Using Clouds to Provide Grids Higher-Levels of Abstraction and Explicit Support for Usage Modes

Shantenu Jha[*†], Andre Merzky[*], Geoffrey Fox[‡§]

[*] Center for Computation and Technology, Louisiana State University
[†] Department of Computer Science, Louisiana State University
[‡] Community Grids Lab, Indiana University
[§] Department of Computer Science, Indiana University

Abstract—**Grids in their current form of deployment and implementation have not been as succesful as hoped in engendering distributed applications. Amongst other reasons, the level of detail that needs to be controlled for the successful development and deployment of applications remains too high. We argue that there is a need for higher levels of abstractions for current Grids. By introducing the relevant terminology, we try to understand Grids and Clouds as systems; we find this leads to a natural role for the concept of *Affinity*, and argue that this is a missing element in current Grids. Providing these affinities and higher-level abstractions is consistent with the common concepts of Clouds. Thus this paper establishes how Clouds can be viewed as a logical and next higher-level abstraction from Grids.**

Index Terms—**Distributed Infrastructure, Distributed Computing, Grid, Cloud, Abstractions, Interfaces, Affinity**

I. Introduction

There is a level of agreement that computational Grids have not been able to deliver on the promise of better applications and usage scenarios. The lack of possible applications is not completely unrelated to the significant challenges in Grid deployment and management, the difficulty of providing interoperability, and of composing cross-Grid applications and services. Although the reasons are often context dependent and resist over-simplified explanations, if there is a single factor that stands out it is probably the complexity associated with Grids – both from a programmatic point of view as well as from a technology, management and deployment perspective.

Grids as currently designed and implemented are difficult to interoperate: there have been major attempts to create Grids that interoperate seamlessly, in particular by the 'Grid Interoperation Now (GIN)' effort within OGF [1]. Understandably, the various Grid programming and runtime environments vary significantly. But even if some level of homogenization could be imposed across different Grids, managing control programmatically across different virtual organization will remain difficult. Additionally, the lack of any commonly accepted minimal level of deployment support for cross-Grid applications (e.g. co-scheduling resource across more than one VO) makes aggregating cross-Grid resources difficult. Many of these difficulties underscore GIN's limited impact on applications in spite of the groups extensive and sincere efforts.

From our own experience as both end-users and developers of Grid infrastructure, there is a need to expose less detail and provide functionality in a simplifed way. If there is a lesson to be learned from Grids it is that the abstractions that Grids expose – to the end-user, to the deployers and to application developers – are inappropriate and they need to be higher level. As we will go on to show, Web-Services and their multiple incarnations have interfaces that are at a level that is too low, to enable the effective deployment of infrastructure and application development.

Clouds are clearly related to Grids, both in goals and implementation, but there are differences which are difficult to discuss as both terms do not have agreed definitions.

We believe that Clouds as systems are not orthogonal to Grids, nor are they necessarily complementary to Grids: in some ways Clouds are the evolution of Grids (and they can both in turn be viewed as evolution of Clusters). In many ways Clouds may just be composed of regular, vanilla Grids with suitable services and interfaces superimposed on them. Whether Clouds are a somewhat fuzzily defined concept or possibly a simple repackaging of earlier concepts from distributed systems, it is important to hash their relationship to existing classic Grids.

A fundamental difference between Clouds and Grids, in our opinion, is the support for interfaces that are syntactically simple, semantically restricted and high-level; standardised or not is an open question. In this paper we will introduce the notion of *Usage Mode* and *Affinities* of systems, which describe the dominant usage patterns of the system and the system's internal properties that support these patterns, respectively. We argue, that an emphasis on *Usage Modes* and *Affinities* is the putative cause for the simplicity of use of Clouds and this will be a major focus of this paper.

To the best of our knowledge this is the first systematic attempt to characterise Clouds in relation to Grids from the perspective of semantics and interface abstractions. The importance of this approach is reiterated by discussions on the next steps for existing Grid infrastructure projects such as the TeraGrid. For example, of the approximately 15 position papers that were solicited and submitted as part of the *"Future of the TeraGrid"* process, more than half mention the need for the next generation of the TeraGrid to take cognizance of the developments in Cloud computing – where Cloud computing is a catch-all term for better contextualization, virtualization and most importantly simplicity of use. Some such as Blatecky [2] advocate stronger positions, i.e. TeraGrid should focus on an exit strategy and give way to developments in virtualization such as Clouds and Web 2.0.

As a side note, the attention given to Clouds is partially due to the (coincidental but) simultaneous development in interests in green computing. Green computing may not be the most critical architectural design point [3], but if green practises arise naturally then that is an advantage. A natural way to construct a Cloud is *ab initio* and thus there is significant scope to utilize green locations and green energy sources. [1] It is conceivable that current social and political trends may lead to a situation where green computing considerations play an important role along side technical ones; we are not advocating green computing trends or technologies – for that matter we are not advocating Cloud computing either, but surely, the alignment of industrial trends and academic computing cannot be harmful for either. It is important to mention that for the purposes of this paper, we do not venture into the analysis of business models associated with Clouds; our focus is on Clouds as technology.

The remainder of this paper is structured as follows: In the next section we briefly outline and discuss the main recurring concepts in this paper, followed by specific examples of these concepts. We will then discuss Cloud *Affinity* as arising from the focus on interfaces and not on implementations. An analysis of the concepts involved leads to a strawman architecture for Clouds; we then analyze the implications of the proposed high-level architecture for Clouds and Grids, and close with some outstanding questions of relevance that we hope will be addressed by the community in the near future.

