Grids Challenged by a Web 2.0 and Multicore Sandwich

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

Abstract
We discuss the application of Web 2.0 to support scientific research (e-Science) and related "e-moreorlessanything" applications. Web 2.0 offers interesting technical approaches to build the core e-infrastructure (Cyberinfrastructure) as well as a host of interesting services exemplified by Facebook, YouTube, Amazon S3/EC2 and Google maps. We discuss why some of the original Grid goals of linking the world's computer systems may not be so relevant today and that interoperability is needed at the data and not always at the infrastructure level. Web 2.0 may also support Parallel Programming 2.0 -- a better parallel computing software environment motivated by the need to run commodity applications on multicore chips. A "Grid on the chip" will be a common use of future chips with tens or hundreds of cores.

1. Introduction

Grid computing has dominated distributed computing research for more than a decade (for reviews, see [Foster2004] and [Berman2003]). Architecturally, Grids (as normally defined by the community, but see below) have been closely aligned with Web Services [Atkinson2003], and the international research communities have worked to define open XML standards [OGF]. The application of Grids has traditionally centered on integrating very high-end resources (supercomputers, extremely large clusters, and data archives) that are run be various real agencies into virtual organizations. The NSF TeraGrid [TG] and NSF/DOE Open Science Grid [OSG] are prominent examples in the United States.

As we discuss in this paper, we foresee two important pressures on or drivers for the future of Grid computing. First, architecturally, the Web Service foundations of Grids are being challenged by so-called "Web 2.0" network computing approaches. As we have discussed in previous papers [Fox2007A, Fox2007B, Liu2007, Mustacoglu2007, Pierce2007A, Qiu2007A, Topcu2007], Web 2.0 (although unlike Web Services a largely uncoordinated activity) provides a comprehensive set of Web computing capabilities that mirrors the Web Service architecture [Hey2007, HinchcliffeBlog, Parastatidis2007]. Second, we see major challenges to Grids as their traditional deployment (providing the middleware for integrating major computing research centers) will be undermined by problem of an abundance (rather than a scarcity) of computing power for many problems. Arguably this state has existed for some time (hence the success of Condor [Thain2005] and related technologies for cycle scavenging), but we see this as being revolutionized by the coming ubiquity of parallel computing which include as a special case loosely

coupled Grid applications. The availability of substantial parallel computing power through dozens of multicore processors available on a single machine will allow current small and medium sized parallel computing jobs to run on a single machine, making the traditional supercomputing centers' infrastructure (user time allocations, multi-user batch queuing systems) a relatively unattractive, complex solution for all but the largest of parallel computing problems. Restructuring current Supercomputer infrastructure as highly parallel multicores in a "cloud" systems architecture may provide a new direction that links Web 2.0 and Grids and satisfy the "common" case of multiple smallish jobs. Power uses may be served with a "Compute Grid (e.g. Globus [Foster2006]) Cloud" with an architecture similar to current supercomputer infrastructure.

Before addressing these driving issues, we begin with a survey of terms and concepts that will be discussed in this paper. These will clarify our internal usage and we hope will also be adopted by others.

Narrow and Broad Grids: This field is confused by inconsistent use of terminology, and it is important to distinguish between *applications*, *infrastructure* and *technologies*, and their different realizations. We define Web Services, Grids and (aspects of) Web 2.0 and its variants like Enterprise 2.0 as *technologies*. Sometimes the term *Grids* is reserved for specific architectures like OGSA or a distributed system built from Web Services. There are also Grid systems like Globus, EGEE [EGEE] and TeraGrid with a particular application, technology or geographic focus. We call all these *Narrow Grids*, but one can also use the term *Grid* to describe any (large-scale) distributed system that is coordinated or managed for some goal. Such a *Broad Grid* concept would for example encompass Globus, general Web Service and Web 2.0 systems. These technologies combine and compete to build electronic (software) infrastructures that are termed e-infrastructure or Cyberinfrastructure. Such electronic infrastructure enables or hosts applications that we can term generically e-moreorlessanything. *e-Science* or perhaps better *e-Research* is of course a special case of e-moreorlessanything where it is science or scholarly research that is being electronically supported [DeRoure2007, Goble2007].

e-moreorlessanything: The originator of e-Science, John Taylor, Director General of Research Councils UK, Office of Science and Technology, provides the following definition: *e-Science is about global collaboration in key areas of science, and the next generation of infrastructure that will enable it*. e-Science involves developing tools and technologies that allow scientists to do 'faster, better or different' research. There are many other specific examples of e-moreorlessanything. For example, e-Business captures an emerging view of corporations as dynamic virtual organizations linking employees, customers and stakeholders across the world. Outsourcing is one aspect of such global corporate enterprise and so another example of e-moreorlessanything. In general these areas have a deluge of data of ever increasing size driven by new instruments, sensors and internet resources. This data must be managed and understood with sophisticated tools. Further people, computers, data (including sensors and instruments) must be linked for both on-demand and asynchronous activities. This distributed system forms a *virtual organization* (i.e. an electronically supported distributed but real organization) supported by a mix of Web 2.0 and Grid tools [Fox2007D].

Cyberinfrastructure and e-Infrastructure: Cyberinfrastructure is a largely USA term for the infrastructure that supports the data, people, and computers of distributed science (i.e. e-Science defined above) – by exploiting Internet technology (Web2.0) adding (via Grid technology) management, security, supercomputers etc [Bement2007, NSF2003].

NSF's Cyberinfrastructure has two rather different foci. It has both parallel and distributed computing systems that are both 'just" collections of networked computers and storage. Of course parallel systems have low latency (microseconds) between nodes and distributed systems higher latency (many milliseconds) between nodes. The parallel components are used to get high performance on individual large simulations with problems that need to be decomposed. These simulations could involve data analysis or data assimilation with data naturally distributed and supported by the establishment of the Cyberinfrastructure. In general the distributed aspect of Cyberinfrastructure integrates already distinct components which currently may or may not be parallel systems. As multicore becomes pervasive, all components of Cyberinfrastructure will become parallel if not "massively parallel".

