Critical Perspectives on Large-Scale Distributed Applications and Production Grids

Shantenu Jha[1,2,3], Daniel S. Katz[4,1,5],
Manish Parashar[6,7], Omer Rana[8], Jon Weissman[9]

[1]Center for Computation & Technology, Louisiana State University
[2]Department of Computer Science, Louisiana State University
[3]e-Science Institute, University of Edinburgh
[4]Computation Institute, University of Chicago
[5]Department of Electrical and Computer Engineering, Louisiana State University
[6]NSF Center for Autonomic Computing, Rutgers University
[7]Department of Electrical and Computer Engineering, Rutgers University
[8]Department of Computer Science, Cardiff University
[9]Department of Computer Science, University of Minnesota

***Abstract*—It is generally accepted that the ability to develop large-scale distributed applications that are extensible and independent of infrastructure details has lagged seriously behind other developments in cyberinfrastructure. As the sophistication and scale of distributed infrastructure increases, the complexity of successfully developing and deploying distributed applications increases both quantitatively and in qualitatively newer ways. In this paper we trace the evolution of a representative set of "state-of-the-art" distributed applications and production infrastructure; in doing so we aim to provide insight into the evolving sophistication of distributed applications – from simple generalizations of legacy static high-performance to applications composed of multiple loosely-coupled and dynamic components. The ultimate aim of this work is to highlight that even accounting for the fact that developing applications for distributed infrastructure is a difficult undertaking, there are suspiciously few novel and interesting distributed applications that utilize production Grid infrastructure. Along the way, we aim to provide an appreciation for the fact that developing distributed applications and the theory and practise of production Grid infrastructure have often not progressed in phase. Progress in the next phase and generation of distributed applications will require stronger coupling between the design and implementation of production infrastructure and the theory of distributed applications, including but not limited to explicit support for distributed application usage modes and advances that enable distributed applications to *scale-out*.**

I. Introduction: Context, Scope and Outline

Significant strategic investments coupled with tremendous technical advances have resulted in distributed computational infrastructures that integrate computers, networks, data archives, instruments, observatories and embedded sensors, with the overarching vision of enabling new paradigms and practices in computational science and engineering. Several large-scale distributed applications have also demonstrated the potential benefits of such infrastructures and have provided new scientific insights in their disciplines. However, such applications have been relatively few, and have also demonstrated the costs, in terms of time and resources, associated with developing, deploying, and managing the applications.

There are many factors responsible for the perceived and genuine lack of large-scale distributed applications that can seamlessly utilize distributed infrastructures in an extensible and scalable fashion. We believe that at the root of the problem is the fact that developing large-scale distributed applications is fundamentally a difficult process, for reasons such as the need to coordinate distributed data and computation. This is true, irrespective of whether data is transported to where the computation takes place, or if computation is decomposed and transported to the location of data, or a hybrid of the two approaches is adopted. Commonly acceptable and widely used models and abstractions remain elusive. Instead, many ad-hoc solutions are used by application developers. The range of tools, programming systems, and environments is bewilderingly large, making extensibility and interoperability difficult. Deployment and execution concerns are often disjoint from the development process. Against this backdrop, the distributed infrastructure available to scientists continues to evolve, in terms of scale and capabilities as well as complexity. Support for and investments in legacy applications need to be preserved, whilst at the same time facilitating the development of novel and architecturally different applications for new and evolving production environments. As we will demonstrate, the process of developing and deploying large-scale distributed applications presents a critical and challenging agenda for researchers and developers working at the intersection of computer science, computational & application sciences and cyberinfrastructure development.

Distributed versus Parallel Applications: To enhance an appreciation of distributed applications, and to set the stage for a discussion of a set of specific applications in Section III, we will begin with a brief definition and discussion of distributed applications. In particular, we will try to understand some of their basic characteristics as contrasted with those of their better known parallel counterparts. Distributed applications utilize multiple resources, or are capable of utilizing them. One could argue that any application that would benefit from increased peak performance, throughput, or reduced time to solution, by using multiple compute and distributed data resources, can be classified as a distributed application.

The complexity of developing applications for such large-scale problems stems in part from combining the challenges inherent in high-performance computing and large-scale dis-

978-1-4244-5149-4/09/$26.00 © 2009 IEEE

tributed systems. In traditional high-performance computing, performance is paramount, and systems are typically homogeneous, assumed to be almost failure free (at least historically), and applications operate within a single administrative domain. Performance, as measured by peak utilization, is *often* considered a secondary concern for distributed systems, which is a reflection of the fact that the concerns and objectives for distributed applications are broader, including issues of scalability, heterogeneity, fault-tolerance and security. In addition, distributed applications differ from monolithic, parallel applications in that their execution modes, heterogeneous environments, and deployment are important characteristics and they determine the requirements and even the performance and scalability of the application.