II. Definitions

This section attempts to list definitions for terms frequently encountered throughout the paper. These definitions are working definitions which are probably not universally applicable nor rigorous. Detailed definitions would have been impossible with the limited scope of this paper. We feel that these working definitions whilst simple and basic, are enough to facilitate discussions of the issues in hand.

Resource: A physical or virtual entity of limited availability. Physical resources are compute, storage and communication resources, etc. Virtual resources are usually services, which provide direct or indirect access to physical resources.

Service: An entity which provides a capability on a resource, or which allows actions to be performed on resources. Services can in turn be *Low Level Services* – which act primarily on physical resources, or *High Level Services* which act primarily on virtual resources (i.e. on other services). Services expose their capabilities via service interfaces.

System: A set of services and resources, which form an integrated whole. The concept of a system is inherently hierarchical, i.e. there are systems of systems. *Higher Level Systems* are systems which make use of other systems (i.e. *Lower Level Systems*), through aggregation.

[1]Grids on the other hand are mostly constructed from a set of existing resources.

Semantics (of Systems): The set of *capabilities*, or features, available within a system. The semantics of a system can be greater (more powerful) than the semantics of its individual (lower level) systems combined.

System Interface: A set of interfaces that allow an application (and other higher level systems) to access the capabilities of a system. *APIs* provide programmatic access to these interfaces. *Application Environments* provide user level abstractions to APIs and thus also access to service interfaces System interfaces often expose only parts of the entire semantics of the system.

Virtualization: An additional layer between real systems and applications which translates concurrent access to real systems into seemingly exclusive access to the virtual system. The virtualization interface often hides details and differences of the real system components.

Application: An entity making use of a system, e.g. by using an API, or an application environment (see below).

Portals and Science Gateways: High level application environments that are oriented towards facilitiating end usage; these access interfaces allows the description, instantiation, deployment and management of applications – both abstract and concrete – on a system. Application environments may provide additional, often application specific, semantics which is originally not available in the underlying system.

Usage Mode: A commonly occuring resource access and deployment pattern for an application or a class of applications. For example, *Usage Modes* maybe parameter sweeps, logical coupling of components (such as in workflows) etc.

Affinity: An inherent property of a system that describes a relationship between two or more (real or virtual) resources. The relationship is indicative of the types of *Usage Modes* that the system supports. Affinities can be indicative of support for data-oriented, compute intensive, or communication intensive computing, etc.

III. Examples

We will show through concrete examples that although, the above definitions are prima facie simple and limited, they permit a description of real world systems. In the following examples, we discuss the semantic properties of various entities, and provide a motivation for a later discussion of *affinities* and usage modes.

A. Resources

Resources can be classified as either compute, storage or communication, amongst other types. Some simple examples of (physical) compute resources are dedicated clusters and idle CPU cycles on workstations; single hard drives or large shared file systems are examples of storage resources; the Internet in various physical representations is an ubiquitous example of a communication resource. Numerous other types of resources exist, such as remote instruments, sensors, software licenses, human experts etc.

The semantics of a resource consist of a set of core capabilities specific to the resource and the ability to manage those

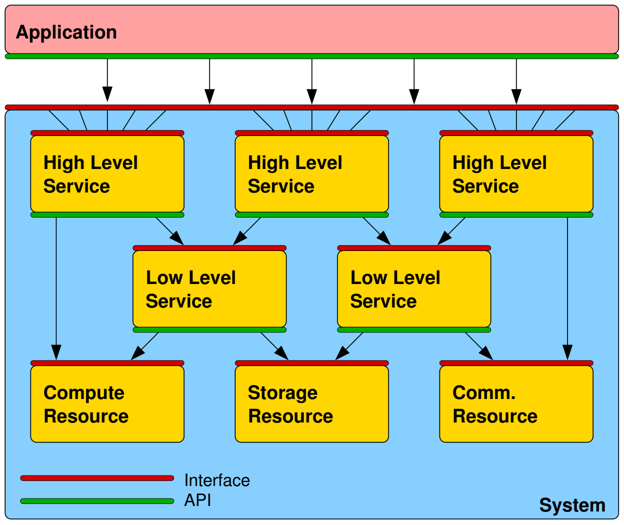

Fig. 1: Figure showing the relationship between the different concepts associated with a system. Systems are composed of services, which provide access to resources. Interfaces allow systems to be accessed and used; APIs in turn allow applications to access interfaces.

capabilities (provisioning, availability, QoS, security etc.) The core capabilities are usually variable – a CPU can run many kinds of applications; a disk can store many types of data; a network can connect to many types of endpoints etc. The means provided to exploit resource semantics, say via the resource interface, are thus also often flexible (assembly for CPUs; various file systems or driver interfaces for disks; TCP and other low level protocols for networks, etc.). The tradeoff is between the flexibility and complexity of these interfaces.

B. Services

Low Level Services: There are many low level services that allow actions on resources; OS level file systems, OS process schedulers etc. are low level services. Note that these services limit the exposed semantic capabilities of the resources (i.e. using a file system, a user will not be able to explicitly address individual blocks on the disk anymore: she must adhere to the notion of files and directories.)

In distributed environments, typical low level services are:

- job/batch schedulers: LSF, GRAM, Mauii, ...
- file systems: Google file system, AFS, GFS, ...
- communication: TCP streams, monitoring systems, ...

High Level Services: High level services often build upon multiple low level services and resources. For example, a replica service such as the Globus RLS would exploit storage resources (e.g., a global virtual file system), other storage resources (e.g., a database for meta data storage), and communication resources (e.g. a network for data movement). The semantic power is often greater than that of the individual pieces combined (a replica service may have the notion of replication policies, which make no sense on the level of the individual resources). On the other hand, the service interface will often limit the *exposed* semantics, according to the target use cases (e.g., the RLS API does not allow the creation of arbitrary tables in the metadata database).

