 4, here we outline how we have nonetheless been working towards developing flexible SAGA-based frameworks that can, in turn, support a range of data-intensive applications. Part of this work has been done with support from the Google Summer of Code program – which has supported the advanced undergraduate students.

A simplified taxonomy of distributed applications that *have already been* developed using SAGA leads to three classes:

- Legacy applications where local function calls are swapped for distributed function calls – for example, replica exchange.
- Applications based upon frameworks that are developed using SAGA. There are multiple examples of successful applications that use this approach, such as MapReduce, All-Pairs, and Migrating applications. Typically, the computational logic here is well separated from the distributed logic; the application is unaware of the distribution environment, while the framework contains the distributed logic.
- First principles distributed applications (such as adaptive applications [11]) where scientific applications are explicitly cognizant of the fact that they will operate in a distributed environment and the computational logic is intertwined with the distributed logic.

Efforts to implement the above application classes using SAGA have been strongly motivated by our research into distributed programming abstractions [7].

A MapReduce framework Using SAGA: MapReduce has emerged as a very successful and popular data-parallel programming model. In the Google scheme, it is implemented using files and uses their Global File Systems. However, the question arises of whether MapReduce is a viable programming model independent of the underlying infrastructure? What are the limitations of the data-parallel programming model? To address some of these questions, as well as to validate the abstractions that the SAGA interface supports, we have

recently implemented a complete stand-alone MapReduce framework that does not depend on any specific infrastructure! We have written adaptors to Hadoop, and will have adaptors to BigTable, thus providing the ability to use SAGA-based MapReduce framework in native infrastructure as well. We are now developing (re-architecting) applications that have not traditionally been written using MapReduce, and are understanding their performance and limitations. Specifically, we are building sequence alignment and search applications using the MapReduce framework. This is an illustrative example of the kind of empirical testing that is made possible using SAGA: one can build infrastructure independent frameworks upon which applications can be developed.

All-Pairs Framework for SAGA: All-Pairs is an abstraction for solving problems that have a combinatoric element; i.e., that require all elements of a set A to interact with all elements of set B. These characteristics hold for a surprisingly large set of biological, chemical, and mathematical problems. The challenge of applying the All-Pairs model is to efficiently co-locate data and computation. This problem space is not easily mappable to MapReduce, but its implementation actually shows characteristics that are similar in terms of compute/data co-location requirements, and in terms of scalability. We implemented the All-Pairs in SAGA, and use it to perform multiple alignment [12].

Virtual Global File System: A straightforward way to make distributed functionality accessible to, e.g., legacy applications, is to replace local system calls by their distributed implementations. We have been using this approach to write a file system driver on top of SAGA using the FUSE library [27]. This provides seamless access to remote filesystems, replica systems, and information services for any application using standard file related system calls. This example demonstrates how SAGA as an abstraction supports higher level application interfaces, allowing one to add distributed functionality to legacy applications without having to touch the code base.

Global File System and BigTable: Also in recent work, we started to interface SAGA to the Google Global Filesystem, and to BigTable on top of the GFS, in order to leverage the scalability and reliability features of GFS and BigTable to higher level programming models, and to SAGA applications. Our implementation is currently not really using the active data features of GFS; to do so is part of the project proposed here.

2.3 Research Theme on Distributed Programming Abstractions

Jha and Katz have been leading a two year long UK e-Science Institute theme on Distributed Programming Abstractions [7, 17]. (Jha also serves as the PI of the theme.) In its first year, the theme has systematically analyzed a range of real distributed scientific applications, defined a set of "vectors" used to characterize these applications (into essentially non-overlapping classes), identified a series of patterns – programmatic, deployment and usage modes – associated with these applications and begun the process of identifying suitable abstractions to support these commonly occurring patterns. In the near future, the theme leaders will be providing a gap analysis between existing tools and programming methods and the patterns identified. We are also finalizing negotiations with Wiley to publish a book on Abstractions for Distributed Systems.

2.4 Cloud Computing

In a recent publication [18, 28], Jha (and co-authors) have analyzed the abstractions of the typical interfaces that Cloud Systems present to the end user, and their semantic relationship to the typical interfaces that Grids have. They found that Clouds are like narrow-Grids (i.e., discipline-specific Grids such as data-grids) with *explicit support* for a well-defined set of usage modes. This leads to a simple user access interface, even though the internals (architectural, deployment or management) of Clouds systems do not have to be any less complicated than general-purpose grid. This publication also defines a Cloud's *affinity* to be the relationship between (real or virtual) resources, which determines the leading characteristics of that Cloud's usage mode.

The paper posits that general-purpose Grids are difficult to use because the semantics of the interfaces are not well defined, which in turn is because they are not designed with the aim to support explicit usage modes. In contrast, Cloud interfaces should be (and so far for the most part are) semantically simpler and well defined, for they are designed to support explicit usage modes (e.g. tasking farming, simple parameter sweeps). In a period when Cloud computing remains at best a business model if not just a weakly-defined concept, this paper has been credited with changing the focus to interface semantics and usage patterns as

a way of defining clouds, or at least providing Clouds with an operative and working distinction from Grids.

Jha and Katz are organizing a workshop at the IEEE e-Science 2008 conference [29] to be held in conjunction with Microsoft Research on Abstractions for Distributed Systems with a focus on abstractions for Cloud systems – both programmatic abstractions as well as abstractions to support commonly occurring usage modes and affinities such as data-data, compute-data, compute-compute.

2.5 Other Related Work

There is a large amount of research on distributed computing in general, and specifically on Grid systems, Clouds, data centric computing, etc., much of which is relevant to the proposed work. Here we discuss some examples of the most relevant recent work. Piernas and Nieplocha [23] provide a good overview of active data and active storage concepts. Bitdew [14] implements several mechanisms to handle active data, i.e., for computation to be scheduled based upon where data resides, specifically placement, distribution and replication. GridBatch [30] provides a batch-like cloud scheduler with active data abilities.