Services: Cyberinfrastructure is made of distributed, ideally autonomous services (originally Web services) that are "just" programs or data sources packaged for distributed access. The data is expressed by XML-based standards like GML, CML and CellML/SBML (for Geography, Chemistry, and Biology respectively), while a service plays the role of methods in traditional programming. Web services use WSDL to define interfaces to the method functionality. In contrast, Web 2.0 follows the old programming library practice: one just specifies the interface without special interface definition language standards. However all approaches to services use a loose coupling of coarse grained entities where the interface establishes a "contract" independent of implementation between two services or a service and a client. Note that software engineering and interoperability/standards are closely linked to the use of services. Although there is no broad agreement on the "right" approach to services, all major approaches to distributed systems today are built around some form of services. These services are composed (linked together) by mashups (typically scripts) or workflow (often based on XML specifications like BPEL), which represent "Grid" or "Service" programming. Since we are discussing distributed systems, note that the composed services are actually aggregations of clients: the clients run in a single environment (i.e. JavaScript in a browser or a Java Virtual Machine in a portal server), while the services run remotely and are unaware of each other.

For e-Science, services fall into several categories including models, applications, and simulations; data access, storage, federation, and discovery; filters for data mining and manipulation; and finally general capabilities such as collaboration, security etc.

2. Web 2.0

There is no precise definition of Web 2.0, but it is operationally defined by a set of technologies (JSON, AJAX, etc. discussed later) and a wide range of Web sites supporting user interaction among themselves (social networking) for sharing 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. Technical capabilities include Start Pages (portals) for access to and mashups for integration of web information. There are also very popular capabilities like Blogs and Wikis for supporting communication either broadly or within an organization. Google maps and related technologies illustrate the power of interactive, integrative, contributory technologies, and are emblematic of Web 2.0, revolutionizing the extremely complicated world of Geographical Information Systems with much simpler XML standards and programming APIs that democratize the development process.

With consequences perhaps analogous to its upheaval of Geographical Information Systems, Web 2.0 has also encroached on more traditional territory of cyberinfrastructure. Recent important Web 2.0 developments include cloud systems, which support the distributed storage and computing that was up to now the distinctive feature of Grids. These clouds address "commodity usage" rather than the high performance simulations and data transport that are characteristic of Grids like TeraGrid (USA) and DEISA (Europe). From the developer's point of view, these systems provide much simpler programming, resource allocation, and security models than Grid computing. This can be traced to motivating problems: "narrow" Grids have a strong research flavor and have often attempted to support relatively complicated use cases [Foster2004B], whereas "cloud" systems are driven by economic considerations and so must appeal to the most popular (and simple) use cases. Some of this complication is a result of conflating the complicated needs of Grid deployers with the simpler needs of Grid developers. Computing clouds are not necessarily simpler to deploy, but they do present a simpler face to developers.

Web 2.0 can benefit e-Science in many ways. Its tools can enhance scientific collaboration, i.e. effectively support virtual organizations, in different ways from Grids, which focus on secure robust managed sharing of high value resources. The popularity of Web 2.0 can provide high quality technologies and software that (due to large commercial investment) can be very useful in e-Science and higher quality than Grid or Web Service solutions. Furthermore, the usability and participatory nature of Web 2.0 can bring science and its informatics to a broader audience. As we mention later, Web 2.0 can even help the emerging challenge of using multicore chips, i.e. in improving parallel computing programming and runtime environments.

We will now make the comparison of Web 2.0 and Grids more concrete. In Tables 1 and 2, we summarize and compare the Grid and Web 2.0 approaches to three major e-Science features.

Table 1: Grid View of e-Science Features	
Feature	**Grid Approach**

1: Community Building	Designed to enable Virtual Organizations based on collaborations between existing organizations such as research groups and supercomputing centers. Top-down approach, closely tied to PKI-based security infrastructure.
2: Collaboration	Focused on real time audio/video collaborations such as Access Grid. Virtual Organizations provide a framework but typically no interesting functions for asynchronous collaboration.
3: Semantic and ontological representation of metadata	Semantic Grid efforts follow closely the Semantic Web and use RDF, OWL for information representation. These can be used for both describing metadata and the contents of digital libraries as well as workflows.

Table 2: Web 2.0 View of e-Science Features	
Feature	**Web 2.0 Approach**
1: Community Building	Web 2.0 communities are typically networks of emergent groups of individuals with shared interests. Facebook, MySpaces, and Flickr are prominent examples.
2: Collaboration	Dominated by asynchronous collaboration: group-edited content (Wikis), shared commenting /rating/tagging of online content. Collaboration and community building are intertwined.
3: Semantic and ontological representation of metadata	Metadata described by Microformats (semantic XHTML extensions) that represent community consensus and convention. Ontologies are replaced by "folksonomies" of conventional tags used to describe a network entity.

Web 2.0 and Web Services: Originally we expected that Web Services would dominate a new generation of Enterprise software and that Grids would leverage commercial investment in this field and be built in terms of Web Services [Atkinson2005]. However this is not what happened. The rise of Web 2.0 shows that commercial software innovation is happening in a different space – that of consumer and media systems – where Web services have not had significant adoption.

Enterprise software and Web Services have evolved in expected directions but slowly. There is good .NET and Java support for Web services and the so-called WS-* specifications provide a rich, sophisticated but complicated standard set of capabilities for security, fault tolerance, meta-data, discovery, notification etc. We defined above a class of "Narrow Grids" build on Web Services, which provide a robust managed environment with growing but still small adoption in Enterprise systems and distributed science (e-Science).

