Grids meet Too much Computing, Too much Data and never Too much Simplicity

Geoffrey Fox[1,2,3] and Marlon Pierce[1]
[1]Community Grids Laboratory
[2]Department of Computer Science
[3]School of Informatics
Indiana University
{gcf, marpierc}@indiana.edu

Challenge

Let us discuss:
Grids are taking too long to solve the wrong problem at the wrong point in stack with a complexity that makes friendly usability difficult. We furthermore observe that Grids (as envisioned c. 2001) are being pressured by both emerging new computing resources (multicore, cell processors, GPUs, reconfigurable computing, etc) and alternative approaches to service architectures (collectively, Web 2.0). We thus believe it is time to reappraise Grids—both the nature of the resources that they aggregate and the middleware that glues these resources together.

In spite of the unclear technology directions, e-Science and more generally e-moreorlessanything are thriving with the advantages of distributed enablement of many fields very clear.
Abstracted from http://grids.ucs.indiana.edu/ptliupages/publications/GridSandwich.pdf

1. Too much Computing

Over the last 20 years we have seen an increase in the power of computers by roughly a factor of a million and this has enabled computational science to become a major force in an ever increasing number of academic fields. This has been nurtured by substantial institutional investment in computational infrastructure spanning campus, laboratory and national scale. This investment involves the use of both grids and parallel computing to increase computing capabilities by aggregating computers together in a distributed or local fashion respectively. Different problem classes are suited to different aggregation architectures; closely coupled problems are suited to closely coupled (i.e. low communication latency) parallel computer systems, loosely coupled problems are suited to loosely coupled (i.e. high communication latency) distributed computer systems. This hybrid strategy ensures an e-infrastructure (Cyberinfrastructure) of broad applicability. This capability must support simulations, the fusion analysis and analysis of data from repositories, instruments and sensors, the distributed users as well as their integration. Such a scenario supports e-moreorlessanything including e-Science, e-Research, e-Business and even e-digitalmedia and e-OpenSocial for the Net-Gens (or more appropriately for this article the e-Gens).

The USA Federal HPCC (High Performance Computing and Communication) initiative started in 1991 was a major driver for both technologies and applications for computational science with parallel computing and simulation as the focus. Following HPCC, Grids and Cyberinfrastructure/e-Infrastructure blossomed and there was a growing recognition of the importance of data which was not stressed with HPCC. In spite of these changes, the community was obsessed with performance shown in many ways. The well known Top500 list measures the stature of your institution by the performance of its supercomputer on matrix algebra. Much of Grid technology aims at linking computers across multiple administrative domains leading to increased total performance but at the cost of significant security challenges that translates into complex Grid virtual organization technology. Further there has been great interest recently in harnessing the potential of graphics boards to deliver very high performance on cluster nodes. When we started this (say around 1980 with the Cray-1 and XMP), we definitely had too little computing as only unrealistic 2D simulations were possible and this motivated much of the early parallel computing research. However the situation is changed and perhaps now most applications have the opposite problem – there is in fact *Too much Computing*.

Some fields can still use essentially unlimited (petascale) parallelized computing; a few examples are stewardship of the nuclear stockpile for national security and Quantum Chromodynamics or globular cluster simulations in fundamental science. However even in e-Research these are probably the exception; the common (average) case of computational science does not need petaflop computers. In particular the “new sciences” such as life science and social science definitely need e-Science but typically modest peak performance. Similarly, the digital humanities have a growing need for distributed computing infrastructure (mass storage, networking, middleware, and some processing) generally but no need for very high end computation. A field needs to be quite mature to develop algorithms and accurate phenomenology needed for useful petaflop computing. We have already enough computing in many parts of e-Research; we need to develop other aspects of Cyberinfrastructure. The situation is even clearer for mass market systems whose CPU chips will be 32-128way parallel in 5 years time, but we currently have little idea how to use them on commodity systems – especially on clients. There are perhaps at most 2 releases of standard software (such as Windows or Microsoft Office) in this time span and we need to find value in these new chips for the broad market so that the multicore instantiation of Moore’s law (roughly constant clock speed, increasing core density) will lead to improved performance. We need to address *Too much Computing* here with approaches that can be implemented in next 3-5 years.