In distributed environments, typical high level services are:

- job managers: meta-schedulers, supporting reservation and co-allocation, registries, ...
- file systems: replica systems, federated file systems, ...
- communication: message passing infrastructures, component systems, publish/subscriber systems, ...

C. Systems

An inherent generality in the definition of the term 'system' permits a wide variety of examples. We limit the discussion however, to examples which are of particular interest to this discussion, viz., Operating Systems, Grids, and Clouds.

1) Operating Systems: Broadly defined[4], an Operating System (OS) can be considered as the *software that manages the resources of a computer and provides programmers with an interface used to access those resources.'* The focus here is on a single computer. Although distributed operating systems exist, one can argue that the existence of the OS implies the existence of a single distributed computer. Interestingly, a system interface is an intrinsic part of the (operating) system; indeed, most systems would be useless without an interface to use that system (possibly apart from truly autonomous systems).

2) Grids: Grids are systems which, according to Ref [5], fulfill the following checkpoints:

- coordinate resources that are not subject to centralized control,
- use standard, open, general-purpose protocols and interfaces,
- deliver non-trivial qualities of service.

The TeraGrid[6], for example, is a system which provides more than 750 Teraflop of compute resources, more than 30 Petabyte of storage, and high performance network connections. Resources are administrated by individual TeraGrid sites. The infrastructure is based on open source software which implements (at least some) open standards. TeraGrid's 'native' system interface is increasingly complemented by a number of application oriented 'Science Gateways'. That is a typical and interesting development: the powerful but complex system interfaces are wrapped and abstracted by domain specific portals, which provide a limited, but simplier interface to the end-users.

The TeraGrid is a *General Purpose Grid*, as its interfaces provide access to a wide variety of capabilities, and does not limit the usage of the Grid resources for a specific application domain. In contrast to general purpose Grids, *Narrow Grids* (i.e. domain specific Grids) provide more focused services and interfaces. For example, the Cern Data Grid aimes to create a Grid with the ability to store and distribute large amounts of data, with less emphasis on high performance computing. A number of high level services have been created to provide that functionality *on top* of a general purpose Grid, effectively limiting its semantic capability, but increasing its ease-of-use for the target domain of distributed data management.

3) Clouds: Cloud systems (or just Clouds) are, in some respects, narrow Grids, with a limited set of features exposed, while still being able to serve a large fraction of the domain specific use cases (the Cloud's *Usage Mode*). For example, the Simple Storage Service by Amazon[7] (S3, details below) is a data Grid which, if compared to the Cern Data Grid, has less exposed semantics. The exposed feature set is, however, large enough to attract a significant user base and meet application requirements, which is also due to the simplicity of the exposed system interface.

Amazon's S3 and EC2: These are probably some of the best known examples of Clouds. S3 provides the abilty to *outsource* data – temporary store or archive with a given tightly defined quality of service guarantee on availability, durability, persistence of data. S3 has a simple cost model based upon usage measured in GB/month, with a certain cost for data transfer across S3 boundaries. Users are not charged for transfer if they use the "cooperative" EC2 compute Cloud – another service by Amazon, although they are charged for the compute time. S3 in principle provides infinite data storage, continous availabilty, and durability.

Similarly EC2 (Elastic Compute Cloud[8]) represents the ability to accomodate a very large number of compute jobs (if not in principle an infinte number), without the end-user realizing that it is a shared resource. EC2 is a nice example of an infrastructure's explicit support for different usage modes (bursty, high-throughput and parameter-sweep).

Ref [9] concluded, controversially at best, that S3 although a useful concept may not suitable for scientific projects and applications such as particle physics experiments, due to reasons primarily related to security, cost model and also performance. That claim is disputed though [10]. In any case, it is unclear if performance (or lack thereof) will be an issue in the uptake of these systems (S3 and EC2 specifically, but Clouds in general) for niche high-end applications; we argue that a "sweet spot" balancing the high-level interfaces and abstractions on the one hand, with the need for performance requirements on the other, is to be an important consideration.

Cloud vendors such as IBM are working towards composite Clouds built from sub Clouds, called ensembles[11].

D. System Interfaces

As defined above, interfaces expose the semantics of systems. We will elaborate on the interfaces of the systems examples presented earlier.

1) Operating Systems: A modern Linux OS has, for example, a number of interfaces: system calls, system tools (which mostly use the system calls), the `/proc` file system, raw devices, and others. These interfaces expose different aspects of both the OS itself, and also of the underlying resources. Often two interfaces expose different aspects of an underlying resource (think file system and raw disk device).

2) Grids: The interfaces exposed by general purpose Grids are mostly programmatic interfaces, e.g. web service interfaces plus client libraries to these web service interfaces. Additionally, tools (using a subset of the programmatic interface) provide the *most commonly used* capabilities in a convenient way to the end-user, e.g. as command line tools or GUIs.

A distinguishing, traditional property of Grid interfaces is that they aim at a very broad application space, and are thus expose many capabilities of the distributed Grid system. This is, arguably, one of the major reasons why Grids have not been as succesful as hoped in engendering distributed applications: the exposed interfaces are too rich, and the level of detail that needs to be controlled for the successful development and deployment of applications remains too high.

In particular, the WS-* services often employed by Grid systems expose very rich distributed system semantics. There is circumstantial evidence that this level of detail has failed the Grid user community, as (a) it is in practice not interoperable, as real implementations of these WS-* are rarely faithful to the standard or just wrong[1]; and (b) it is too hard to build software against these rich interfaces[12].