Commercial providers and industry, led by Google, Amazon, Yahoo, IBM, and others, have spearheaded the design, development, deployment, and usage of Clouds. As most of these companies have not disclosed the specific details of their implementations, the academic community is left to guess and to rely on popular press accounts.[2] It is clear that the focus of the commercial work has been on hosting and virtualization, not on programming models. Most work in the usage of Clouds has not been on scientific applications, which have different characteristics than typical commercial applications.

3 Applications

We will apply our research and development results to three applications: mpiBLAST [31], LIGO data analysis [3], and Montage [32]. These applications operate on very large data, represent large and disparate user communities (genome sequencing, gravitational-wave analysis, astronomical imaging). These applications are also very diverse; they represent rather different application architectures, from simple but massively parallel mpiBLAST, to data transformation and analysis pipeline for LIGO, to data driven and DAG defined workflow in Montage. The increasing architectural complexity allows us, to some extent, to incrementally implement and evaluate the programming models which are the central topic of this project: models that will support the simple (as in architecturally simple) mpiBLAST will form a subset of the models for the more complex LIGO analysis pipeline, which in turn will form a subset of the full blown DAG-based Montage workflow.

3.1 mpiBLAST

Basic Local Alignment Search Tool (BLAST) [2] used for sequence alignment is one of the most commonly used bioinformatics applications. Thanks to rapid increases in sequencing technology, it is a well established fact that the number of target sequences available – either as genomes or protein fragments, is increasing exponentially. The implications for global optimal sequence alignments searches is thus very significant, i.e., the number of comparisons required is increasing exponentially. It is unlikely that alignment algorithms in of themselves will be more efficient that those in use currently; thus without an effective implementation (programming model) and deployment of these algorithms the exponential increase in the number of targets will dominate (i.e., an effective slow-down!). Thus we propose that even though BLAST is an extensively studied and has many optimized variants, it is an important application candidate for study on an Active Data cluster such as the Google/IBM cluster.

In a classical Master-Worker (MW) application, tasks are created by the master and scheduled to the workers. Once a task is scheduled, the worker has to download the data needed before the task is executed. In contrast, a data-driven or Active-data approach requires that data are first scheduled to hosts. The programmer does not have to code explicitly the data movement from host-to-host, neither to manage fault tolerance. Programming the master or the worker consists of operating on data and attributes and reacting on data copy. This simple, but illustrative example outlines how a simple reformulation of the implementation of

[2] The publicly available information about Google's software, including Google File System, BigTable, All-Pairs, and Mapreduce, is a notable exception.

BLAST – from a classical MW to a data-driven MW formulation may yield performance dividends on the Google/IBM cluster (obviously assuming that the programming model will somehow be supported).

ParaMEDIC [22] is a framework for decoupling computation and I/O in applications that have large quantities of both; it has been specifically used for mpiBLAST [31]. The framework differs from traditional distributed I/O in that it uses application-semantic information to process the data generated by treating it as a collection of high-level application-specific objects rather than as a generic byte-stream. It uses such information to transform the data into orders-of-magnitude smaller metadata before transporting it over the distributed environment and regenerating it at the target site.

The basic operation is (1) distribute the data, (2) send query to all parts of the data, and (3) combine partial results. As such, ParaMEDIC/mpiBLAST seems ideally suited for an active storage system. However, the data-model conscious operation will give us the opportunity to prove the overall architecture of our implementation, with respect to the stated objectives of WP-1 (see Section 4).

3.2 LIGO

Gravitational waves are ripples in space-time, and their existence is a straightforward prediction of Einstein's theory of General Relativity. However, their predicted amplitude is small enough that only effects from astrophysical systems are expected to be detected: the collision of two neutron stars in the closest galaxy cluster would generate stretching of distances of 1 part in 10^{21} in the last few minutes before their coalescing into a black hole. There is very hard evidence for the existence of gravitational waves from the evolution of binary systems, but direct measurement has eluded the experimental effort in the US and worldwide, since sensitivities to such small strains have been beyond the technology up to the present. The Laser Interferometer Gravitational-wave Observatory (LIGO) [3], NSF's largest project, is designed to make the first direct detection of gravitational waves, but more importantly, to open a new window to the universe by regularly detecting gravitational waves from compact systems such as black holes and neutron stars. [33]

The first phase of the LIGO detectors achieved the designed sensitivity in late 2005, and two years of data was taken by the LIGO Scientific Collaboration (LSC), which has several hundred members. The data is being analyzed to search for strong transients of unknown origin, signals from coalescing binary systems, continuous sinusoidal signals from rotating stars, and continuous random signals from a cosmological background.

There has been no gravitational-wave signal detected yet, but the sensitivity of the detectors, while an improvement of orders of magnitude over previous efforts, had a low likelihood for detecting signals from known sources (although the current data is still being analyzed and could yield surprises). However, the technology worked as designed, and there are now better technologies that will be used for achieving a sensitivity ten times better, with a high likelihood of success in achieving the designed sensitivity. This "Advanced LIGO" has predicted observation rates of tens of coalescences per year [34]. Advanced LIGO received its initial funding this year, and is expected to begin taking data in about five years.