It once appeared that use of Web Services in Grids was inevitable but this is no longer clear. Experience has shown that Web services are often complicated, slow and of lower functionality than traditional approaches. For example, WS-Security is quite slow while WS-RM (Reliable Messaging) seems to have poor adoption and is for example inadequate for multi-cast operations. Standards like WSDM (distributed management)

seem unnecessarily complex, which hampers its broad adoption, as does the difficult deployment (too much Java!) of Web Service infrastructure.

Web 2.0 supports a similar architecture to Web services despite being developed in a more chaotic but remarkably successful fashion with a service architecture using a variety of protocols including those of Web and Grid services. For example there are over 500 Interfaces defined at [PW.com]. These interfaces and data formats (such as KML for Google maps) are often proprietary or de facto standards. However the communicatory nature of Web 2.0 makes interface information readily available while the arcane UDDI Web Service registry approach has failed. One can easily combine SOAP (Web Service) based services/systems with Web 2.0 services and simple REST or XML over HTTP messages. However in such a hybrid world, the systems will naturally evolve to the "lowest common denominator" and the additional structure and complexity of SOAP messaging and WS-* specifications will not thrive.

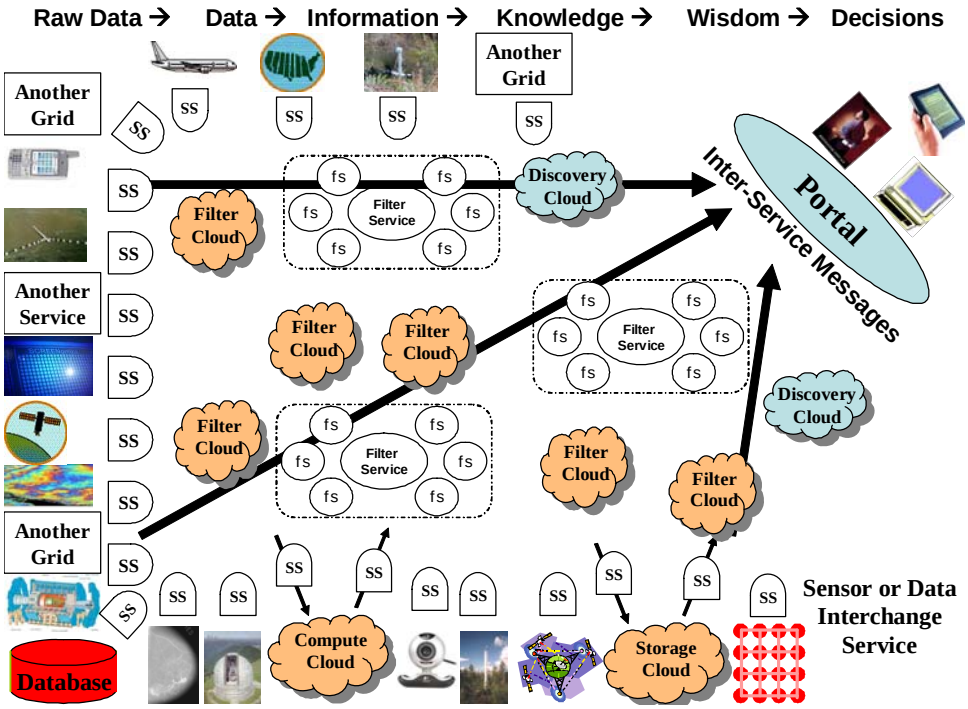

*Figure 1: Information architecture combining Web 2.0 and Grid Concepts. Wisdom is obtained by fusing and transforming data that comes from sensors, instruments, services, Grids and Clouds. Data is transformed by filters that perform data analysis, transformation, assimilation or production from simulations. 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*

Service Oriented Information Architecture

There is agreement on a general service architecture for information infrastructure – one creates a Cyberinfrastructure consisting of distributed services accessed by portals,

gadgets, gateways, and/or RSS feeds. This is illustrated in Fig. 1 which includes a rich set of sensor services that wrap all sources of data which can be simulations, video cams, robots, instruments such as LHC or even just any other service, Grid or Cloud. This data is then processed by a workflow that fuses and transforms the data into more refined forms enabling decisions. There is a traditional DIKW (Data, Information. Knowledge and Wisdom) pipeline. In the figure a service that accepts raw data and produces data is architecturally no different from one that accepts knowledge and produces wisdom. The filters shown in figure could involve geometric corrections, simulations or sophisticated data-mining algorithms but each most importantly defined by the input and output data which are of course transported as messages between services. We indicate in figure how current Grid architectures identify individual services while the Cloud architectures identifies collections of services (clouds) whose internal service infrastructure is opaque. In either case, we see a classic system of systems or rather Grid of Grids hierarchical composite architecture. Note that in this approach, sensor services and filter services are only distinguished in their input; filter services and clouds get messages as input from other services or clouds. Sensor services have some "out-of-band" source of data but both sensors and filters have similar outputs – namely "e-moreorlessanything" formatted data streams. In this approach an RSS feed from Flickr for example is a sensor as it produces messages every now and then as a time series.

Narrow Grids and Web Services A striking feature of traditional Web services is the rich WS-* specifications defining in detail the overall system infrastructure. Although a beautiful idea, it has proven hard to implement well and realize the benefits of the infrastructure level interoperability. In contrast, Web 2.0 focuses on a few simple system principles with interoperability as discussed for Figure 1 only at the application data level. Note this data focus is consistent with Semantic Grid/Web but so far the sophisticated capabilities (built around say RDF and OWL) of the Semantic Web have had modest use. A cynic might note that the lack of detailed standards in Web 2.0 come about as it is preferable to industry which can get proprietary advantage inside their clouds. Returning to an earlier discussion, one needs to share computing, data, people in e-moreorlessanything, Grids initially focused on computing but data and people (the Web 2.0 focus) are currently more important.