On the other hand, we think that the level of detail exposed by traditional Grid interfaces may be suitable for a different application space, e.g. that of specific narrow grids, high level distributed services, and, most importantly, Clouds. These systems with their inherent rich semantics may be able to make use of the rich Grid system interfaces.

General purpose Grids tend to expose *maximal* available semantics to the end-user, while narrow Grids tend to focus on a domain specific subset of the Grids semantics.

So called 'high level Grid APIs such as SAGA and CoG are an additional layer on top of Grids, providing additional system interfaces with increased simplicity and usability, while limiting the degree of semantics exposed.

3) Clouds: In contrast to the standard Grid system interfaces, cloud system interfaces are minimalistic: they provide a semantically very limited set of capabilities, and do not expose internal system characteristics: the exposed capability set is usually much more limited than the set of capabilities available *in* the Cloud system itself. The dominant consideration determining *which parts* of the system semantics are exposed via the Cloud interface are the Cloud's target *Usage Mode* (clouds seem to aim at exactly one usage mode, usually).

A distinguishing feature of Cloud interfaces are that instead of exposing the largest possible amount of semantics for a specific domain, **Clouds tend to expose a *minimal* amount of semantics to the end-user, while still being useful**.

E. Virtualization

Recent virtual machine technologies such as Xen, VMWare, amongst others provide excellent examples for virtualization: a well known system interface (the virtual machine and its OS) is provided on top of another similar system interface (the host resource), hiding details of the host system, providing seemingly exclusive access to the system resource (on the virtual machine), while actually performing concurrent resource sharing within the system (multiple virtual machines can run on the host resource).

F. Application

Although it may be intuitively obvious to the reader what an application is, we want to give an explicit example here.

Assume a distributed map-reduce creating a genome index of a genome dataset is running on the Amazon EC2 compute Cloud, using data from the Amazon S3 cloud. The map-reduce components (executables) are running on virtualized resources, which utilize physical resources, managed by EC2's Cloud system. For the end-user of the genetic information, the index creation algorithm is the application, with map-reduce being the programming model, and EC2/S3 being the systems used to run that application. The system interfaces utilized are the virtual machines of EC2, the REST/HTTP based interface of S3 for data access, and the REST/HTTP based interface for starting jobs on EC2.

It is important to stress however, that what may be an application for one user, may be consider as a system by another: Amazons EC2 cloud itself can well be considered an application of the underlying service layer.

G. Portal/Science Gateway/Application Environment

The usability of a system is greatly increased if a high level interface is provided to the end-user, which is designed to specifically support that users native work modus[2]. That can be achieved in multiple ways, depending on the prefered work environment of that user, or on the need to integrate with other, existing user tools.

With repect to the genetics application example discussed above: a portal which allows the end-user to easily switch from a data aquisition application (genome sequencing) to a data analysis application (the indexing described above) will greatly facilitate the usability of a system to the geneticist. Other styles of application environments, such as workflow environments, or command line tools etc, may achieve the same goal. The key here is the integration into the prevalent working environment of the end-user in a minimally disruptive way.

IV. Usage Modes and System Affinities

We state above that Grid system interfaces (in particular for general purpose Grids) tend to be complete (i.e. they try to expose a complete set of available system capabilities), and that Cloud interfaces tend to be minimalistic (i.e. they expose only a limited set of capabilities, just enough to 'do the job').

A. Usage Modes

It is important to understand the reason for this difference. In our experience, general purpose Grids are mostly designed bottom-up: existing, often heterogenous resources are federated as VOs, and their combined capabilities, plus additional capabilities of higher level Grid services, are offered to the end-user. This is not applicable for Clouds: the design of Clouds seems to be, mostly, top down. Clouds are designed to serve a limited, specific set of use cases and usage modes, and the Cloud system interface is designed to provide *that* functionality, and no other. Furthermore, the Cloud system itself, and in particular its high level services, may be designed to implement specific target use cases, while not supporting others (e.g., a Cloud could be homogenous by design). These differences do not imply that Clouds are trivial to implement. In practise the opposite is most likely true (due to issues of scale, amongst other things). Clouds may very well build upon general purpose Grids, or narrow Grids, and at least face the same challenges; but their system interfaces do not expose those internal capabilities.

Specific users and user communities tend to create different applications but with shared characteristics. For example, the particle data community tends to focus on very loosely coupled, data intensive parameter sweeps, Monte Carlo simulations, and statistical analysises. Systems used by these communities are thus designed to support these application classes before others.

The *Usage Mode* defined earlier tries to catch the dominant properties of the main application classes, insofar they are relevant to the design of the system, and to the operational properties of the system. For example, the usage mode *'massively distributed, loosely coupled'* implies that the system's design prioritizes on compute resources (e.g. cycle scavanging, large clusters), an to a lesser degree on communication (no need for fast links between application instances), or on reservation and co scheduling.

In contrast, the usage mode *'massively distributed, tighly coupled'* would imply a system's design to focus on compute resources, but importantly also on fast communication between near nodes, and on (physical) co-location of processes.

B. Affinities

Currently Clouds seem to be designed to mostly support exactly one usage mode, e.g. data storage, *or* high throughput computing, *or* databases, etc. This does not preclude Clouds targeting more than one domain or usage mode, however. The overarching design guideline to support the main target usage mode, of Cloud systems, we defined as its ***affinity***. In other words, affinity is the term we use to indicate the type of computational characteristics that a Cloud supports. That property can very often be expressed as the need to use different aspects or elements of a system *together* (hence the term 'Affinity', in the sense of 'closeness').