The analysis of LIGO data is very challenging and data intensive, and it is expected that the science goals in data-taking runs will need original ways to use computing resources, if the observations are to be fully exploited. Presently, the LIGO instruments (two detectors in Washington, and one detector in Louisiana, closely associated with LSU) acquire 1 TB of data per day; this rate will likely be even larger for Advanced LIGO. Louisiana State University has a large group dedicated to LIGO hardware and data analysis: the Co-PI Gabriela González is very involved in issues related to data analysis and detector characterization

LIGO envisions two major types of data analysis: event-driven online analysis and bulk-wave pattern matching for continuous time sources. In the first case, new data are fed into an analysis pipeline as they are acquired, to try to detect interesting patterns in near real-time. A search for events like binary coalescing systems, often considered a "flagship" search for LIGO, takes between 50 and 6000 cpu-days per day of data, depending on the signal templates used. If such signals are detected, other instruments can be tuned to the suspected source in order to confirm detection of an event.

However, the scientific potential is much larger than these searches, as well as more costly: a search for signals from rotating stars on the whole sky takes about 10^9 cpu-days per day of data. These searches do not need low latency, and are suitable for off-line strategies, where huge amounts of data (possibly taken over a long

time) are sifted for interesting patterns in a non-real-time analysis pipeline. Also, the analysis can involve parameter sweeps at several stages of that analysis pipeline. Also, the analysis can involve parameter sweeps at several stages of that analysis pipeline. An example of the results of this analysis on a month's worth of data can be found was completed by the LSC [35].

The search for these sources will need new computing paradigms, involving distributed computing. Current efforts involve computer time contributed by the general public [3], as well as analysis on a dedicated grid [4]. Using the Google/IBM cluster and generalizations thereof for the analysis of LIGO data – which is arguably unique both in volume/amounts and qualitative challenges associated with the analysis – will allow the exploitation of LIGO's science potential for the most computationally expensive searches. The LSC has explored some ways for analyzing data on the grid, with collaborators Brady and Koranda being some of the experts leading the effort, especially related to data management. The group at UWM also is heavily involved in LIGO data analysis.

Creating appropriate interfaces so that a user in the LSC can search a particular patch on the sky without being limited by computing infrastructure will go a long way towards making gravitational-wave astronomy begin to resemble modern astronomy, based on electromagnetic data taken by expensive instruments, analyzed by users with different science goals. Given the amount of data to be processed and the exciting science to be discovered, LIGO is a natural application for exploiting the results of the proposed research into extending the SAGA interface and its supports for underlying programming models.

3.3 Montage

Many science data processing applications can be expressed as a sequence of tasks to be performed. One such astronomy application builds science-grade mosaics from multiple images as if they were single images with a common coordinate system, projection, etc. This software must preserve the astrometric and photometric integrity of the original data, and rectify background emission from the sky or from the instrument using physically-based models. The Montage project [4, 32, 36, 37, 38] delivers such tools to the astronomy community. Montage has been used by the following projects: Spitzer Space Telescope Science Legacy Projects SWIRE, GLIMSE, and SAGE; Spitzer's Outreach Office; 2MASS; IRSA; the Hubble Treasury program; iPHAS; COSMOS; Astrogrid; and IPAC Cool Cosmos.

Montage has been designed as a scalable, portable toolkit that can be used by astronomers on their desktops for science analysis, integrated into project and mission pipelines, or run on computing grids to support large-scale product generation, mission planning and quality assurance. It is part of the National Virtual Observatory, and uses NVO-standard services to locate and access data.

There are often five steps to building a mosaic with Montage, though other processing is also possible:

- Select and acquire input images
- Re-projection of input images to a common spatial scale, coordinate system, and projection
- Modeling of background radiation in images to achieve common flux scales and background levels by minimizing the inter-image differences
- Rectification of images to a common flux scale and background level
- Co-addition of re-projected, background-corrected images into a final mosaic

Montage accomplishes these tasks in independent, portable, ANSI C modules. This approach controls testing and maintenance costs, and provides flexibility to users. They can, for example, use Montage simply to re-project sets of images and co-register them on the sky, implement a custom background removal algorithm, or define another processing flow through custom scripts. Montage input data files are between 512 pixels and 6,000 pixels on edge, and output mosaics can be O(100,000) pixels on edge. Typical mosaics can involve tens to tens of thousands of input files per band, and are usually one to three bands. Montage processing can be done by running a script that runs the Montage modules in sequence, or on a parallel computer, the execution of the instances of a single module can be run in parallel, or on a grid, a multi-level workflow can

[3] http://www.einsteinathome.org/
[4] http://www.lsc-group.phys.uwm.edu/lscdatagrid/

be defined and mapped to resources, and then executed using tools such as Pegasus [39, 40, 41, 42], and DAGMan [43]. Montage includes a module that can be used to build a DAG for this latter case.

The generality of Montage as a workflow application has led it to become an exemplar for those in the computer science community who study workflows and workflow-based applications, such as those working on: Pegasus, ASKALON [44], QoS-enabled GridFTP [45], SWIFT [46], SCALEA-G [47], VGRaDS [48], etc.

4 Proposed Research and Development Plan

Our plans for this project cover both research and development. On the research side, we plan to achieve a better understanding of the data related properties of a variety of classical scientific applications, which will give us insight into appropriate programming models, some of which are available, and others of which are not. On the development side, we will provide these programming models to a small but high profile selection of applications, which all have distinct characteristics, and are so far known not to map easily to the well known data centric distributed (DCD) programming paradigms, such as MapReduce, Hadoop, All-Pairs, etc.

Note that our work will not require any special access to the cluster; it all can be done by a typical user.