Not only do the usage requirements that drive distributed applications differ from those of typical parallel applications, but the nature and characteristics of distributed applications are also different from those of parallel applications. Ref [1] introduced the concept of *Application Vectors* – defined as attributes that help understand the primary characteristics of distributed applications, and importantly, provide insight into possible mechanisms for their development, deployment, and execution, and discussed how distributed applications typically differ from traditional HPC applications along each of the execution, communication, and coordination vectors. More often than not, parallel applications are monolithic entities and the challenge is to determine a scalable way to decompose these applications while maintaining stringent requirements on communications. In contrast, distributed applications are mostly either naturally decomposable or are existing, loosely-coupled components and the challenge often is to find mechanisms that permit the effective coordination of these components. In other words, for parallel applications skillful decomposition is required; for distributed application skillful composition is required. Another, almost fundamental difference between distributed applications and traditional parallel applications is the notion of a static execution model versus a dynamic execution model. Most parallel/cluster applications are developed with a model that assumes a specific execution environment with well-defined resource requirements/utilization and control over these resources. Thus, most parallel applications are not developed with the flexibility to utilize additional resources during execution, even though some may have varying resource requirements during execution (arising say, from some "interesting" computational result). Such a static execution model has clearly carried over into the domain of distributed applications and has, in a way, defined the first-generation of distributed applications and, unfortunately, the first-generation of production High-Performance Grids such as the TeraGrid and DEISA. However, given the fact that the computational and scheduling requirements of the individual components can change, as well as the fact that the distributed execution environment is less controlled and more variable than traditional parallel environments, it is fair to assume that distributed applications should, more often than not, be developed so as to explicitly support dynamic execution. As we will discuss, there are many distributed applications that by definition require support for dynamic execution models.

TABLE I
TeraGrid Usage Mode distribution for 2006, the latest year for which data is available. Notice the small proportions of users/applications that utilize multiple resources collectively – concurrently or otherwise

Usage Mode	Number of Users
Batch Computing on Individual Resource	850
Exploratory and Application Porting	650
Science Gateway Access	500
Workflow, Ensemble and Paramater Sweep	250
Remote Interactive Steering and Visualization	35
Tight-Coupled Distributed Computation	10

Scope and Outline: This paper undertakes an integrated analysis of distributed applications and production Grid infrastructure (PGI). Although related to Ref [1], this paper supplements it by taking a different and arguably novel approach. We discuss representative PGI and applications in Section II and III respectively. Our approach is to choose a representative set of four applications; our selection, must, by definition be constrained. We choose applications that are either representative of a broader class of applications, or serve as prototypes of important advances in the style of development or type of distributed applications. Not surprisingly, a time-ordering of the applications shows an increasing level of sophistication. Early applications were mostly trivial generalizations of applications with static execution – spatial and temporal; later applications relatively more *dynamic and distributed* – in execution and data handling (I/O). In Section IV (Critical Perspectives), we provide an integrated but coarse-grained analysis of applications and PGI, that amongst other things, shows that developments and advances in distributed applications have often been out of phase with production infrastructure capabilities and offerings. We hope to both suggest mechanisms to bridge the chasm for current PGI and prevent a repeat in the next generation of upcoming distributed cyberinfrastructure, such as XD and EGI.

II. Production Grid Infrastructure

The large number of production and research Grid projects that have emerged over the past few years is testimony to the potential and importance of distributed cyberinfrastructure. However, we limit the number of PGI analyzed to two; the choice is admittedly determined by the familiarity of the authors, but we believe they capture the main aspects of most PGI. Thus the first PGI studied (TeraGrid) primarily, though not exclusively, caters to computationally intensive applications, and the second (Open Science Grid) primarily to data-intensive applications. Interestingly, similar PGI models exist in Europe, where DEISA and EGEE are the counterparts to the TeraGrid and OSG, respectively.

1) TeraGrid: The TeraGrid [2] is an open scientific discovery infrastructure combining leadership class resources at eleven partner sites to create an integrated, persistent computational resource. It began in 2001 when the U.S. National Science Foundation (NSF) made an award to four centers to establish a Distributed Terascale Facility (DTF). The DTF became known to users as the TeraGrid, a multi-year effort to build and deploy the world's largest, fastest, most comprehensive, distributed infrastructure for general scientific research.

The initial TeraGrid specifications included homogeneous

clusters capable of 11.6 teraflops, disk-storage systems of more than 450 terabytes of data, visualization systems, data collections, integrated via Grid middleware and linked through a 40-Gbps optical network. The initial vision of this system was very "griddy", with users foreseen to be running on multiple systems, both because their codes could run "anywhere", and because in some cases, multiple systems would be needed to support the large runs that were desired. The software that made up the TeraGrid was a set of packages that needed to be identical on all systems.