For example, the usage mode *'distributed, tightly coupled'* implies that an application requires the use of multiple compute resources, which need to be 'near' to each other, together with fast communication links between these compute resources. The system needs to have a *'compute-communication affinity'*, and a *'compute-compute affinity'*.

Affinities as used in this paper are, however, not always mappable to 'closeness'. For example, we say that a system that supports 'persistent storage, data replication, data intensive' usage mode, may have 'bulk storage affinity' – in the

[2]'work modus' in the sense of a day-to-day sequence of actions performed by an end-user to achieve a specific scientific, or commercial etc, goal.

sense that it needs to be designed to have bulk storage properties (availability guarantees, long term consistency guarantees etc). This example also shows that affinites are, in some sense, related to Quality of Service (QoS) properties exposed by the system, and thus to Service Level Agreements (SLAs) about these qualities of service.

C. Discussion

Affinity is thus a high level characterization of the kind of application that could be beneficially executed on a particular Cloud implementation, without revealing the specifics of the underlying arhitecture. In some ways, this is the "ideal abstraction" for the end-user who would like to use infrastructure as a black-box. Some classic examples of affinity are: tightly-coupled/MPI affinity, high-throughput affinity (capacity), fast-turnaround affinity (capability), or bulk storage affinity. Our observation is that Clouds have at least one affinity, a corollary to which is that Cloud system interfaces are, designed to serve at least one specific set of users or usage modes

One can argue that narrow Grids also expose affinity, e.g. that a Data Grid has data affinity. That may well be true, and we think that the term affinity may be useful for the discussion of narrow Grids as well, but the main difference between Clouds and Grids remain that the interfaces of narrow Grids are still designed so as to expose the complete set of capabilities related to the affinity of narrow Grids, whereas Cloud system interfaces expose the minimal set of capabilities related to its affinities. For the application developer, but more likely the application deployer, information about the affinity of Clouds should be complemented by SLA information, e.g. providing replicated data in case of loss, co-scheduling at the application level, or low latency communication. Traditionally SLAs are, implicitly or explicitly, provided by the "service provider" based upon infrastructure, policy, usage modes, or negotiation. For Clouds, SLAs are an implicit part of the system interface: the Cloud's affinities imply a certain QoS to be met, for every use of the system.

V. Observations

In this section, we list a number of high level observations made while investigating real world systems. It would go beyond the scope of the paper to discuss these observations in full detail. They are, however, useful in discussing the matter at hand.

Observation 1

System interfaces expose a complete semantic feature set as required by the set of target applications.

This observation may seem either trivial or run contradictory to our earlier examples, where we claim that broad Grids, for example, expose as much semantics as possible. The resolution of the apparent contradiction emerges when the large target application space is factored in, i.e., broad Grids have large semantic feature sets because they try to address a broad range of appplications and usage modes.

On the other hand, narrow Grids expose a subset of the semantics exposed by broad Grids. This is wholly consistent with the fact that narrow Grids are used by a smaller (narrower) set of target applications. Within the target semantic space, however, narrow Grids expose a complete set of semantics that any single application could use. So, a corollary to the observation above is: *If the target application space of a system is very narrow, the system interfaces tend to be narrow, too, i.e. tend to expose only the semantics required by that application space,* $\pm\epsilon$.

Example: Operating systems have an extremely broad target application space, and thus expose semantically powerful interfaces, which allow a range of services, distributed applications, purely number crunching codes, graphical user interfaces and others to be programmed. There are hardly any applications which require richer semantics to access the underlying resources, and if there were, extensions to the OS interface will emerge shortly (e.g., many operating systems allow for additional drivers and libraries to (ab)use the graphics card resources for number crunching).

Observation 2

Higher-level systems tend to support more specific target application and usage modes than lower-level systems.

Although higher-level systems, by aggregation or otherwise, can have richer semantics than lower level systems, they expose only a subset of that semantics via their interfaces, specifically targeting increasingly specific applications and application usage modes.

Example: Compared to the operating system example above, Grids expose only a small subset of semantics internally available: it is not possible to access resources like graphics cards directly via their service interfaces, but only via the underlying operating system interfaces. The reason is that applications performing graphical rendering on these cards are not in the target application space for Grids.

Observation 3

The narrower a system interface, the easier it is to use.

This observation has a definite subjective element, as ease of use is always in the eye of the beholder; but even then, we feel that narrow interfaces, by virtue of Observation 2, target specific usage scenarios, and are thus easier to use for these specific scenarios.

Example: Some people may find MPI more difficult to use than BSD sockets, although MPI is, without doubt, much narrower than sockets. One should keep in mind that the comparision should be done for the same application: sockets are certainly not that easy to use if you have to implement performance collective operations on a dynamically changing set of processes. MPI, as a narrow system interface, makes that specific use case simple.

VI. Implications

A. For System Architects

The observations above allude to order the real world systems discussed before by the semantic expressiveness of their interfaces. Please note that this order, as shown in fig. 2, does not neccessarily imply a system architecture, but is really just an ordering.

Obviously, applications can use any of the systems shown. We want to argue, however, that applications tend to use the highest level system possible, as that makes the appliciation development as simple as possible (Observation 2 & 3).