A Web 2.0 fanatic might argue that Web Services and Narrow Grids are taking too long to solve the wrong problem at the wrong point in stack with a complexity that makes friendly usability difficult. Note that in spite of the unclear technology directions, e-Science and more generally e-moreorlessanything are thriving with the advantages of distributed enablement very clear in many fields.

3. Multicore and Too much Computing

Traditionally both grids and parallel computing have tried to increase computing capabilities by aggregating computers together in a distributed or local fashion respectively. This approach has naturally optimized the performance of codes but often this is at the cost of re-usability. One also sees great interest in exploiting all possible CPU's such as graphics co-processors on a motherboard while in a distributed system scavenging software makes use of any "idle cycles" on computers

often across administrative domains. Again approaches like TeraGrid for NSF in the USA and those in other communities have linked many large computers together. Science Gateways (portals) show the positives of such Grids as they allow more seamless choice of use of networked resources; such brokering is in the spirit of Web 2.0. However the original meta-computing goal of Grids, i.e. the integration of capabilities of multiple resources, creates a lot of the complexity of today's systems that cross administrative domains. This should be contrasted with the cloud system approach which give the illusion of a "black-box cluster" i.e. a single uniform system. We have emphasized already that data is naturally distributed and simple data format interoperability standards support exchange of information between different clouds and between clouds and client systems. Traditional "Compute/File" Grids do increase the available computing but is this really needed?

Arguably the next crisis in technology area will be the opposite problem to that tackled by Grids; namely, *too much computing* will be available. Mass market CPU chips will be 32-128way parallel in 5 years time, and 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 this issue with approaches that can be implemented in next 3-5 years. Multicore servers have one source of "natural parallelism" as many users can access and use machines simultaneously on separate cores but there is no such obvious universal parallelism on clients.

One of most interesting analyses of possible applications that can exploit multicore comes from Intel with its RMS (Recognition Mining Synthesis) analysis [INTELRMS]. They identify gaming and generalized decision support (data mining) as possible multicore applications. Perhaps it will be *too much data* and its data mining that will come to the rescue of *too much computing*? There will be an increasing data deluge including the scientific observations for e-Science but these are most naturally addressed by parallel data mining on servers. However the deluge is pervasive and clients could host data from local (video, 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 relevant 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" both for server-side science and client-side PC's.

3.1. Attack of the Killer Multicores

Today commodity Intel systems are sold with 8 cores spread over two processors. Specialized chips such as GPU's and IBM Cell processor have substantially more cores [Dongarra2007]. Moore's Law implies and will be satisfied by and imply exponentially increasing number of cores doubling every 1.5-3 Years. However, we can expect only

modest increases in clock speed in individual processors. Moore's Law will be upheld by the ever increasing number of cores on a single processor. Intel has already prototyped an 80 core server chip that is a foretaste of systems that could be on the market as early as 2011.

In order to use these cores, most hardware and software vendors have started significant activities in parallel computing programming (at least partially recycled from the past [Fox2007C].) Some of the programming models and application styles are similar to Grids (Web 2.0) programming styles (as we will discuss in Section 4.2) [Patterson2007, SALSA]. Parallel computing will use where possible distributed system technology to benefit from the enormous software investment in the area. On the other hand, dozens of cores on a chip will motivate many to build a Broad Grid on a chip. This will encourage the developers of Broad Grid technology and applications to make them run very well internally to multicore systems.

3.2. Grids meet Multicore Systems

The expected rapid growth in the number of cores per chip has important implications for Grids. With 16-128 cores on a single commodity system five years from now, one will both be able to build a Grid like application on a chip and indeed must build such an application to get the Moore's law performance increase. Otherwise one will "waste" cores. Indeed, one of the challenges facing chip manufacturers is to identify parallel applications that can be used to provide a justification for the cores.

One will not want to reprogram when moving an application from a 64 node cluster or transcontinental implementation to a single chip Grid. However multicore chips have a very different architecture from Grids: shared rather than distributed memory. Similarly, latencies are measured in microseconds not milliseconds. As we have discussed in previous papers, millisecond latencies in messaging ("the rule of the millisecond") actually provide a fair description that can be used to distinguish Grid applications from parallel applications. Thus Grid and multicore technologies will need to "converge" and converged technology model will have different requirements from current Grid assumptions

3.3. Grid versus Multicore Applications: the Role of Data

It seems likely that future multicore applications will involve a loosely coupled mix of multiple modules that at least include the three application classes: Data access, query and store; Analysis, Filtering, Transformation and/or simulation; User visualization and interaction. This is precisely the mix that Grids support. Grids of course involve distributed modules implementing the different components, while multicore machines must integrate them in a single system. Grids and Web 2.0 use service-oriented architectures to describe system at module level and we will argue later that this is an appropriate model for multicore programming. Given the importance of data in all applications (the "too much data" hypothesis), we need to analyze carefully where the data for multicore applications will come from as all this computing is not useful if one spends all one's time migrating data around. One typically addresses this by placing compute (analysis) at the data but this is not so obvious if most of the computing power is

instantiated as multicore clients on the edge of the network (that is, the server under a user's desk). These multicore clients can get data from the Internet, i.e. distributed sources. As mentioned earlier, this data captures the personal interests of client and can be used by client to help user interact with world. Another possibility local source of data is that from a set of local sensors (video-cams and environmental sensors) naturally stored on client or locally to client. Alternatively the multicore client could perform a standalone calculation or be part of a distributed coordinated computation (SETI@Home)

Of course one may be able to afford the network use to copy or cache remote data on a client. In this regard, note that as you increase sophistication of data analysis, you increase the ratio of compute to input-output data transfer. Maybe as "too much data" and "too much computing" inexorably take off, algorithms will evolve and increase their complexity to allow easier matching of data and multicores. For example, a typical modern data-mining approach like Support Vector Machine is sophisticated (dense) matrix algebra and not just text matching [BYOPA2007]. The time complexity of sophisticated data analysis will make it more attractive to fetch data from the Internet and cache/store on client. The increasing algorithmic computational complexity will also help with memory bandwidth problems in multicore chips. In this vision, the current Grid "just" acts as a source of data and the Grid application runs locally.