The TeraGrid has since expanded in capability and number of resource providers – with roughly four expansion phases in 2002, 2003, 2006, and 2007. In 2002, NSF made an Extensible Terascale Facility (ETF) award to expand the initial TeraGrid to integrate Pittsburgh Supercomputer Center's flagship machine. This introduced heterogeneity and thus added complexity to the Grid ideals of the initial DTF, as the common software no longer could be completely identical. This led to the concept of common interfaces, with potentially different software underneath the interfaces. To further expand the TeraGrid's capabilities, NSF made three Terascale Extensions awards in 2003. The new awards added to the TeraGrid access to neutron-scattering instruments, large data collections, additional computing and visualization resources. In 2004, as a culmination of the DTF and ETF programs, the TeraGrid entered full production mode, providing coordinated, comprehensive services for *general* U.S. academic research. In 2005, NSF extended support for the TeraGrid with a set of awards for operation, user support and enhancement of facilities. In 2006, the users of national center supercomputers (at NCSA and NPACI) were merged into TeraGrid, which led to TeraGrid increasing its focus on supporting these users and their traditional parallel/batch usage modes. In 2007-08, three additional TeraGrid resource providers were added.

Using high-performance network connections, the TeraGrid, the world's largest, most comprehensive distributed cyberinfrastructure for open scientific research, now integrates high-performance computers, data resources and tools, and high-end experimental facilities around the US. The TeraGrid software stack is now a set of capabilities with common interfaces. Sites can choose to either include or not include these kits as part of their local software.

Table I provides instructive information about how the TeraGrid is primarily used. Interestingly, the TeraGrid is primarily used as several independent HPC resources, and often not collectively as a single system. Users submit jobs to the batch queues of the particular system on which they want to run their application. Users are encouraged to use the TeraGrid User Portal to monitor the batch queues and to use the batch queue predictor to assist them in selecting the systems that will be best suited to their needs. Users may request special handling of jobs, including access to dedicated system time, to address special job processing requirements. Single-identity/credentials and global file-systems have been common outstanding requests from the user community; the former is partially implemented now, and the latter is being researched, with some initial implementations on a subset of systems existing now.

TABLE II
TYPES OF APPLICATION RUNNING ON THE OPEN SCIENCE GRID

Application Type	Characteristics & Examples
Simulation	CPU-intensive, Large number of independent jobs. e.g., Physics Monte Carlo event simulation
Production Processing	Significant I/O of data from remote sources e.g, Processing of physics raw event data
Complex Workflow	Use of VO specific higher-level services; Dependencies between tasks e.g., Analysis, Text mining
Real Time Response	Short runs; Semi-guaranteed response times, e.g., Grid operations and monitoring
Small-scale Parallelism	Allocation of multiple CPUs simultaneously; Use of MPI libraries, e.g., Protein analysis, MD

2) Open Science Grid : The Open Science Grid (OSG) [3] is a collaborative activity, where stakeholders come together to pool resources and build a shared environment for the common benefit of all stakeholders. The largest contributors of resources and also the largest users of the OSG are from the High Energy Physics community, specifically associated with the Large Hadron Collider (LHC), ATLAS and CMS (Compact Muon Solenoid) experiments. The integrated software that OSG offers on its resources is the Virtual Data Toolkit (VDT) [4], which includes many (possibly-customized) software packages that enable access to the OSG compute and storage resources. The data and processing needs of the LHC led to the LHC Computing Grid (LCG). This was brought together with the Grid3 project and other, smaller projects, to create the OSG and its European peer, EGEE (Enabling Grids for E-sciencE) around 2004-5.

In contrast to TeraGrid, the OSG has encouraged distributed resource utilization, for example, OSG users must use a remote job launch mechanism to start their work, unlike TeraGrid.The OSG is also different from the TeraGrid in its model of resource aggregation. Resource Providers have to apply and be approved by a central (funding) agency in order to join the TeraGrid, whilst interested parties can *voluntarily join* the OSG.

Applications running on the OSG span simulation and analysis of small to large-scale (CPU days to centuries) scientific application runs. The OSG facility architecture has special utility for high-throughput computing applications. The characteristics are large ensembles of loosely coupled parallel applications for which the overhead in placing the application and data on a remote resource is a fraction of the overall processing time. Also, added value is available to computations (loosely coupled and able to run on heterogeneous sites) that can take advantage of opportunistic resources. Table II summarizes the types and characteristics of applications running on the OSG. Any particular application may have one or multiple such characteristics: simulation, production processing, complex workflow, real time response and small-scale parallelism.

Usage of the computational resources through the OSG is one of three modes: Guaranteed through ownership by the user's community; Agreed upon through policies between the resource owner and the user's community; Or opportunistic use through resource sharing. When communities make their resources accessible through the OSG, they define the policies of their use. Resource owners retain control of their resources

including: prioritization of use, which communities and users to support, and policies of access. As members of the OSG consortium, resource owners are encouraged to provide access to available resources (typically of the order of 10% or more) to other communities for specific computational goals as well as dynamic use of currently available cycles. Opportunistic use is a hallmark of the OSG.