2. Too much Data

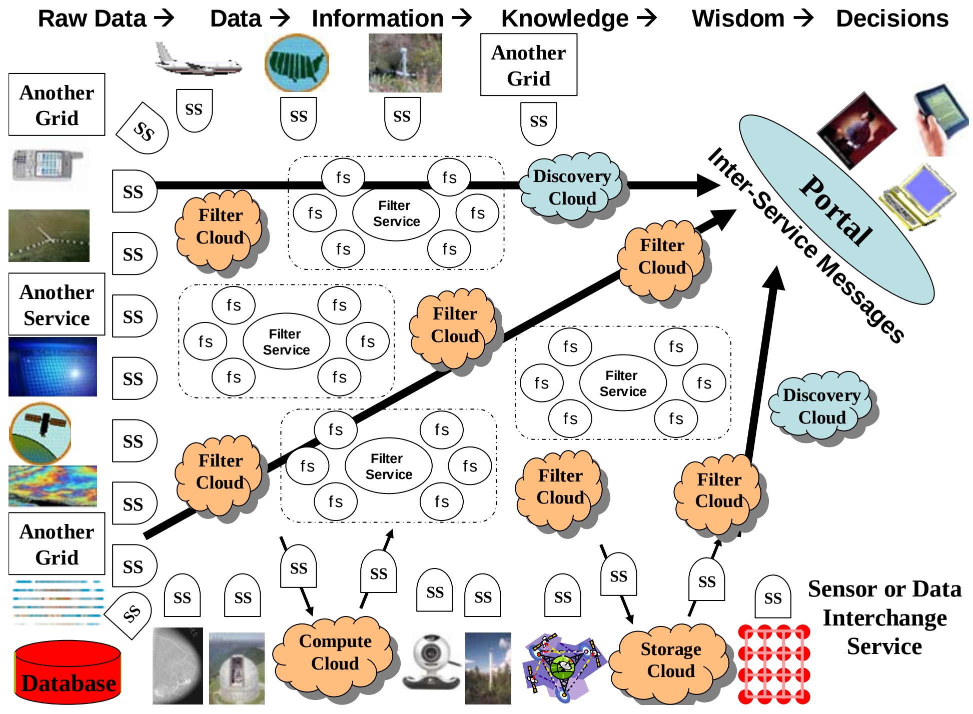

*Figure 1: Information architecture combining Web 2.0 and Grid Concepts. Wisdom is obtained by fusing and transforming data that comes from sensors, instruments, services, Grids and Clouds. Data is transformed by filters that perform data analysis, transformation, assimilation or production from simulations. Traditional Grids expose constituent services as illustrated by Filter Service surrounded by other (fluff services **fs**) services in dashed rectangles. Compute, Storage and Filter clouds hide this detail and expose data interfaces. Discovery is needed for both Clouds and Grid services and all components are linked by messages*

The deluge of increasing data is a clearly recognized feature of e-moreorlessanything including 30 billion web pages with an increasing number of two-way interactive pages from sites like Flickr and YouTube. In e-Science, there are many observatories, satellites, sensors, high throughput screens and instruments whose data products also track Moore's law with the LHC soon to generate 10 petabytes of data per year.
These data sources of course feed repositories that support scientific discovery which also itself produces further auxiliary data adding to the *Too much Data* scenario. However there are interesting differences between the *Data* and *Compute Deluges*. Data is intrinsically distributed as both sources and the managing organizations are distributed. Further multidisciplinary research and commodity mashups link information of different kinds at different places. The data challenges involve its management, federation, access and analysis. This involves computing but architecturally the data and computing are naturally co-located to avoid large

communication costs. Intel proposed the RMS (Recognition Mining Synthesis) class of applications corresponding to data-mining and gaming as enabled by future multicore chips. It is clear how this works for servers and especially servers near data sources/repositories but Internet Clouds, Supercomputer centers and Clients are not clearly co-located near relevant data. Clients could host data from local (say video and environmental) sensors plus data fetched from the network (Intranet and Internet). The latter might be mined automatically by the client to provide an "intelligent environment" for user sessions. Most data mining algorithms can be efficiently parallelized as long as the datasets are large enough. Thus we imagine that data-mining of this "Too much data" will use up the "Too much computing" on client-side PC's.