The proposed work is thus structured into the following work packages:

- WP-1: Research of DCD related programming models
  - Task-1.1: Survey of existing DCD programming models and applications
- WP-2: Implementation of DCD programming models, with SAGA
  - Task-2.1: Data format and model abstractions in SAGA
  - Task-2.2: Algorithmic encapsulation related programming models
  - Task-2.3: Data Exchange and communication related programming models
- WP-3: Development of DCD applications
  - Task-3.1: mpiBLAST
  - Task-3.2: LIGO data analysis
  - Task-3.3: Montage
- WP-4: Deployment and portability related work
  - Task-4.1: Application deployment and scalability tests with the Google/IBM cluster
  - Task-4.2: Experiments with Eucalyptus, GFS, BigTable, Amazon EC2/S3

WP-1: Research of DCD related programming models

Task-1.1 Survey of existing DCD programming models and applications

Jha, Katz, and participating researchers have been working on surveying *compute-centric* distributed (CCD) programming models in the past [17]. In the course of this project, we intent to apply a similar approach to *data-centric* distributed (DCD) programming models, and possibly to applications with aspects of both. That survey (currently in a draft state) will pinpoint the dimensions, attributes required to describe the respective programming models, to allow for a systematic and ordered approach to the problem space.

We do not require that survey be complete, nor that it span the whole space of *possible* models, but we expect the survey to be (a) a useful guideline to application developers of DCD applications, and (b) to be a useful tool in a coverage and gap analysis for DCD programming models.

Deliverables:

- D-1.1.1: Survey of existing DCD applications
- D-1.1.2: Survey of existing DCD programming models
- D-1.1.2: Coverage and gap analysis of DCD programming models

WP-2: Implementation of DCD programming models, with SAGA

The identified programming models from WP-1 are to be made available to application developers. At the very minimum, those models required by the three application domains in the project (see Sections 3.1, 3.2, and 3.3) are to be implemented, within SAGA (as motivated in Section 2.1). This work package thus depends on WP-1, and feeds into WP-3.

Task-2.1: Data format and model abstractions in SAGA

One of the seemingly biggest inhibitors for the use of SAGA in data intensive science is SAGA's ignorance of data syntax and semantics. Earlier work on that topic [49, 50, 51, 52, 53] conveys the impression that it is impossible to implement efficient data intensive applications without prior knowledge, and appropriate optimization, of the involved data models and formats.

But, the lesson to be learned from Google's and other's recent work on algorithms like MapReduce, frameworks like Hadoop, and technologies like the GFS, is that it is indeed possible to provide an almost complete vertical software stack in a mostly data-model and data-format agnostic manner, without any loss in performance: applications running in these environments keep all data-model and data-format related assumptions solely at the application level.

Interestingly, the SAGA approach to data model and format agnosticity is very similar: the `saga::buffer` class is designed to encapsulate data transparently, and to leave syntactic and semantic interpretation of these data to the application level.

We believe that a combination of the programmatic data abstractions available in SAGA, and of high level programmatic data algorithms such as MapReduce, provides powerful and elegant tools for the application developers with Grid background, i.e. the 'standard' SAGA target users.

- D-2.1.1: Design of a complete vertical model and format agnostic data management
- D-2.1.2: Map of the above design to well known DCD patterns, such as MapReduce

Task-2.2 Algorithmic encapsulation related programming models

Typically, MapReduce- and Hadoop-like algorithms allow application developers to insert custom hooks into their runtime stack. These hooks basically form the kernel of the algorithm. The hooks are invoked once or more times per unit of data. This way, the complete application is really data-driven, not compute-driven.

SAGA, on the other hand, provides means of *event*-driven computation, by allowing applications to implement callbacks for specific events. This approach, however, is limited, as it neither allows for data- or compute-related events, nor easily provides means for the application to specify new events.

We think that, for example by exploiting QT [54]-like signal/slot mechanisms, and by utilizing generic SAGA data buffers (see above) as data exchange mechanism, we can elegantly combine data- and event-driven paradigms and also provide the means to specify algorithmic hooks (such as required by MapReduce).

- D-2.2.1: Prove the viability of SAGAs algorithmic abstraction layers for the project's applications.
- D-2.2.2: If required, provide additional means for algorithmic abstractions, as described.

Task-2.3 Data Exchange and communication related programming models

Similar problems as those discussed above have hindered the creation of data format and data model independent communication and data exchange solutions. On the one end, XML- and RDF-based data model languages are considered flexible and powerful but too slow and complex to use. On the other end, binary data encapsulation and storage formats (i.e., RPC, CORBA, DCOM, HDF5) have existed many years, but are considered of limited portability or flexibility.

Recent developments such as the Hadoop File System (based on the Google file system) and other distributed, replicated, block-based file systems have shown that shared concurrent data access to structured data is not prohibitively expensive. Similarly, Google Protocol Buffers [55] shows this for structured message exchange.

Both paradigms match well with SAGA's abstractive approach to data exchange, and should, combined with support for explicit and policy-based data co-location, provide a good foundation for DCD-style applications.

- D-2.3.1: Provide SAGA binding to Google Protocol Buffers, and prove ability to vertically integrate data modal and format aware technology into SAGA,
- D-2.3.2: Extend SAGA for explicit and policy based data/compute co-location abilities
- D-2.3.2: Provide data synchronization, communication, storage, and persistency as needed in WP-3.

WP-3: Development of data centric distributed applications

The results of WP-2 are a set of programming models, and possibly frameworks, that utilize active storage systems for *novel types* of applications. WP-3 will apply these models to three application use cases: mpiBLAST, LIGO data analysis and Montage. The complexity of the respective distribution pattern increases over the three tasks: mpiBLAST has a relatively simple, 1-dimensional distribution pattern; LIGO data analysis forms a more complex pipeline with integrated parameter sweeps; and Montage will require support for full-blown DAGs. The tasks will thus build upon the results of each other.

Task-3.1: mpiBLAST

As described in Section 3.1, mpiBLAST, and ParaMEDIC, seem ideally suited for an active storage system. The data-model conscious operations (distribute the data, send query to all parts of the data, and combine partial results) will give us the opportunity to prove the overall architecture of our implementation, in respect to the stated objectives of WP-1.