III. Specific Applications

The choice of applications discussed is determined by the coverage they provide, the specific class of applications they represent, and/or the fact that they represent an underlying trend or shift in a capability. In addition to a brief description of each application, we attempt to provide a description of degree of distribution of the application and the key paradigms that each application represents, as well as how applications have evolved as the underlying infrastructure changed.

1. SF Express: The Synthetic Forces Express (SF Express [5]) project began in 1996 with the goal of investigating the use of high-performance computers as a means of supporting very large-scale, distributed interactive simulations. It was one of the first successful examples of metacomputing, utilizing separate stand-alone multi-site supercomputers, used initially to demonstrate a scalable communications architecture supporting tens of thousands of vehicles. It emerged as a distributed application harnessing multiple high-performance computational resources to meet the demands of large-scale network-based distributed simulations. SF Express was "Grid-enabled" – able to incrementally incorporate services and interfaces afforded by the evolving Grid environment. The underlying paradigm is a data parallel application structure where entities are simulated in parallel and communication is performed based on "regions of interest". A hierarchical communication paradigm was used to aggregate communication across gateway machines at each site. MPI was used for inter-machine communication.

The first major milestone for SF Express was achieved in November 1996, with the successful simulation of more than 10,000 vehicles using the 1024-processor Intel Paragon. In 1997, SF Express was extended to include multiple high-performance computers. A simulation of 50,000 vehicles was achieved using 1,904 total processors on six computers at sites distributed across seven time zones.

To improve the functionality and validity of large-scale distributed simulation experiments and to address issues including scenario distribution, resource configuration, resource management, information logging, monitoring, and fault tolerance, SF Express used services provided by the Globus Metacomputing Toolkit. In 1998, a record-breaking simulation was conducted using 100,298 vehicles – the largest, distributed, interactive battlefield simulation up to that date. This simulation was executed on 1,386 processors that were distributed over 13 computers among nine sites that spanned seven time zones. Global and local services were decoupled, allowing the application to run in a flexible, resource-aware environment. As far as underlying distributed infrastructure, SF Express leveraged Globus support for remote execution and communication. Significant changes to the Globus architecture and interface over the years meant that the application had to undergo cumbersome changes too. While co-allocation would, in principle, be needed for SF-Express to run, it was not available at the time, so it appears that manual reservations and on-demand access to the machines for the executions were required.

2. SETI: The Search for Extraterrestrial Intelligence (SETI) [6] uses radio telescopes to listen for signals from space. Such signals are not known to occur naturally, so a detection would provide evidence of extraterrestrial technology. Radio telescope signals consist primarily of noise (from celestial sources and the receiver's electronics) and man-made signals such as TV stations, radar, and satellites. SETI projects analyze the data digitally, with a virtually unlimited need for computing power. Previous SETI projects used special-purpose supercomputers located at the telescope to do the bulk of the data analysis. In 1999, a virtual supercomputer composed of large numbers of donated wide-area Internet-connected computers (SETI@home) was launched. SETI's paradigm is master-worker, with the master having a near infinite supply of work (tasks) for the workers. The SETI client runs continuously when the machine is idle (screen saving is running). It fetches a chunk of data from a server, analyzes the data, and reports the result back to the server. The same data may be analyzed by multiple computers for integrity checks. Volunteer computers are incentivized by gaining "credits" for performing SETI tasks. The degree of distribution within SETI is vast, containing individual PCs spanning many continents, all running asynchronously.

SETI did not rely upon any existing production Grid infrastructure and still does not. It requires continuous on-demand access to resources, a requirement not met by the queue-based paradigms of supercomputer-based Grids. SETI led to the formation of BOINC in 2002-2003, an open-source volunteer computing framework. BOINC is both a middleware infrastructure and a specific deployment platform. The current BOINC deployment contains over 500,000 computers and 1.7 PFlops of computing power. BOINC continues to evolve supporting newly emerging clients (e.g. GPUs) and advanced features such as client data caching and project-defined replication and verification procedures. Many groups have used BOINC successfully to establish their own volunteer Grids.

3. Montage: Montage [7] is an exemplar workflow application that is used both for constructing astronomical image mosaics and for testing workflow software and ideas. It consists of a set of tools, each of which is a C executable. Each executable reads and writes to/from files. Given a set of initial images to be mosaicked, and the bounds of the output image, it reprojects the input images to the output projection, finds overlaps between pairs of images, fits planes to the overlaps, solves a set of equations to minimize the overlap differences, applies the calculated background rectifications to each reprojected image, then co-adds the reprojected, background-rectified images into a final mosaic. Most of these stages involve a number of steps that can be done in parallel, e.g. the reprojection of each of the input images, but there are interstage dependencies.

Putting this together, there are a few different ways to use the Montage tools. One is to use a single computer by running a

script that goes through each stage, and within each stage runs through each element of processing. On a parallel computer, the various elements within a stage can be parallelized over the set of processors; Montage supplies MPI routines to do this. Finally, the dependencies between the processing can be represented as directed acyclic graph (DAG), using a Montage-supplied tool, and this DAG can then be enacted on a single computer, a parallel computer, or a distributed set of computers, using a variety of tools, but often in the distributed case using Pegasus, Condor DAGman, and Condor-G. This version of Montage is often used behind a portal, where the users don't need/want much knowledge about the resources being used.