Internet storage and computing clouds as well classic supercomputer centers can be used as shown in figure 1 where the data deluge flows through the computing deluge leading to wisdom and effective decisions. We term the Clouds through which data, information and knowledge flows as Filter Clouds to indicate that they input and output streams of data. Clouds have an attractive interface defining the input and output data but how they and traditional computer centers cope with *Too much nonlocal data* is not clear. Perhaps this will drive the deployment of networks of even higher performance; this could be practical as its bandwidth not latency that is relevant in costs scenarios represented by figure 1

3. Never Too much Simplicity

There is no precise definition of Web 2.0, but it is operationally defined by a set of technologies (JSON, AJAX, etc) and a wide range of Web sites supporting user interaction among themselves (social networking) and with resources such as images and video. Media sharing and bookmarking are structured to allow communities (i.e. virtual organizations) to grow up around resources. Similarly peer production sites allow users (people in communities) to select and rate presented information. There are also very popular capabilities like Blogs and Wikis supporting communication either broadly or within an organization. These capabilities provide attractive tools enhancing collaboration in all areas of e-moreorlessanything.

Google maps illustrate the power of Web 2.0 deployment of important capabilities – in this case geographic information systems. Google Maps and related systems have transformed GIS technologies not just through their interactivity (enabled through innovative uses of AJAX, JSON, and image tile caching), but also through the simplicity they bring to development. As we will discuss later, Google Maps is by far the most popular API for mash-up building. Its sophistication and power are accessible through simple APIs and data models (KML) that can be manipulated by anyone with basic programming skills.

Web 2.0 has also encroached on more traditional territory of cyberinfrastructure. Web 2.0, with consequences that may be as transformational as Google Maps has been to GIS. Start Pages challenge portals for access to services and Web 2.0 mashups challenge workflows for integration of services. Further Web 2.0 cloud systems such as Amazon web services directly support the distributed storage, data management, and computing that were up to now the distinctive feature of Grids.

These clouds address “commodity usage” and do not currently provide massively parallel systems for the high end users. However, they do make the important distinction (missing in, for example, the OGSA use cases) between deployment and development. Deployment has very sophisticated challenges, and the Global/Open Grid Forum has attempted to define a standard set of use cases for this. Unfortunately the deployment scenarios often overlap with the Grid development process—that is, building a Grid and developing for a Grid are not adequately distinguished. In contrast, cloud systems, while at least as sophisticated as conventional Grid deployments, do a much better job at defining simple developer interfaces. Resource allocation, data replication, and security are all sophisticated capabilities in computing clouds, but they are hidden from the typical application developer.

Web 2.0 capabilities and technologies are simpler than their Grid counterparts and are user friendly for developers and users. For example as shown in figure 1, Cloud systems are “Grid islands” which only expose the key interface to input and output data and control information. Grids adopt the Web service philosophy where all services are exposed allowing greater power but at greater complexity. Originally we expected that Web Services would dominate a new generation of Enterprise software and that distributed systems would leverage commercial investment in this field and be built in terms of Web Services. However this is not what happened. There is general agreement on service architectures but Web Services were characterized by very specific specification of operating environment through WSDL Interfaces and SOAP header elements. This leads as shown in figure 1 to Grids where individual services are visible and potentially interoperable; as discussed above Clouds have a more limited but friendlier interface. Grid services tend to expose even more detail than Web services. Web 2.0 systems may consider such information important but leave such details to the application allowing a much simpler hosting environment. Rather than Web services dominating practical distributed systems, they will have some use but their complexity and Grid refinements are likely to be ignored as they are accessed by a sea of Web 2.0 services and mashups; Web 2.0 achieves interoperability by simplicity rather than the detailed WS-* specifications. Web 2.0 teaches us that we only need interoperability at a few sweet spots; user data formats at cloud interfaces and the lowest level protocols typified by HTTP transport. Web 2.0 teaches to re-use wherever possible; scripting rather BPEL for workflow for example. Web 2.0 teaches us the power of distributed simple specification; developer uploads of API’s to http://programmableweb.com rather than the stultified UDDI; use of microformats rather than large ontologies; rapid deployment of modules rather than lengthy development of universal systems.

There is still substantial uncertainty but history teaches us that *simplicity* is often a winning principle. Grids and e-infrastructure should examine its implications for both low level protocol and the high level user and user data interfaces. *Too much computing* and *Too much data* have nontrivial implications for computer and networking hardware as well associated software and standards. Perhaps five years ago, we settled on software and decided on needed standards too quickly; we should expect change and mix research with robust deployment this time around.