The programming models of mpiBLAST applications seem mostly to focus on data centric, massive parallel computation. That simple pattern will allow us to start to work on applications early on in the project, and will also provide a useful foundation for the more complex patterns required by the other applications.

Deliverables:

- D-3.1.1: Exploitation of active storage for mpiBLAST applications
- D-3.1.2: Data model conscious implementation of mpiBLAST applications

Task-3.2: LIGO data analysis

LIGO bulk wave pattern matching, as described in Section 3.2, will be the subject of this task. Here, huge amounts of data are sifted for interesting patterns in a non-real-time fashion, where the patterns of interest are defined by target waveforms that are obtained by (computationally very expensive and difficult) simulations of relativistic astrophysical events [56, 57]. This data analysis is not as simple as that in particle physics. The LIGO data needs to undergo a relatively complex, non-linear set of transformations before the actual pattern matching of waveforms can be performed. Also, the analysis can involve parameter sweeps at several stages of that analysis pipeline.

Thus, this work package will need to handle two types of challenges, in respect to the topics targeted in this proposal: (a) it needs to allow for pipeline type algorithms on large data sets, and (b) needs to be able to accommodate parameter sweep like fan-out of these algorithms.[5]

Deliverables:

- D-3.2.1: Exploitation of active storage for pipeline type data flows
- D-3.2.2: Exploitation of active storage for parameter sweeps (fan-out)
- D-3.2.3: Mapping of LIGO data analysis pipelines to the above techniques

Task-3.3: Montage

Montage, as described in Section 3.3, is typical of many successful Grid applications: it comes with a wide variety of use cases, works on a large variety of data sets of data, and is loosely coupled, but compute intensive. Its workflow-like nature makes it comparatively difficult to execute on a active data storage system: while the data locality of the input data can be pre-determined, the locality of intermediate data sets can only be determined through the DAG that represents the specific Montage instance. This characteristic makes it hard to optimize Montage for more massive parallel/loosely coupled frameworks like MapReduce or Hadoop. Typical applications that use these frameworks can be thought of as having a one- or two-level DAG, and are thus significantly easier to manage than a full-blown DAG-based workflow (or data flow) like Montage.

[5] Note that these items are preparing the more complex approach to complete DAGs on active storage systems, which is the topic of the next task.

This work package will particularly use the results from Task-2.2 (algorithmic encapsulation) and Task-2.3 (data exchange and communication) to exploit active storage for DAG based systems.

Deliverables:

- D-3.3.1: Programmatic means in SAGA to analyze DAGs, and to determine intermediate data location
- D-3.3.2: Utilization of active storage for scheduling of DAG components
- D-3.3.3: Mapping of Montage instances to active storage resources

WP-4: Deployment and portability related work

Due to the Google/IBM cluster's integrated active-data capabilities, access to the cluster, provides an excellent opportunity to validate our data-centric distributed programming models. We will use the cluster to (a) test and deploy SAGA, for the development work in WP-2, (b) install and deploy the applications from WP-3, to support the WP-3 development work, and (c) perform stress tests, and scalability and performance analysis of the resulting applications.

We intend, however, to go beyond the Google/IBM cluster, and in a second step, to prove the viability of our work on other experimental facilities such as Eucalyptus and on commercial production Cloud environments such as Amazon's EC2/S3. The data management abilities of these environments are less sophisticated, or at least have different strengths and abilities, than the Google/IBM cluster. This will allow us to (a) perform a reality check of our approaches, (b) determine performance and scalability dependencies from the underlying infrastructure, and (c) investigate barriers to application-level interoperability. In particular, for large scale collaborations such as Montage or LIGO, this information is important in determining potential uptake of the results of this project.

We can then deploy and experiment with additional and alternate programming models of active data – using data schedulers such as Stork [58] and Bitdew [14]. While it is unclear what role such data-schedulers play in systems that have active storage, it is clear that for any production science distributed cluster, there will be a need to have a local cluster with heterogenous extensions to the larger cluster; the extensions arising in the context of active data policies and management. Our approach provides this capability.

Task-4.1: Application deployment and scalability tests with the Google/IBM cluster

Perfect scalability is a goal in distributed computing that is rarely achieved, but in many ways, it defines the success metric for large scale distributed applications. Scalability in data-centric distributed applications means that the amount of data an application is able to handle is proportional to the number of resources (data and compute) involved. Our experiments will strive to confirm that the implementation overhead for our DCD programming models does not negatively impact either the performance or the scalability of the respective applications, and to find remedial measures where they might.

Deliverables:

- D-4.1.1 Deployment of SAGA on Google/IBM cluster, including simple performance and scalability tests
- D-4.1.2 Deployment, performance/scalability tests of mpiBLAST
- D-4.1.3 Deployment, performance/scalability tests of LIGO
- D-4.1.4 Deployment, performance/scalability tests of Montage

Task-4.2: Experiments with Eucalyptus, GFS, BigTable, Amazon EC2/S3

SAGA allows us to decouple the application programming model from the data and compute platform. Task 4.2 will show this by performing experiments outside CluE's Google/IBM cluster, on a variety of platforms, while measuring performance and scalability on each platform. This will enable us to compare system and environment contribution and factors related to performance.

Deliverables:

- D-4.2.1 SAGA and application deployment on Eucalyptus, performance and scalability tests

- D-4.2.2 SAGA and application deployment on GFS/BigTable, performance and scalability tests
- D-4.2.3 SAGA and application deployment on EC2/S3, performance and scalability tests

5 Intellectual Merit and Broader Impact

Intellectual Merit: It is clear that the "data deluge" or explosion of data is making processing that data an increasingly important part of computing. It also appears that Clouds and active storage clusters may be good platforms for such computing. The proposed work has the potential to advance knowledge across all areas of science, technology, business, etc., where the processing of large amounts of data is currently a bottleneck. If the work in providing abstractions and mechanisms for such general problems is successful, it could transform the ease with which large amounts of computing is done, potentially opening whole new areas to new communities who are currently unwilling to try to work with very large data sets. The team proposing this work is highly qualified to carry it out, based on experience with distributed and parallel programming in general, abstractions for distributed programming, and experience in data processing in two separate fields of science. The plan of research is clearly well organized, and sufficient resources (computing, software, abstractions, tools, application knowledge, and data) are available to provide a very good likelihood of success.