For the case of a single computer or a parallel computer, or even a distributed computer with a shared global file system, each Montage component will have access to the outputs of components that have already run. For a general distributed set of computers, output data from one tool running on one system needs to be transferred to the system on which the next tool is going to run. The Pegasus/DAGman/Condor-G set of tools handles this automatically. However, there are issues remaining with the mapping of executables to resources in the distributed case. Ideally, one would like all of the mapping and execution to be dynamic and "just-in-time", but this is not possible today. Also, with schedulable or monitorable networks, one would like to be able to schedule work that includes data transfers with knowledge of network bandwidth over time, or the ability to reserve bandwidth for when needed.

Montage was developed to meet science goals and to be customizable, even on a single computer, with the knowledge that just using one computer would be too time consuming for many real problems. Fitting the results of this process into the parallel and distributed computing arenas has been fairly straightforward, as the idea of a set of executables that can be described by a graph is probably the most general idea that is supported on all production infrastructures.

4. LEAD: Linked Environments for Atmospheric Discovery (LEAD [8])aims at making meteorological data, forecast models, and analysis and visualization tools available to anyone who wants to interactively explore the weather as it evolves. LEAD software enhances the experimental process by automating many of the time consuming and complicated tasks associated with meteorological science. The LEAD workflow tool links data management, assimilation, forecasting, and verification applications into a single experiment. The experiment's output also includes detailed descriptions of the product (metadata). LEAD includes a portal that hides the details of the resources (data, models, analysis and visualization tools, and HPC resources).

A major underpinning of LEAD is dynamic workflow orchestration and data management: Workflow Orchestration for On-Demand, Real-Time, Dynamically-Adaptive Systems (WOORDS). WOORDS provides for the use of analysis tools, forecast models, and data repositories not in fixed configurations or as static recipients of data, as is now the case for most meteorological research and operational forecasting technologies, but rather as dynamically adaptive, on-demand, Grid-enabled systems that can, a) change configuration rapidly and automatically in response to weather; b) continually be steered by new data; c) respond to decision-driven inputs from users; d) initiate other processes automatically; and e) steer remote observing technologies to optimize data collection for the problem at hand. Although mesoscale meteorology is the particular problem to which the WOORDS concept is being applied, the methodologies and infrastructures being developed are extensible to other domains such as medicine, ecology, oceanography and biology.

Some of the parts of the LEAD system that are dynamic include: running models more frequently when the weather is changing more quickly, running models on different grids as needed to resolve dynamic features, running ensembles of varying size, varying the number of instruments used for data ingestion, making dynamic use of computing platforms and networks, and allowing a variety of usage scenarios.

Some parts of LEAD are similar to Montage, but it is a much more complex application overall. The authors are not aware of any general computer science work being done with LEAD outside of contributors to the LEAD project, unlike Montage, where at least ten different computer science projects have chosen to use Montage to demonstrate some aspect of their systems. Additionally, LEAD is often data-driven, where Montage is purely user-driven. While Montage can take advantage of many types of computing systems, LEAD was written specifically for distributed systems.

IV. A Critical Assessment of Distributed Applications and Production Grid Infrastructure

The discussion in the previous sections of this paper and earlier references [1] clearly indicate that distributed computational infrastructures – both existing (Grids) and emerging (Clouds) – provide tremendous capabilities and opportunities, and can potentially support scientific and engineering investigation at unprecedented scales. However, despite an overwhelming need and great potential, the number of production-level, large-scale high performance distributed applications has been relatively small and the effort required to bring these applications to fruition has been high. Some reasons include, but are not limited to: (1) in general, distributed applications present unique challenges resulting in increased complexity, and often have complex structures that compose multiple patterns at different levels as well as at different phases of application execution; (2) emerging distributed computational infrastructures present significant operational challenges, which constrain the capabilities provided to distributed applications; (3) appropriate abstractions that simply and efficiently support prevalent application patterns and that can be used by the applications to address development and deployment challenges are lacking, and finally (4) issues of policy, infrastructure design decisions (narrow versus broad Grids [9], research infrastructure versus production infrastructure) and the harmonization of its development with application requirements are also important barriers.

The lack of explicit support for addressing these challenges of distributed applications on distributed infrastructure become both self-fulfilling and self-perpetrating. For example, a con-

sequence of the small number of distributed applications is that most PGI do not support these application as well, which in turn, further increases the effort required to develop new applications. For example, as previously discussed, there are a relatively small number of distributed applications on the TeraGrid, which led to a focus on supporting single site applications (or more generally, legacy localized applications.)