4. A Comparison of Web 2.0 and Grid Technologies

In this section we review Web 2.0 technologies, which we have organized into categories matching our earlier classification of Web Service standards. These are presented in Tables 3 and 4.

Table 3: Ten Web Service Areas with Examples

WS-* Area	Grid/Web Service Examples
1: Core Service Model	XML, WSDL, SOAP
2: Service Internet	WS-Addressing, WS-MessageDelivery; Reliable Messaging WSRM; Efficient Messaging MOTM
3: Notification	WS-Notification, WS-Eventing (Publish-Subscribe)
4: Workflow and Transactions	BPEL, WS-Choreography, WS-Coordination
5: Security	WS-Security, WS-Trust, WS-Federation, SAML, WS-SecureConversation
6: Service Discovery	UDDI, WS-Discovery
7: System Metadata and State	WSRF, WS-MetadataExchange, WS-Context
8: Management	WSDM, WS-Management, WS-Transfer
9: Policy and Agreements	WS-Policy, WS-Agreement
10: Portals and User Interfaces	WSRP (Remote Portlets)

Table 4: Web 2.0 Approach to Web Service Capabilities

WS-* Area	Web 2.0 Approach

1: Core Service Model	XML becomes optional but still useful (especially if kept simple and small enough for simple parsing and manipulation in memory limited applications); SOAP becomes JSON, RSS, or ATOM; WSDL becomes REST with generic API (GET, PUT, etc); HTTP remains the primary transport mechanism.
2: Service Internet	No special Quality of Service. Assumes TCP/IP provides sufficient guarantees. Use JMS or equivalent although JMS not very aligned with Web 2.0. Most naturally use inside Clouds.
3: Notification	XmlHttpRequest plus HTTP with polling– JMS perhaps?
4: Workflow and Transactions (no Transactions in Web 2.0)	Workflows analogous to mashups, Google MapReduce. Scripting with PHP, JavaScript ….
5: Security	SSL, HTTP Authentication/Authorization; OpenID is Web 2.0 Single Sign on.
6: Service Discovery	Web sites such as http://www.programmableweb.com; no standard programmable discovery system, but examples based on Atom Publishing Protocol are possible.
7: System Metadata and State	Processed by application – no exposed system state (REST); Microformats are a universal metadata approach
8: Management (Interaction)	WS-Transfer-style protocols (GET, PUT, etc).
9: Policy and Agreements	Service dependent. Processed by application
10: Portals and User Interfaces	Start Pages, AJAX and Widgets (Netvibes), Gadgets

4.1. Web 2.0 and Grids

In this section, we compare Web 2.0 and Grid [Fox2007D] environments and especially portals and workflow engines. Given the level of adoption of Web 2.0 technologies relative to Grid technologies, we suggest the replacement of many Grid components with their Web 2.0 equivalents: Mashups could replace workflow; Start Pages with gadgets and widgets could replace portlets, while UDDI could be replaced by user generated registries (discovered by standard Web search engines), and so on. Microformats (that is, community-defined XHTML extensions to represent nuggets of metadata) can be used to describe Web 2.0 services, providing potentially a more automatic and sophisticated discovery mechanism. Alternatively, one may adapt the Atom Publishing Protocol as an information system. This REST-like API is designed to support the discovery of new Atom feeds using Atom itself as an information and metadata format. Atom is extensible, so one may embed other markups (such as microformats) into the Atom feed.

Mashups and Workflows: One controversial observation in our comparison above (Tables 3 and 4) is that mashups are at least operationally equivalent to Grid workflow systems. Both are best defined through examples. Mashup applications combine operations from two or more Web 2.0 services with the linkage either client or server side.

The relatively sophisticated Web 2.0 approach to clients encourages much that is server side in a classic Web Service approach to be replaced by client (often JavaScript) software. Google Maps, Yahoo Maps or Microsoft Virtual Earth supplemented with services providing custom data are the most well known mashups. The popularity of mashups has grown to the point that compositional tools are now available [Pipes, Popfly]. These are reviewed at [HinchcliffeBlog] while Web Service and Grid Workflow tools are reviewed at [Gannon2006, Yu2005]. As with mashups, workflow engines combine Web Services into composite applications. They are typically "scripted" with XML (BPEL), just as mashups are typically written using JavaScript or PHP. However both now offer visual interfaces which are familiar from systems like AVS which like Pipeline Pilot (Cheminformatics) and Khoros (Image Processing), we now recognize as early workflow and mashup engines.

Workflows and mashups themselves are excellent candidates for other Web 2.0 applications. Here we note the important myExperiment effort, which is building a Web 2.0 social network including shared Taverna workflows [myexperiment]. Yahoo Pipes offers a somewhat similar mechanism: pipes may be shared, extended, annotated, and rated by collaborators. One may also search for interesting workflows or see which ones are the most viewed.

Web 2.0 Mashups and APIs: One of the best sources for mashup examples and APIs is the Programmable Web [PW.com], which has (as of December 23, 2007) 2607 mashups and 579 Web 2.0 APIs. Google Maps is the most often used in mashups. Amazon S3 is also growing in popularity, being number 21 in the list of API's in use by Mashups. Note that the Programmable Web acts as a UDDI style registry for the Web 2.0. It is striking that in spite of the emphasis on open standards of the research community, there is no such site registering e-Science and Grid services. Each site has API and its features that are divided into broad categories. Only a few are used a lot with only 56 API's or around 10% of the total used in 10 or more mashups. APIs are also available through an RSS feed, providing a programmable way for interacting with the APIs. It seems unlikely that complex technologies like BPEL for specifying workflows will be competitive with the simpler scripting technologies used in Web 2.0 and some Grid workflows. XML is very powerful but it is not an elegant way to specify a programming language.