Broader Impacts: The majority of the work in this proposal will be done at LSU, in an EPSCoR state. We are also working on an statewide NSF EPSCoR project generally referred to as Cybertools [59], which is aimed at advancing both tools/services and applications at the same time by using each to drive developments in the other. This work will fit directly into the tools/services part of Cybertools. Cybertools also has a set of education activities to which this work will be added. These include summer boot camps for high school students, including predominately minority schools, and summer research projects for undergraduates, where we successfully strive to find students from underrepresented groups. Additionally, Jha and Katz (with LSU CS Prof. Gabrielle Allen) are currently planning a graduate course in Abstractions for Distributed Systems, where lessons from this project will be taught.

LSU operates the LONI network and distributed high performance computing (HPC) resources for the state of Louisiana. In this role, we provide statewide training to potential HPC and distributed computing users. We will include the results of this work in our training workshops. LSU is also a driving member of SURAgrid, so this work will be pushed out across the SURA region, which has been shown to have a relatively low level of HPC and distributed computing knowledge and usage compared with the rest of the United States. LONI (as led by LSU) is also a Resource Provider (RP) partner in the TeraGrid, which give us the opportunity to push the results of this work to other national computing centers and their national audience.

6 Management, Staffing, and Outcomes

Project Management: Overall management of this work will be done by PI Jha who will liaise with NSF and ensure that the entire LSU team and collaborators at UCSB and UWM work closely together by coordinating regular conference calls and exchanges. Jha will also supervise the research programmer and work on application analysis and programming abstractions. Co-PI Katz will provide Montage experience and contribute to programming abstractions aspects. Co-PI Gonzalez and collaborators Brady and Koranda will provide LIGO experience as well as general user feedback. SI Kaiser, who has 15 years software and industrial experience, will supervise the software engineering aspects of the project.

A research programmer will be primarily responsible for most of the work related to analyzing applications, extending SAGA, implementing abstractions, as well as providing frameworks for the common patterns. PI Jha and co-PI Katz will work closely with the Research Programmer on most of these areas. One graduate student for two years will be dedicated to the effort of implementing the abstractions and programming models for LIGO's data analysis requirements. This student will be jointly supervised by Gonzalez and Katz. A second graduate student is being requested for partially doing the same for Montage.

Resource Justification: We are asking for one Research Programmer, one graduate student for two years, and one graduate student for one year (in the second year of the project). We are asking for approximately $20,000 per year in travel money, of which $5,000 per year is provisioned (as suggested by the solicitation) for program related meetings; $5,000 per year is requested to support biannual visits for UWM (LIGO team

members) and LSU collaboration; $2,500 per year is requested to support the collaboration between the Eucalyptus and LSU teams, and approximately $7,500 is requested to attend related conferences and paper presentations. Finally, PI Jha seeks salary support for one summer for contributions to research efforts and the management of the project.

Result Dissemination: SAGA is currently available under the Boost OSS License v1.0. The updates to SAGA that are made as a result of this project will be made available under the same license. Other resulting lessons from this work will be publicized through workshops, conferences, and papers. In conjunction with the book on Abstractions for Distributed Systems, the patterns, abstractions to support these patterns including code examples will be made available via the PI and co-PI (Katz) webpages and will be used as material for (planned) Graduate course in Abstractions for Distributed Systems.

Relevant Prior Research Funded by NSF

Jha has recently moved to the US from UK and has not been a PI on an NSF grant. He is the PI of the SAGA project and research theme on Distributed Programming Abstractions, both of which are current and funded by the EPSRC – the UK's national-level science foundation. Jha is a SI on several NSF-funded projects, including the NSF-RII Cybertools project and HPCOPS. Jha also has small grants from Google ("Exposing the Power of Google using SAGA: A SAGA implementation of MapReduce") and the US NIH relevant to this proposal.

Katz is a Co-PI and project lead for LONI's participation in the TeraGrid [60]: "HPCOPS: The LONI Grid – Leveraging HPC Resources of the Louisiana Optical Network Initiative for Science and Engineering Research and Education," NSF award OCI-0710874, $2m from 10/2007 to 9/2009. In this work, the LONI system, Queen Bee, was integrated into the TeraGrid on schedule, and the national community is satisfied with the system, based on higher-than-expected usage. His work that is most closely related to this proposal has been as a member of the Montage team under NASA funding.

Gonzalez and Brady are leading members of the LIGO Scientific Collaboration (LSC), where they were co-chairs of the working group on searches of compact binary coalescences until end of 2007. This working group wrote several papers for the LSC, on searching signals in data taken in the several science runs with LIGO detectors (a complete list of papers is found in www.ligo.org, under "Observational Results"). Gonzalez is a recognized member of the experimental gravitational-wave community, where she has contributed to the successful commissioning and diagnostic of the LIGO detectors; she has received continued funding from NSF for this purpose since 1998.

7 Conclusions

The work proposed as part of "Abstractions and Programming Models for Data Intensive Science" is a balanced mix of theoretical work, empirical testing, and re-architecting and deploying three important applications that represent three distinct application classes. The three applications, while being mostly data-intensive, are not without computational challenges.