Looking back, production Grid application and deployments can be analyzed along multiple axes: (1) the way the application formulation perceives and uses the Grid environments, (2) the ability of application formulations to address the challenges of distributed Grid environments outlined above, and (3) abstractions and programming support available for implementing and running the applications on Grids. For example, in cases where no programming support was available and application development and deployment relied on the "hero programmer", there has been very limited sustainable success, if any. However, even in cases where significant programming support was available, success strongly depended on the suitability of the application formulation and the ability of the distributed computing paradigm used to address the Grid challenges outlined above. For example, high-throughput or parameter sweep type formulations based on master/worker or bag-of-tasks paradigms, which perceive the Grid as a pool of resources, have been extremely successful as these formulations can very effectively handle heterogeneity, dynamism and failures, and are supported by relatively mature programming systems such as Condor and BOINC. Similarly, loosely coupled application workflows that also handle these challenges and are supported by programming systems have seen similar successes. On the other hand, applications based on the MPI paradigm have had limited success even though they were supported by MPICH-G and were shown to result in an overall reduced time-to-completion, even if there was a minor increase in the total CPU time consumed [10]; this suggests that generalizing from the traditional HPC to distributed HPC requires support beyond programming abstractions and systems. However, as the use of workflow systems has demonstrated, some HPC applications can be developed by aggregating components (which may wrap complete applications), executing in their own native deployment environments. This aspect needs further investigation, and has not been effectively exploited in the Grid community.

Furthermore, and somewhat ironically, distributed applications (as well as the supporting systems) have typically been developed/formulated to hide from the heterogeneity, dynamism, and distributed nature, as opposed to developing them in a way that they can embrace these features. Heterogeneity and dynamism have been identified as major barriers in the uptake of distributed systems and the development of distributed applications. This approach is clearly not sustainable; it results in applications that are brittle and very closely tied to their target Grid infrastructure [11]. The rest of this section further expands on these observations.

1) Programming Systems for Distributed Applications: Many applications have used distributed infrastructure to advance understanding in their disciplines. A number of applications have been able to use existing programming systems and tools, without having to worry about customized extensions. For example, multiple applications have used the abstractions provided by MPICH-G2 to address distributed communication and coordination. Co-scheduling capabilities have existed (HARC and others) and have been used/needed in several applications. Similar observations can be made in the way dataflow coordination is supported across multiple resources.

However, there also exist a number of applications that have had to implement new capabilities at one or more levels, i.e., at the application, programming system, middleware, and/or infrastructure level. Such crosscutting (functional and non-functional) issues include hiding delay, providing fault-tolerance, addressing data/system uncertainty, addressing security (including privacy and integrity issues), and supporting dynamic discovery and adaptation, among others. Specific examples include composition (workflow) systems that enable the specification (and, in some limited context, enforcement) of QoS requirements, and component/service-based programming systems that are extended to support dynamic resource-sensitive bindings and policy-driven runtime adaptations. In general, where abstractions, underlying tools, and programming systems have been developed to support just one application, or at most a few, the level of success of these efforts depended not only on the resulting programming system, but also on the suitability of the underlying paradigm for Grid environments.

It is important to understand why some programming systems and tools have been widely used and others have not. Condor and BOINC are two of the more widely used tools supporting distributed applications, which has as much to do with the programming models they support and enable as any other reason. The success of BOINC may be based on its singular capability: to provide applications with huge resource pools. Donated nodes do not interact and the model is fairly simple. Programmers may be willing to learn new paradigms and restructure programs if the new model is simple. In addition, as an added feature, BOINC is flexible. Extending widely used tools with which users are already comfortable to distributed versions seems to be a promising pathway. A similar story holds for Condor: users can program their applications in familiar languages and the environment takes care of distribution.

As was mentioned, many existing programming systems are not widely used by application developers. Often, developers prefer to "roll out their own" capabilities, even though many of these capabilities already exist in available programming systems. Utilizing existing programming systems would enable the developer to focus on specifics of the application, rather than the implementation of such core infrastructure. But as this is often not possible, the lack of reusability thus has the effect of further supressing distributed application development. Thus, it can be asked in this context, why have application developers not considered using capabilities from an existing programming system (such as CCA) to support the required capability? This may be due to a number of factors: (a) Many programming systems lack robustness when used by developers in scientific applications – for instance, they may have limits on scalability.

(b) The level of maturity and development schedule of the programming system may not align with the requirements of the application developers. (c) It may be difficult to isolate a specific capability from a programming system, thereby limiting re-use by an application developer requiring that capability. In general, programming systems fall short of the requirements of distributed applications due to:

i. Incompleteness: Programming systems and tools are often incomplete or inflexible with respect to application needs, e.g., tools that support the master-worker paradigm often only address failures of workers and not of the master. Furthermore, they may not address non-functional issues such as security and privacy, which may be critical for some applications.

ii. Customization: Programming systems and tools also tend to be customized to applications and their requirements, with limited concern for reuse or extensibility. Applications and tools are also often highly dependent on (and tuned to) a specific execution environment, further impacting portability, re-usability, and extensibility.