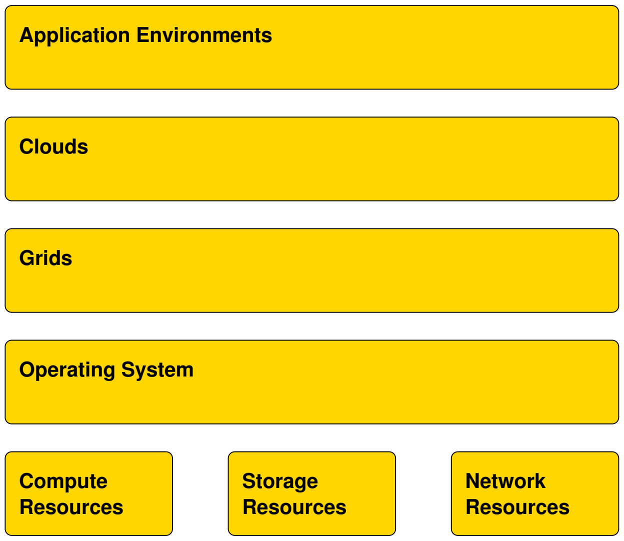

Fig. 2: Semantic ordering of an abstract representation of the entities discussed in this paper. The semantic complexity of entities decreases towards higher-levels; the usability of the entities from end-users perspective increases as the semantic complexity decreases.

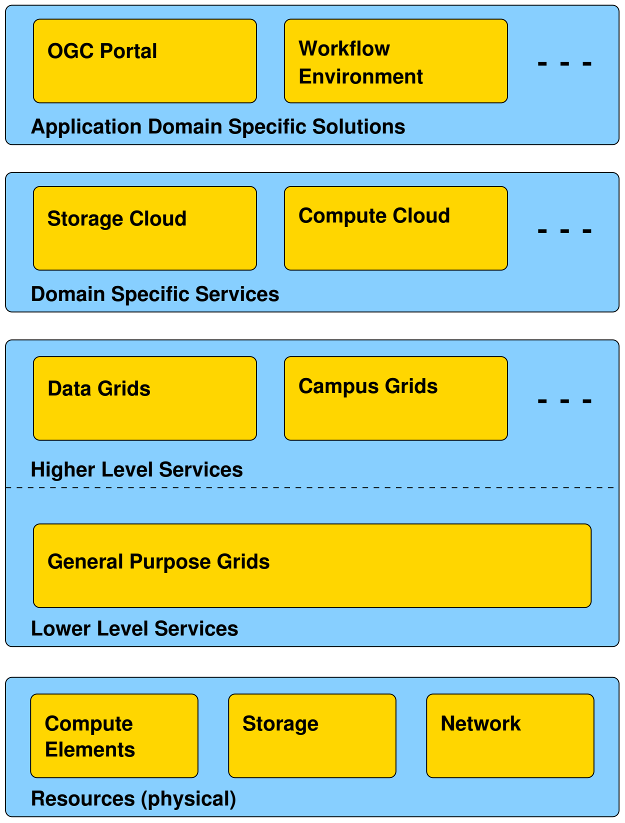

Fig. 3: A concrete realization of the abstract ordering in Figure 2, in form of a system architecture. This is a validation that although simple, the abstract ordering and relationship between entities can be useful to represent real systems.

Further, we think is is useful to consider the abover system order when designing systems. In particular, we find that the system order allows for a very generic architecture of real systems, as shown in fig. 3: this figure shows the same system order as above (Operating systems left out), but the annotations connect the examples we gave in the earlier discussion very nicely.

Assuming that this architecture is indeed able to describe real world systems, it seems that Grids are required to expose a number of core capabilities, required to implement scalable Clouds. Amongst these core capabilities we would count:

- system management and monitoring
- authorization / authentication / accounting
- resource virtualization (compute, data, communication)
- scalability, fail safety, QoS, SLA

These are all capabilities which are certainly required within a Cloud system, but are only partially (if at all) exposed to the Cloud application layer. One should also note that these capabilities are amongst the declared target capabilities for current Grid systems[13]!

B. For Resource Providers

Resource providers and thus system implementors should carefully look at their target user space. Ignoring the level of semantic abstraction required by the application space, and thus providing system interfaces which either expose too much semantics, or too little, will always result in interfaces which are cumbersome to use, or not to use at all.

There are clear imperatives for high level resource providers to adopt the Cloud model of utility computing: as Grids, Clouds are able to leverage economies of scale, and by supporting a limited but very common set of application classes, Clouds are able to reach out to a large fraction of the user base.

Given that Clouds are *large*, it is not obvious at the moment whether there is a strong need for Clouds to interoperate; but there is possibly a need for passive interoperation, often referred to as seamless access, i.e. the ability to submit to NSF or Amazon Clouds. Cloud's high level interfaces should make this easy, relative to current Grid interoperability efforts [1].

Further, Clouds seem to natively support an evolving internal infrastructure: it seems perfectly acceptable to keep engineering "the base" infrastructure, as long as this high-level interface is stable. In contrast, the detailed interfaces that Grids expose makes a straight forward evolution very difficult. This may pose an important advantage to resource providers, as it allows the evolution of their internal infrastructure, without major disruption of the service provided to the end users.

Finally, we want to shortly discuss the role of CPU virtualization in Clouds: Clouds have so far emphasized the utility of intra-CPU virtualization, as for example provided by systems such as Xen and VMWare. This could possibly be integrated with the inter-CPU virtualization provided by Grids, where brokers manage the deployment of services and applications on the most appropriate resource.

C. For Application Developers and End Users

We believe that program developers should be able to develop applications in the environment they are most comfortable with, and should not have to include details of their run-time environment into their development process in anything but the most simple way. This is in contrast to the approach where many applications are developed to be explicitly aware of their run-time distributed environment. Applications for Clouds can almost by definition be unaware of the the distributed environment they run.