The first major product and research outcome of this project will be a general purpose tool usable to address many of the questions related to programming models and abstractions for data-intensive computing. A second major research impact will be that three important applications will have been re-architected and programmed so as to be able to use the Google/IBM cluster and other environments. We will thus address "how can old programs use these" clusters, while also providing insight into the limitations of such environments. Answers learned will be readily generalizable to a wide range of application classes and types. The usage of active storage [23] will have been shown to be a valid paradigm for a new range of scientific applications, which so far have been run in a more conventional, compute-centered environment.

Real, scientific applications and a focus on their effective usage of new systems form a central theme of our proposal. We hope that this will set the tone in these early days of Clouds and active storage clusters, so that the evolution of such systems will consider the centrality of applications – something that may be argued was not the case for traditional Grids.

References

[1] Tom Goodale, Shantenu Jha, Hartmut Kaiser, Thilo Kielmann, Pascal Kleijer, Andre Merzky, John Shalf, and Christopher Smith. A Simple API for Grid Applications (SAGA). Grid Forum Document GFD.90, 2007. Open Grid Forum.

[2] BLAST - Basic Local Alignment Search Tool. http://blast.ncbi.nlm.nih.gov/Blast.cgi.

[3] LIGO - Laser Interferometer Gravitational-Wave Observatory. http://www.ligo.caltech.edu.

[4] Montage Project,
http://montage.ipac.caltech.edu/.

[5] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters.
http://labs.google.com/papers/mapreduce.html.

[6] Christopher Moretti and Jared Bulosan and Douglas Thain and Patrick J. Flynn. All-Pairs: An Abstraction for Data-Intensive Computing in Shared Distributed Systems. http://cse.nd.edu/ dthain/papers/allpairs-tr.pdf.

[7] eScience Institute Theme on Distributed Programming Abstraction,
http://wiki.esi.ac.uk/Distributed_Programming_Abstractions.

[8] Eucalyptus - Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems.
http://eucalyptus.cs.ucsb.edu/.

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

[10] Shantenu Jha, Hartmut Kaiser, Yaakoub El Khamra, and Ole 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.

[11] Shantenu Jha, Hartmut Kaiser, Yaakoub El Khamra, Andre Merzky, and Ole Weidner. Developing Large-Scale Adaptive Scientific Applications with Hard to Predict Runtime Resource Requirements. In *Accepted for TeraGrid08 Conference, Las Vegas, June 2008*, Winner of the Performance Challenge Award.

[12] Shantenu Jha and Hartmut Kaiser and Michael Miceli and Christopher Miceli and Joao Abecasis. Implementing Abstractions for Data Intensive Applications using SAGA. In *Accepted for publication in UK e-Science and special issue Phil. Trans Royal Society A, London, 2008*, 2009.

[13] Google Summer of Code Project, http://code.google.com/soc/2008/omii/about.html,
Project title: "A Map Reduce Framework Using SAGA".

[14] Bitdew: Open Source Middleware for Data Desktop Grids, http://www.bitdew.net/.

[15] Hadoop. http://hadoop.apache.org/core/.

[16] Fay Chang and Jeffrey Dean and Sanjay Ghemawat and Wilson C. Hsieh and Deborah A. Wallach and Mike Burrows and Tushar Chandra and Andrew Fikes and Robert E. Gruber. Bigtable: A Distributed Storage System for Structured Data. http://labs.google.com/papers/bigtable.html.

[17] S. Jha et al., *Programming Abstractions for Large-scale Distributed Applications*, to be submitted to ACM Computing Surveys; draft available at
http://www.cct.lsu.edu/~sjha/publications/dpa_surveypaper.pdf.

[18] Using Clouds to Provide Grids with Higher-Levels of Abstraction and Explicit Support for Usage Modes, S Jha, A Merzky, G Fox, accepted for publication in Concurrency and Computing Practice and Experience, pre-print available at
http://www.ogf.org/OGF_Special_Issue/.

[19] The Landscape of Parallel Computing Research: A View From Berkeley. http://view.eecs.berkeley.edu/wiki/Main_Page.

[20] Adam Barker, Jon B. Weissman, and Jano van Hemert. Eliminating the middleman: peer-to-peer dataflow. In *HPDC*, pages 55–64, 2008.

[21] Infrastructure Independent and General Purpose Implementation of MapReduce using SAGA. http://www.cct.lsu.edu/~sjha/saga-mapreduce.

[22] P. Balaji and W. Feng and H. Lin and J. Archuleta and S. Matsuoka and A. Warren and J. Setubal and E. Lusk and R. Thakur and I. Foster and D. S. Katz and S. Jha and K. Shinpaugh and S. Coghlan and D. Reed. Distributed I/O with ParaMEDIC: Experiences with a Worldwide Supercomputer. http://www.supercomp.de/isc08/content/e1738/e1926/e1937/e4139/index_eng.html.

[23] Juan Piernas and Jarek Nieplocha. Efficient Management of Complex Striped Files in Active Storage. In *Europar*, 2008.

[24] Tom Goodale et al. SAGA: A Simple API for Grid Applications – High-Level Application Programming on the Grid. *Computational Methods in Science and Technology: special issue "Grid Applications: New Challenges for Computational Methods"*, 8(2), SC05, November 2005.

[25] Shantenu Jha et al. A Collection of Use Cases for a Simple API for Grid Applications. Grid Forum Document GFD.70, 2006.

[26] JSAGA. http://grid.in2p3.fr/jsaga/index.html.

[27] Filesystem in Userspace. http://fuse.sourceforge.net/.

[28] Using Clouds to Provide Grids with Higher-Levels of Abstraction and Explicit Support for Usage Modes, S Jha, A Merzky, G Fox, invited talk delivered at OGF Barcelona session on Cloud Computing.

[29] IEEE e-Science 2008 Conference http://escience2008.iu.edu/.

[30] H. Liu and D. Orban. GridBatch: Cloud Computing for Large-Scale Data-Intensive Batch Applications. *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on*, pages 295–305, 2008.