Portlets and Google Gadgets: Portals (also known as Science Gateways especially when used for the TeraGrid) for Grid Systems are commonly built using server-side Java technology and have a component-container model (portlets). Standard compliant portlet containers such as GridSphere aggregate portlets on the server-side into a single Web application. Portlet components are portable across standard compliant applications but must be deployed on the server with the container. The Web Services for Remote Portlets OASIS standard (WSRP) provides the potential of decoupling portlets from the Java Virtual Machine of the container, but this standard has been hampered by lack of well-implemented open source tools and probably also a compelling use case, at least for science gateways.

In contrast, Start Pages such as iGoogle offer user-customizable content driven by a large collection of community contributed components ("gadgets"). Gadgets are clients to Web 2.0 services such as RSS feeds and more complicated applications. iGoogle uses client-side instead of server-side aggregation. Gadgets can also be designed to be work with Google sidebar. Although more user friendly than most portlet containers and much richer in content, Start Pages do not support the level of security assumed in most portlet-based applications. The resolution of this is either to design an open source Start Page container that can support more sophisticated security requirements, or alternatively to develop more science gadgets that do not require sophisticated security models.

4.2. Parallel Programming 2.0

Web 2.0 can also help address long standing difficulties with parallel programming environments which we can illustrate with the multicore Service Aggregated Linked Sequential Activities (SALSA) project from Indiana University [Qiu2007A, SALSA]. Their aim is to link parallel and distributed (Grid) computing by developing parallel applications as services and not as programs or libraries. Given the expected "too much data and computing" scenario explained in Section 3, SALSA is developing a set of services (library) of multicore parallel data mining algorithms that can be composed as mashups.

We need to define a model for parallel programming [Patterson2007]. Experience from parallel programming (largely for scientific applications) suggests we break parallelism into two areas: firstly, building parallel "kernels" (libraries) and secondly, composing parallel library components into complete applications.

Scalable Parallel Components (Kernels): There are no agreed high-level programming environments for building library members that are broadly applicable. However, lower level approaches where parallelism must be defined explicitly are available and although quite hard to use, are reliable and well understood. Such models include MPI for messaging or just locks within a single shared memory. SALSA is currently using a very flexible messaging system, CCR from Microsoft. There are several low level messaging patterns to support here including the collective synchronization of MPI, dynamic irregular thread parallelism needed in search algorithms, and more specialized cases like discrete event simulation. We currently assume that the kernels of such scalable parallel libraries will be built by experts with a broader group of programmers composing library members into complete applications using approaches described below.

Composition of Parallel Components: The composition step has many excellent solutions as this does not have the same drastic synchronization and correctness constraints as in scalable parallelism above. Approaches to composition include task parallelism in languages such as C++, C#, Java and Fortran90; general scripting languages like Python; and domain specific environments like Matlab. Recent approaches include MapReduce, F# and DSS. Many scientific applications use MPI for the coarse grain as well as fine grain area parallelism. The new languages from DARPA's HPCS program support task parallelism (composition of parallel components), but we expect that decoupling composition and scalable parallelism will remain popular and must be

supported. Graphical interfaces were popularized with AVS and Khoros 10-15 years ago and recently are seen in Grid/Web Service workflow systems such as Taverna, InforSense KDE, Pipeline Pilot (from SciTegic), and XBaya (part of the LEAD environment built at Indiana University). As discussed in section 4.1, Mashups from Web 2.0 are also usable here and as this is the broadest area can be expected to develop the most user friendly software. It may need to be enhanced to provide needed security (Grids) and performance (multicore) but we assume that Web 2.0 mashup technology will be very attractive for composition of parallel kernels. We term this *Parallel Programming 2.0*.

Note both programming and runtime for kernels and their composition must be supported in three environments: inside chips (the multicore problem); between machines in clusters (the traditional parallel computing problem); or in Grids. The building of kernels is typically only interesting on true parallel computers as the algorithms require low communication latency. However composition is similar in both parallel and distributed scenarios and it seems useful as discussed above to allow the use of Grid and Web composition tools for the parallel problem. Thus we suggest that it is useful to capture parallel library members as (some variant of) services. Note that we are not assuming a uniform implementation and in fact expect good service composition inside a multicore chip to often require highly optimized communication mechanisms between the services that minimize memory bandwidth use. However very different mechanisms would be used to integrate services between computer systems. Further bandwidth and latency requirements reduce as one increases the grain size of services and this again suggests the smaller services inside closely coupled cores and machines will have stringent communication requirements. The above discussion defines the “Service Aggregation” term in SALSA; library members will be built as services that can be used by non expert programmers.

We generalize the well-known CSP (Communicating Sequential Processes) of Hoare to describe the low level approaches to kernel building as “Linked Sequential Activities” in SALSA. We use the term activities (and not processes) in SALSA to allow one to build services from either threads, processes (usual MPI choice) or even just other services. We choose linkage to denote the different ways of synchronizing the parallel activities that may involve shared memory rather than some form of messaging or communication.

There are several engineering and research issues glossed over above. We mentioned the critical communication optimization problem area already. We need to discuss what we mean by services; the requirements of multi-language support; supporting implementations on multicore, cluster or Grid infrastructure. Further it seems useful to re-examine MPI and define a simpler model that naturally supports threads or processes and the full set of communication patterns mentioned above.