2) Distributed Applications and Production Grids: It is important to mention that we are not going to address the operational issues associated with Production Grid Infrastructure, which although both non-trivial and interesting, are not in the scope of this distributed-application-centric discussion. Many of the operational challenges associated with production Grid infrastructure, outlined in Ref. [12], will remain valid independent of the issues that we focus on. The match between the application formulation and the Grid infrastructure significantly impacts how successfully the application can use the Grid. In case of applications that are performance critical, it is necessary to customize the environment and architecture, but for the majority of applications, the greater concerns are those of simplicity, extensibility, low time-to-productivity, and usability. Just as the choice of the right algorithm can overcome the use of "faster" or "better" hardware, to an extent, the choice of the appropriate infrastructure, in terms of what usage modes, policies and types of computing are supported, can override simple architectural or performance differences.

Evolution of Supported Capabilities and Infrastructure: Understanding the evolution of certain infrastructure capabilities in response to application and user needs is both instructive and interesting. Given OSG's need to support HTC, Condor has evolved from a scavenging system in the late 80s to becoming the basic block for Grid infrastructure in the 2000s. Condor Flocking, which provides aggregation of resources, is a fine example of a continuous transition versus discontinuous transition. Similarly, experiences from SETI@Home led to BOINC, which was then used for other @Home applications.

Science Gateways on the TeraGrid grew out of a number of computationally-saavy application developers who realized that others in their communities would benefit from the TeraGrid resources, if the interfaces to use the resources could be simplified. The gateways that have been developed often use a graphical user interface to hide complexity, but provide capabilities such as: workflows, visualization software and hardware, resource discovery, job execution services, access to data collections, applications, and data analysis and movement tools. Another common aspect of many science gateways is the idea of a community account, where the gateway developer applies for a TeraGrid allocation on behalf of a community, and members of that community then use the allocation without individually having to register with TeraGrid. Because of this, the TeraGrid does not currently have a good way of reliably counting science gateway users, but the number of cycles used through science gateways increased by a factor of five from 2007 to 2008. By working with some of the initial gateways developers, the TeraGrid has developed capabilities that can be used by other developers to build new gateways.

However, there are several examples where the requirements of distributed applications are often out of phase with the deployed capabilities of infrastructure. One example of such out-of-phase development and deployment, is the capability of co-scheduling on production Grids. Another is the ability to create a DAG and then just run it over multiple resources, e.g. on the TeraGrid. Co-scheduling is an interesting case study, because it involves both policy and technical challenges. The policy issues have been a barrier because of the inability or the lack of willingness of HPC centres to relinquish the batch-queue mode of operation. The resource providers' preference for batch-queue computing and corresponding emphasis on overall utilization of HPC resources has inhibited other modes of computing, such as urgent computing, ensemble, and QoS-based computing (e.g., user X will be allowed Y number of jobs over period Z.)

There are also external reasons to promote the use and importance of specific classes of applications. For example, the emergent ease and abundance of sensor data and thus the effectiveness in coupling real-time simulations to live sensor data has driven the class of dynamic data-driven distributed applications scenarios (DDDAS). When coupled with the maturity of workflow tools (a natural framework for the composition of DDDAS,) an increase in LEAD-style applications is to be expected. There are similar reasons why other distributed applications "come of age"; anticipating these trends and supporting them on PGI would be beneficial to the wider scientific community. Currently, neither the OSG nor the TeraGrid can support large-scale DDDAS out of the box and without significant customization. OSG supports opportunistic and HTC but not large-scale HPC, whilst the TeraGrid supports HPC but does not natively support dynamic or opportunistic requirements.

Underlying infrastructure and capabilities change quicker than the timescale over which scientific distributed applications once developed are utilized. In addition to SFExpress, another example of an application that has been around for a long time and has successfully transitioned from parallel to distributed infrastructures is Netsolve, which has branched and evolved into GridSolve. It is worth noting that although underlying distributed infrastructure may change rapidly, for example, the sudden emergence and prominence of Clouds, the basic principles and requirements of distribution do not change, viz., the fundamental problem of coordinating distributed data and computation remain. Therefore it is imperative that distributed

application developers consider developing their applications using programming systems, tools and interfaces that provide immunity from the natural evolution of infrastructure and capabilities. Well designed distributed programming abstractions can be critical in supporting these requirements [13].

3) Miscellaneous Factors: Various other factors have also contributed to the current state of affairs with production Grid infrastructures and distributed applications. These include the role of standardization, funding agencies, and other geopolitical issues. For example, the changing funding landscape in the US has directly impacted the progression of Grid research, as well as the design of the current national Grid infrastructure. For example, the TeraGrid has moved from its original vision of a run-anywhere distributed infrastructure, with the middleware and tools to support this vision, to the remit of supporting all scientific users and applications. The changing focus of the funding agencies may also be one reason why more researchers have focused on localized and high-throughput computing, rather than on more complex problems such as robust distributed workflows that can *scale-out*, or making co-scheduling capabilities widely available.