[31] A.E. Darling, L. Carey, and W. Feng. The Design, Implementation, and Evaluation of mpiBLAST. *Proceedings of ClusterWorld*, 2003, 2003.

[32] G. B. Berriman, D. Curkendall, J. C. Good, J. C. Jacob, D. S. Katz, M. Kong, R. Moore S. Monkewitz, T. A. Prince, and R. E. Williams. An architecture for access to a compute intensive image mosaic service in the NVO. In *Virtual Observatories, A. S. Szalay, ed., Proceedings of SPIE*, volume 4846, pages 91–102, 2002.

[33] B. Barish and R. Weiss. *Phys. Today*, 52(44), 1999.

[34] R. K. Kopparapu, CHanna, V. Kalogera, R. O'Shaughnessy, G. Gonzalez, P R Brady, and S Fairhurst. *The Astrophysical Journal*, 675:1459–1467, March 10 2008.

[35] B. Abbott et al. (LIGO Scientific Collaboration). *Phys. Rev. D*, 77, 2008.

[36] G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, D. S. Katz, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, and M.-H. Su. Montage: A grid enabled engine for delivering custom science-grade mosaics on demand. In *Scientific Return for Astronomy through Information Technologies, P. J. Quinn, A. Bridger, eds., Proceedings of SPIE*, volume 5493, pages 221–232, 2004.

[37] D. S. Katz, N. Anagnostou, G. B. Berriman, E. Deelman, J. Good, J. C. Jacob, C. Kesselman, A. Laity, T. A. Prince, G. Singh, M.-H. Su, and R. Williams, *Astronomical Image Mosaicking on a Grid: Initial Experiences*, in Engineering the Grid - Status and Perspective, B. Di Martino, J. Dongarra, A. Hoisie, L. Yang, and H. Zima, eds., American Scientific Publishing, January 2006.

[38] D. S. Katz, G. B. Berriman, E. Deelman, J. Good, J. C. Jacob, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, and M.-H. Su. A comparison of two methods for building astronomical image mosaics on a grid. In *Proceedings of the 7th Workshop on High Performance Scientific and Engineering Computing (HPSEC-05)*, 2005.

[39] E. Deelman, S. Koranda, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, and R. Williams. GriPhyN and LIGO, building a virtual data grid for gravitational wave scientists. In *Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing*, 2002.

[40] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, and K. Vahi. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1), 2003.

[41] E. Deelman, R. Plante, C. Kesselman, G. Singh, M.-H. Su, G. Greene, R. Hanisch, N. Gaffney, A. Volpicelli, J. Annis, V. Sekhri, T. Budavari, M. Nieto-Santisteban, W. OMullane, D. Bohlender, T. McGlynn, A. Rots, and O. Pevunova. Grid-based galaxy morphology analysis for the national virtual observatory. In *Proceedings of SC2003*, 2003.

[42] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny. Pegasus: mapping scientific workflows onto the grid. In *Across Grids Conference*, 2004.

[43] J. Frey, T. Tannenbaum, M. Livny, and S. Tuecke. Condor-G: a computation management agent for multi-institutional grids. In *Proceedings of the 10th IEEE Symposium on High-Performance Distributed Computing*, 2001.

[44] Marek Wieczorek, Radu Prodan, and Thomas Fahringer. Scheduling of scientific workflows in the askalon grid environment. *ACM SIGMOD Record*, 34(3):52–62, 2005.

[45] Marty Humphrey and Sang-Min Park. Data throttling for data-intensive workflows. In *Proceedings of 22nd IEEE International Parallel and Distributed Processing Symposium*, 2008.

[46] Borja Sotomayor, Kate Keahey, Ian Foster, and Tim Freeman. Enabling cost-effective resource leases with virtual machines. In *Proceedings of HPDC 2007*, 2007.

[47] Hong-Linh Truong, Thomas Fahringer, , and Schahram Dustda. Dynamic instrumentation, performance monitoring and analysis of grid scientific workflows. *Journal of Grid Computing*, 2005(3):1–18, 2005.

[48] VGRaDS: Montage, a project providing a portable, compute-intensive service delivering custom mosaics on demand. http://vgrads.rice.edu/research/applications/montage.

[49] FiberBundleHDF5: The Fiber Bundle HDF5 Library. http://www.fiberbundle.net/.

[50] J. Ambrosiano, DM Butler, C. Matarazzo, M. Miller, and L. Schoof. Development of a common data model for scientific simulations. *Conference: 11. international conference on scientific and statistical database management, Cleveland, OH (United States), 28-30 Jul 1999*, 1999.

[51] XML.org: Focuse Are Community. http://www.xml.org/.

[52] The HDF Group, HDF5 Homepage. http://hdf.ncsa.uiuc.edu/HDF5/.

[53] DFDL: Data Format Description Language. http://forge.gridforum.org/projects/dfdl-wg/.

[54] Qt Cross-Platform Application Framework. http://trolltech.com/products/qt/.

[55] Protocol Buffers - Google Code. http://code.google.com/apis/protocolbuffers/.

[56] Cactus Framework - An Open Source Problem Solving Environment for Scientists and Engineers. http://www.cactuscode.org.

[57] Tom Goodale et al. The Cactus Framework and Toolkit: Design and Applications, Vector and Parallel Processing — VECPAR'2002, 5th International Conference, Springer, (2003).

[58] T. Kosar and M. Livny. Stork: Making data placement a first class citizen in the grid. In *In 24th Int. Conference on Distributed Computing Systems*, 2004.

[59] The Cybertools Project. http://cybertools.loni.org.

[60] D. S. Katz et al. Introduction to the TeraGrid. to be presented at 2008 UK e-Science All Hands Meeting.