5. Summary and Looking to the Future

Web 2.0 and Grids are addressing similar application classes although Web 2.0 has focused more on user interactions and less on computing (or to be precise, on running jobs). Thus the component technologies for Grids and Web 2.0 have comparable

capabilities and it should be fruitful to compare, contrast and as appropriate combine ideas and systems with portals, workflow and registries fruitful areas. The other side of the Grid Sandwich is multicore which has some similarities (both have lots of processing units) but very different issues in area of performance and as discussed in section 3.3, the location of data. We noted that both for Grids and multicore systems, mining the data deluge is expected to be a critical application. Although multicore requires low latency run time (microseconds) for the parallel runtime of the kernels, the requirements for "task parallelism" (module composition) are less stringent and we proposed "Parallel Programming 2.0" where multicore kernels are built as services and composed using some variant of mashup or workflow technology.
The contrast between services and the more traditional object approaches for parallel programming deserves more study. Note that a two level programming model is common for Grids with services being constructed from one language (for example Java or C#) but composed with another (MapReduce, BPEL or Popfly). However a single integrated language (Java, C++, HPCS) is probably most popular in parallel computing. In Parallel Programming 2.0, we assume traditional (possibly object oriented) languages will be used to build kernels as services while in the two level model we use a different mashup or workflow technology to compose the kernels. Web 2.0 has highlighted the value of simplicity in protocols; one might speculate that standards like MPI whose functionality is certainly needed on multicore chips [Qiu2007A], could be usefully simplified in a broadly adopted Parallel Programming 2.0. System of Systems, Grids, Web 2.0 and Multicore are likely to build systems hierarchically out of smaller systems, so we need to support Grids of Grids, Webs of Grids, Grids of Multicores etc. i.e. systems of systems of all sorts

Looking to the future, Web 2.0 has momentum as it is driven by success of social web sites and the user friendly protocols attracting many developers of mashups. For narrow Grids, their momentum is driven by the success of eScience and the commercial web service thrusts largely aimed at Enterprise. We expect application domains such as business and military, where predictability and robustness are often essential, might be built on Web Service (Narrow Grid) technologies with the user interactivity of Web 2.0 added to support social interactions in their virtual organizations. However, the higher complexity of Web Services discourages both broad adoption and high implementation quality of WS-* components, requiring substantial investment. Maybe this will just wither away, leaving a simpler Web 2.0 technology base. On the other hand robustness and coping with unstructured blooming of a ten thousand flowers are forces pressuring Web 2.0 and confusing its future role. The usability and full exploitation of Multicore systems will drive the development of Parallel Programming 2.0, and we expect this to see much innovation. Perhaps the most interesting near term questions for distributed system Grids and Web 2.0 are the Grid Cloud architecture, data interchange standards and usage models.

References

1. [Atkinson2005] M. Atkinson et al., *Web Service Grids: An evolutionary approach*, Concurrency and Computation: Practice and Experience 17, 377-389, 2005; http://www.nesc.ac.uk/technical_papers/UKeS-2004-05.pdf

2. [Bement2007] Arden L. Bement, *Cyberinfrastructure: the Second Revolution*, the Chronicle of Higher Education Volume 53, Issue 18, Page B5 January 5, 2007
3. [Berman2003] *Grid Computing: Making the Global Infrastructure a Reality* edited by Fran Berman, Geoffrey Fox and Tony Hey, John Wiley & Sons, Chichester, England, ISBN 0-470-85319-0, March 2003. http://www.grid2002.org.
4. [BYOPA2007] Geoffrey Fox, *Parallel Computing 2007: Bring your own parallel application* Presentation in [Fox2007C] http://grids.ucs.indiana.edu/ptliupages/presentations/PC2007/PC07BYOPA.ppt
5. [DeRoure2007] Dave DeRoure, *The New e-Science*, keynote talk at eScience 2007, the 3rd IEEE International Conference http://www.escience2007.org/index.asp on e-Science and Grid Computing Bangalore India December 13 2007
6. [Dongarra2007] Jack Dongarra Editor *The Promise and Perils of the Coming Multicore Revolution and Its Impact*, CTWatch Quarterly Vol 3 No. 1 February 07, http://www.ctwatch.org/quarterly/archives/february-2007.
7. [EGEE] Enabling Grids for E-Science (EGEE): http://www.eu-egee.org/.
8. [Foster2004] *The Grid 2: Blueprint for a new Computing Infrastructure*, edited by Ian Foster and Carl Kesselman, Morgan Kaufmann 2004.
9. [Foster2004B] I. Foster, D. Gannon, H. Kishimoto, and J. J. Von Reich, "Open Grid Serivces Architecture User Cases." Global Grid Forum Informational Document GFD-I.029, 2004.
10. [Foster2006] Ian T. Foster: Globus Toolkit Version 4: Software for Service-Oriented Systems. J. Comput. Sci. Technol. 21(4): 513-520 (2006).
11. [Fox2007A] Geoffrey C. Fox, Rajarshi Guha, Donald F. McMullen, Ahmet Fatih Mustacoglu, Marlon E. Pierce, Ahmet E. Topcu, and David J. Wild *Web 2.0 for Grids and e-Science* INGRID 2007 - Instrumenting the Grid 2nd International Workshop on Distributed Cooperative Laboratories - S.Margherita Ligure Portofino, ITALY, April 18 2007 http://grids.ucs.indiana.edu/ptliupages/publications/INGRIDFinal.pdf
12. [Fox2007B] Geoffrey C. Fox, Marlon E. Pierce, Ahmet Fatih Mustacoglu, and Ahmet E. Topcu *Web 2.0 for E-Science Environments* Keynote Presentation at 3rd International Conference on Semantics, Knowledge and Grid SKG2007 Xian China October 28-30 2007 http://grids.ucs.indiana.edu/ptliupages/publications/PID470571.pdf
13. [Fox2007C] Geoffrey Fox tutorial at Microsoft Research *Parallel Computing 2007: Lessons for a Multicore Future from the Past* February 26 to March 1 2007 http://grids.ucs.indiana.edu/ptliupages/presentations/PC2007/index.html
14. [Fox2007D] Geoffrey C. Fox and Marlon E. Pierce *Web 2.0 for eScience: SC07 Education Program Tutorial* Education Program Tutorial at SC07 November 12 2007 Reno Nevada http://grids.ucs.indiana.edu/ptliupages/presentations/sc07tutorial/Web20Tutorial_SC07.ppt
15. [Gannon2006] Dennis Gannon and Geoffrey Fox, *Workflow in Grid Systems* Concurrency and Computation: Practice & Experience 18 (10), 1009-19 (Aug 2006), Editorial of special issue prepared from GGF10 Berlin http://grids.ucs.indiana.edu/ptliupages/publications/Workflow-overview.pdf