Another significant factor worth discussing is the role of standardization, as driven by bodies such as the OGF [14]. It has been argued that these efforts, though well intended and useful, were premature, i.e., that the nature of requirements of Grid infrastructure and applications were still evolving, and not ready for standardization. Yet again, there is a bit of a chicken-and-egg situation: applications don't really utilize distributed resources and thus do not cry out for standards, and thus resource providers (or funding agencies) don't feel compelled or motivated to support standards. Focused and simple standards are a good idea that represent efforts in the right direction; they should be encouraged, especially as the current crop of PGI transition into their next-generation incarnations. For example, the job submission specification troika of JSDL, HPC-Basic Profile and OGSA-Basic Execution Service (BES) [15] provide the basic community agreement to standardize job-submission across different heterogeneous resources.

V. Conclusions

In summary, with the luxury of hindsight, many theories and explanations can be developed to explain the current state of production Grid infrastructures and distributed applications. For example, it may be said that perhaps there were unrealistic expectations, in terms of how challenging it would be to design (and operate) Grids that would support a range of distributed applications, or the effectiveness of designing production Grids as extended clusters around traditional HPC applications. This has prevented developers from clearly understanding the requirements of targeted distributed applications and their implications on the design of software and hardware infrastructure. Production Grids for distributed applications clearly warrant fundamentally different sets of assumptions and design decisions. Conversely, it can be argued that the effort of transitioning HPC applications to effective distributed HPC applications was underestimated.

There isn't a single unifying paradigm (such as message-passing for parallel programming) that can be used to enable distributed applications; this is a reflection of the fact that the concerns and objectives for distributed applications are both large in number and broad in scope. This has important ramifications. For example, there isn't a single (if not simple) design point (such as performance for HPC applications) around which to architect production distributed systems. Additionally and as a consequence, at one level, the range of tools, programming systems and environments is bewilderingly large, making extensibility and interoperability difficult; at another level, there exists insufficient support for the development of distributed applications integrated with deployment and execution of even common usage modes. Addressing the issues and gaps identified *universally* for all applications and usage modes will not happen anytime soon, but addressing them in the context of a well-defined set of scientific applications and usage modes on PGI is both achievable and should be the next step. We hope that the upcoming generation of PGI – XD, PRACE, EGI, will find some of this analysis timely and useful.

Acknowledgements

This paper is an outcome of the UK e-Science Institute sponsored Research Theme on Distributed Programming Abstractions. We would like to thank many people who participated in the workshops and meetings associated with the theme. We would like to thank Ruth Pordes and Paul Avery for sharing their insight into OSG.

References

[1] Abstractions for Large-Scale Distributed Applications and Systems, submitted to the ACM Computing Surveys, S. Jha et al, draft available at: http://www.cct.lsu.edu/~sjha/dpa_publications/dpa_survey_paper.pdf.
[2] C. Catlett. The Philosophy of TeraGrid: Building an Open, Extensible, Distributed TeraScale Facility. In *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on*, pages 8–8, May 2002.
[3] R. Pordes et al, *New science on the Open Science Grid*, 125, pg 012070, Journal of Physics: Conference Series (2008).
[4] The Virtual Data Toolkit web site, http://vdt.cs.wisc.edu/.
[5] Synthetic Forces Express, http://www.cacr.caltech.edu/SFExpress/.
[6] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.
[7] J. C. Jacob, D. S. Katz, G. B. Berriman, J. Good, A. C. Laity, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, T. A. Prince, and R. Williams. Montage: A grid portal and software toolkit for science-grade astronomical image mosaicking. *International Journal of Computational Science and Engineering*, 3, 2007.
[8] B. Plale et al, *CASA and LEAD: Adaptive Cyberinfrastructure for Real-Time Multiscale Weather Forecasting*, Computer, 39 (11), 2006.
[9] S. Jha et al, Using clouds to provide grids with higher levels of abstraction and explicit support for usage modes, Concurrency and Computation: Practice and Experience, 21 (8), 2009.
[10] P. Chakraborty et al, *Novel Submission Modes for Tightly-Coupled Jobs Across Distributed Resources for Reduced Time-to-Solution* Phil Trans R Soc A Vol. 367, 1897 (2009).
[11] Parashar and Brown. Conceptual & Implementation Models for the Grid. *IEEE, Special Issue on Grid Computing*, 93(2005):653–668, 2005.
[12] W. Gentzsch. Top 10 Rules for Building a Sustainable Grid, OGF Thought Leadership Series, http://www.ogf.org/TLS/documents/TLS-Top10RulesForSustainableGrid.pdf.
[13] A Merzky et al, Application Level Interoperabilty between Grids and Clouds, Workshop on Grids, Clouds and Virtualization, GPC09 (Geneva).
[14] Open Grid Forum, http://www.ogf.org.
[15] OGF Specification Summaries http://www.ogf.org/UnderstandingGrids/grid_specsum.php.