4. Reappraising Grids

We will use the term “Grid” here to mean both deployed infrastructure (such as TeraGrid and OSG) and the binding middleware (Globus, Condor, SRB, etc).

Arguably, Grid middleware (the TeraGrid CTSS and OSG VDT) represent the successful culmination of more than a decade’s worth of investment in Grid technologies. Although there is room for improvement (both GRAM and WS-GRAM have scaling limitations and information services vary widely between TeraGrid, OSG, and EGEE, for example), Grid software is stable, well-packaged, relatively easy to install, and widely deployed on production systems. The infrastructure problems of data and computing time allocations, account management, and wide-area single sign-on have all been addressed. Interesting new services (such as the batch queue wait-time prediction service, QBETS) add value to Grids. Workflow tools (Kepler, Taverna, XBaya) all provide visual interfaces for integrating Grid and Web Services into composite tasks. There is still work to do to integrate all of these capabilities into tools that are useful for the end user.

From a different perspective, however, one can also argue that the major trends of Grid research and development have reached dead-ends, or at least rapidly diminishing returns.

Converting Users into Developers: A key problem with Grid middleware and Web services is the complication. It is very difficult for computational scientists to use, extend, and customize Grid software because the available tools and programming libraries are too complicated. Consequently, Grid development is largely in the hands of computer scientists and systems experts rather than domain scientists. The poor scientist is too often treated as a “customer” of the Grid development team. He or she may be the source of requirements and use cases, and may be drafted as a beta tester for the wonderful Grid infrastructure but will not be involved in the day-to-day development. This is an unnecessary waste of human resources.

This state of affairs is not by accident: most Grid developers (including the authors) have in the past advocated the adoption of “Enterprise” technologies (particularly Web Services and server-side portal technologies) for building Grids. As we have argued above, it is not clear at all that Enterprise technologies are developing along the most promising paths.

One of the many attractive aspects of Web 2.0 is the simplicity of its programming interfaces: anyone with general programming expertise can build a mashup. There really needs to be an equivalent mind set in the Grid development world.

To do this, organizations like the Open Grid Forum should change their focus. The problem (in our view) is that the OGF has concentrated too much on infrastructure *deployment use cases* (see OGSA Use Case documents for example) and not on *developer use cases.*

Virtual Organizations or Social Networks: Grids have been primarily focused on PKI-derived virtual organizations that are used to connect real organizations with existing bureaucracy, managements, etc. From the user's point of view, this mainly is useful for unifying accounts and providing single sign on. While useful and perhaps time-saving, this is not extremely compelling.

Going forward, Grids should instead take a lesson from Web 2.0 social networks. Unlike Grid VO's, these are user driven and focus on users sharing information, data, and services.

Multiscale Grids: Within 5 years, it will be possible to buy servers and workstations with dozens of cores. Parallel computing, displaced by Grid (wide area distributed) computing for the last decade, is enjoying a renaissance. On the other hand, scaling algorithms for most scientific domains are much further away. Thus, we can argue that the "sweet spot" for many parallel computing jobs in the next decade will be about the same size as a single multicore server. Only very mature parallel computing fields will be able to utilize petascale computing.

Thus we foresee the need for Grid software itself to be much more scalable, integrating desktops and servers and individual cores. Most of today's Grid services do not really add value until one attempts to integrate entire computing centers.

Application Services, not Infrastructure Service: Grid services provide job management, treating the underlying science code as an abstract job (typically in a queuing system) needing to be managed. This should be inverted. The goal should be making computational science services with simple, self-contained service APIs that can be programmed to. These application services can then be bound to underlying Grid services as necessary.

Cyberinfrastructure for Data Centric eScience and the Social Sciences and Humanities: The current cyberinfrastructure deployments are largely aimed at traditional computational users (i.e. scientists with codes to run on parallel machines and data to store on UNIX-like data archives) for obvious reasons. Unfortunately, this does not meet the needs of social sciences, humanities, and other academic research fields. These groups have some computing requirements (image processing, data mining), but they are more data-centric. They furthermore are probably in greater need of a simple to use, national scale IT infrastructure.