16. [Goble2007] Carole Goble, David De Roure *Grid 3.0: Services, Semantics and Society* Grid 2007 Conference, Omni Austin Downtown Hotel Austin Texas, September 20 2007 http://www.semanticgrid.org/presentations/Grid2007-GOBLE2.ppt
17. [Hey2007] Tony Hey *eScience and Digital Scholarship* The 2007 Microsoft eScience Workshop at RENCI Friday Center Chapel Hill, NC 27599-1020 October 21-23 2007 https://www.mses07.net/main.aspx
18. [HinchcliffeBlog] Enterprise Web 2.0 http://blogs.zdnet.com/Hinchcliffe
19. [INTELRMS] Pradeep Dubey from Intel, *Teraflops for the Masses: Killer Apps of Tomorrow*, Workshop on Edge Computing Using New Commodity Architectures, Chapel Hill, North Carolina 22-24 May 2006. http://gamma.cs.unc.edu/EDGE/SLIDES/dubey.pdf
20. [Liu2007] Zao Liu, Marlon E. Pierce, and Geoffrey C. Fox *Implementing a Caching and Tiling Map Server: a Web 2.0 Case Study* Proceedings of The 2007 International Symposium on Collaborative Technologies and Systems (CTS 2007) http://grids.ucs.indiana.edu/ptliupages/publications/CachingTilingMapServer.pdf
21. [Mustacoglu2007] Ahmet Fatih Mustacoglu, Ahmet E. Topcu Aurel Cami, Geoffrey Fox *A Novel Event-Based Consistency Model for Supporting Collaborative Cyberinfrastructure Based Scientific Research* Proceedings of The 2007 International Symposium on Collaborative Technologies and Systems (CTS 2007) http://grids.ucs.indiana.edu/ptliupages/publications/CTS2007_camera_ready.pdf
22. [myexperiment] myexperiment Home Page http://www.myexperiment.org/
23. [NSF2003] Report of the National Science Foundation Blue-Ribbon Advisory Panel led by Dan Atkins, *Revolutionizing Science and Engineering Through Cyberinfrastructure*, http://www.nsf.gov/publications/pub_summ.jsp?ods_key=cise051203
24. [OGF] Open Grid Forum http://www.ogf.org
25. [Patterson2007] David Patterson *The Landscape of Parallel Computing Research: A View from Berkeley 2.0* Presentation at Manycore Computing 2007 Seattle June 20 2007 http://science.officeisp.net/ManycoreComputingWorkshop07/Presentations/David%20Patterson.pdf
26. [Parastatidis2007] Savas Parastatidis *Web 2.0 Cloud Era and its impact on how we do research* OGF21 Web 2.0 Workshop Seattle October 15 2007. http://www.ogf.org/OGF21/materials/1031/2007.10.15 - OGF - Web 2.0-Cloud Era and its Impact on how we do Research.pdf
27. [Pierce2007A] Marlon E. Pierce, Geoffrey Fox, Huapeng Yuan, and Yu Deng Cyberinfrastructure and Web 2.0 Proceedings of HPC2006 July 4 2006 Cetraro Italy
28. [Pipes] Yahoo Pipes Mashup Tool Home Page http://pipes.yahoo.com/pipes/
29. [Popfly] Microsoft Popfly Mashup Tool Home Page http://www.popfly.ms/
30. [PW.com] Programmable Web Site http://www.programmableweb.com with Web 2.0 API's listed at http://www.programmableweb.com/apis
31. [Qiu2007A] Xiaohong Qiu, Geoffrey Fox, H. Yuan, Seung-Hee Bae, George Chrysanthakopoulos, Henrik Frystyk Nielsen High Performance Multi-Paradigm

Messaging Runtime Integrating Grids and Multicore Systems September 23 2007 Proceedings of eScience 2007 Conference Bangalore India December 10-13 2007
32. [SALSA] Service Aggregated Linked Sequential Activities (SALSA) project http://www.infomall.org/salsa.
33. [Thain2005] Douglas Thain, Todd Tannenbaum, Miron Livny: Distributed computing in practice: the Condor experience. Concurrency - Practice and Experience 17(2-4): 323-356 (2005).
34. [TG] TeraGrid Web Site: http://www.teragrid.org/.
35. [OSG] The Open Science Grid Web Site: http://www.opensciencegrid.org/.
36. [Topcu2007] Ahmet E. Topcu , Ahmet Fatih Mustacoglu, Geoffrey Fox , Aurel Cami *Integration of Collaborative Information Systems in Web 2.0* 3rd International Conference on Semantics, Knowledge and Grid SKG2007 Xian China October 28-30 2007 http://grids.ucs.indiana.edu/ptliupages/publications/topcu_IntegrationWeb2.pdf
37. [Yu2005] Jia Yu and Rajkumar Buyya, *A Taxonomy of Workflow Management Systems for Grid Computing*, Technical Report, GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, March 10, 2005. http://www.gridbus.org/reports/GridWorkflowTaxonomy.pdf