Related to the need to be unaware of runtime environment details, is the need to provide simple interfaces, potentially in multiple renderings, to keep the application agnostic of the underlying system implementation. Clouds do have these simple interfaces, which are sometimes rendered in different technologies (For example, the S3 interfaces is provided as REST, SOAP, HTTP and BitTorrent renderings).

Both, the abstraction from resource mapping, and abstraction from resource details, lead us to the notion of abstarct applications: (i.e. applications that do not have run-time environment dependencies built into them). As an example, workflow descriptions are usually abstract applications: they can be mapped to different systems and resources, as long as the individual components do not depend on specific resource details.

Clouds, and other high level systems, seem to support that notion of abstract applications, and and provide a mechanism to create instances of these abstract applications, by binding them to specific resources. An additional bonus for the application developer is that abstract applications provide horizontal interoperability, in that these applications can be instantiated on different systems, with no or moderate porting effort, depending on the standardization of the system interfaces. That also avoids vendor lock-in, which may be crucial for a wider acceptance of Cloud technologies.

D. For OGF

The observation is that the development of Grid applications has proven difficult, as has the managment and deployment of Grids. How should a primarily Grid oriented standards organization such as the OGF respond to seemingly broad industrial support for Clouds?

In general, the discussed architecture for Grids and Clouds motivates opportunities for standards at two levels: on infrastructure level (core capabilities), and on Cloud interface level. We stated already that the need for latter is arguable, but motivated by passive interoperability for applications. The need for infrastructure standardization may also be arguable (Are the current Clouds built upon standards? Does it matter?), but (a) we feel that standardization is important for any future academic Cloud efforts, and (b) we believe that infrastructure standardization will enable companies to offer Cloud services on top of externally provided resources.

1) Standardization at the Interface Level: We have discussed earlier that the need for standard on Cloud interface level is arguable, as the current Clouds indicate that these interfaces are so simple that porting an application may be trivial. Nevertheless, we feel that for users who wish to composing interface from different Cloud systems into their thin or thick client applications will also want standards to enable them to interact with any Cloud service. Standardization activities to this Cloud service interface may be premature at this time as the Cloud usage models are not yet fully understood, but it is certainly an area that OGF could be involved in. The obvious next steps are to better understand the usage models.

There is currently no international group that is dealing with the standardization of interfaces to Cloud systems, the closest being the Computing Community Consortium who are organizing events to try to get parts of the Cloud community together. This is therefore a relatively green field for the OGF, who has the expertise and mandate to show how interfaces defined within OGF can be used to access Clouds. OGF also has the community to define use cases and develop core architectures/technologies.

Access to many of the services specified by OGF can be encapsulated within APIs such as those produced from the SAGA-WG. Applications developed using these high-level interfaces should seemlessly migrate to Clouds. In fact, the far future with standardized Cloud system interfaces may very well make the need for SAGA obsolete.

2) Standardization at the Core Capability Level: OGF offers a set of standards to support the compute aspects of resource services (HPC-BP, BES, DRMAA), and an emerging set of standards to support data resource services. To the OGF community's credit, impressive strides in the development of standard interfaces which are agnostic to the underlying architecture and infrastructure details have been made. We believe that these standards can form an essential core when designing and implementing Cloud systems.

Current Cloud implementations seem not to be overly concerned about the internal use of standardized system components (or at least do not document this). We want to remind the reader though that the percieved need for standardization is usually small when a technology is new, and only increases above a certain threshold with the broadness of adoption of that technology. We predict that, if Clouds continue to deliver to the application community, and if Clouds tecnology uptake will thus increase in the future, then the need for the standardization of Cloud internal system components will also increas. If these components will ultimately be Grids is a different question, and may well depend as much on technical as on social and political issues. Crucial however will be an early and open engagement of OGF and the Grid standards community, for any chance of involvement in Cloud system standardization.

E. For SAGA

SAGA[3] is an application level interface that provides a unified and consistent API to the most commonly used distributed functionality. Given possible changes in the development landscape, a pertinent question is: what might be the implications for SAGA? As a first step, there is clearly a need to understand the interfaces that are typically exposed by Clouds. However simple the native system interface, there is a need for programmatic support for application development and deployment via abstractions, e.g., providing abstractions to express and address the affinity of Clouds. The SAGA group in OGF should try to analyze if the notion of affinity can help by designing APIs which are oriented toward specific application domains and usage modes.

In general, the emergence of the Clouds with an emphasis on *usage modes* is an interesting complement to the SAGA approach, in that both are a top-down approaches to provide application oriented interfaces to developers. Whereas SAGA addresses that at the generic level of system interfaces, Clouds provide this, to some extent, at the internal system level, i.e. by providing capabilities required by applications as intrinsic part of the Cloud system. For example, support for MapReduce/Hadoop by major providers such as Google and Yahoo is just one indication of the utility and need for programming abstractions.

VII. Discussion and Open Issues

We have barely begun to understand Clouds as a viable distributed computing architecture and there are many technical issues, both internal and external, to a Cloud that remain to be formally addressed. A limited, random sampling of these are given below:

- The model of computing that a Cloud can support needs to be well defined, and is arguably the most important public attribute of a Cloud; we have introduced the concept of Cloud Affinity to address this important attribute. What types of internal configuration are available to support these? For example, can Clouds with suitable network connectivity between compute nodes provide affinity for capability distributed computing (i.e. multiple modest size MPI jobs)? We argue that the internal configuration should not be a public attribute that is exposed. We currently find "homogenous" Clouds, i.e. Clouds are currently either just data (S3) or compute Clouds (EC2) or just very large private (commercialized) data centers. Is that an intrinsic property, or can future Clouds be heterogeneous? That question would certainly revitalize the discussion about Grid/Cloud relationships!
- For most scientific computing needs, Clouds that provide only data storage facilities are probably going to be insufficient. Thus there is a need to introduce data-computing affinity, i.e. how easily can compute power be provided to data, or possibly how can data be moved across to the compute (without significant costs of transfer). Currently S3's business model charges for data transfer across S3 boundaries; but with network capacity on average doubling faster than compute capacity (though not storage capacity) there is clearly scope for "integrated" Clouds, at least at the logical level if not physically.
- Both Grid and Cloud systems are evolving technological fields, and thus there is a mine-field of unanswered questions, the answers to many of which will become obvious possibly only with hindsight:
  - Is there going to be a situation where we would want to link Clouds together either from different providers or between different functionalities (data Cloud to compute Cloud)?
  - If interoperability is required widely (if not universally), then what will be the model of aggregation of Cloud resources? A "Cloud of Grids"? A "Grid of Clouds"?
  - Will Clouds internally span cross-domain?
  - Are individual institutions or groups going to want to construct their own Clouds, as they have campus Grids?
  - Is there an underlying scheduler for the use case where demand exceeds supply and how would this affect externally available services?
  - What can end users expect in terms of fail safety of the Cloud system as a whole? How can users avoid vendor lock-in?
- Cloud Security has currently not been seriously explored. Note that Clouds do not cross administrative domains, at the moment, and this could simplify the discussion of security models compared to Grids. But if they do, is it likely that the security model will remain simpler? Either way, an important point will be that this does not show on the Cloud interface level.
- High-level interfaces have a role to play in making Clouds and other infrastructure easy to use. There is a need to address how utilizing distributed systems can be made easier via the use of abstractions, i.e. via support for commonly occuring patterns, which could be either programming patterns, application usage patterns and/or infrastructure usage patterns. High-level interfaces should make supporting programming abstractions easy, whether it be widely known and exploited abstractions such as Map-Reduce [14], or more recently adopted approaches such as All-Pairs [15] for data-intensive computing.
- Do Clouds have an effect on the distribution of computing infrastructure, as is commonly represented by the Branscomb Pyramid[16]? Clouds with different affinities and support for different usage modes, would seem to flatten the pyramid into several isolevel blocks.

Understanding these issues will be critical to a fuller appreciation of how Clouds are related to Grids beyond the obvious enhanced support for virtualization. Additionally, before any intellectually honest conjecture that Clouds are viable, useful systems can be made with any level of rigour, many of these

[3]**S**imple **API** for **G**rid **A**pplications, an OGF proposed recommendation. Disclaimer: two of the authors are co-chairs of the SAGA Working Group.

open issues and questions will need to be placed on a firm footing.

VIII. Conclusions

These are interesting times: Grids are clearly evolving both due to internal and technological pressures as well as external developments including market forces. As the efforts to build scalable systems with standardized interfaces have starting to yield dividends, the approach of Clouds – with the not so insignificant commercial interest behind – has also emerged as potentially competing approach for architecting large distributed systems.

We hope to be able to contribute to that discussion, (a) by providing a common terminology to build an analysis of Grids and Clouds, including the notion of *affinity*; and (b) by discussing the key differences between them: Grids on the one hand provide a wide semantic scope to a broad target space of distributed systems and applications; Clouds on the other hand expose a limited, if not minimal set of semantics to support a set of well defined usage modes.

These key differences allow us to re-evaluate a number of observations, and further to investigate a number of implications, for system architects, resource providers, application developers and end users, and for OGF and its SAGA effort. Finally, and not surprising, we are able to identify a significant number of open issues which need to be addressed in order to arrive at an considered opinion about the near term future of large scale distributed systems.

IX. Acknowledgements

SJ would like to acknowledge support from the e-Science Institute, Edinburgh for supporting the theme, “Distributed Programming Abstractions”. We thank Malcolm Atkinson for helpful discussions, and Steven Newhouse for slides of his presentation at OGF-Boston which lead to the formalization of the ideas around support for abstractions and usage modes.

References

[1] M. Marzolla, P. Andreetto, V. Venturi, A. Ferraro, S. Memon, S. Memon, B. Twedell, M. Riedel, D. Mallmann, A. Streit *et al.*, “Open Standards-Based Interoperability of Job Submission and Management Interfaces across the Grid Middleware Platforms gLite and UNICORE,” *IEEE International Conference on e-Science and Grid Computing*, pp. 592–601, 2007.

[2] “The Future of the TeraGrid, Position Papers.” [Online]. Available: http://www.teragridfuture.org/positionpapers

[3] C. Lee, “Evolutionary Pressures on the TeraGrid and the Larger Grid/Distributed Computing Community,” *The Future of the TeraGrid, Position Papers*. [Online]. Available: http://teragridfuture.org/node/222

[4] “Wikipedia.” [Online]. Available: http://en.wikipedia.org/

[5] I. Foster, “What is the Grid? A Three Point Checklist,” *Grid Today*, vol. 1, no. 6, pp. 22–25, 2002.

[6] C. Catlett, “The philosophy of TeraGrid: building an open, extensible, distributed TeraScale facility,” *Cluster Computing and the Grid 2nd IEEE/ACM International Symposium CCGRID2002*, pp. 5–5, 2002.

[7] Amazon, Inc., “Amazon Simple Storage Service.” [Online]. Available: aws.amazon.com/s3

[8] Amazon, Inc., “Amazon Elastic Compute Cloud.” [Online]. Available: aws.amazon.com/ec2

[9] e. a. Mayur Palankar, “Amazon S3 for Science Grids: a viable solution?” *submitted to DADC08 (private communication)*, 2008.

[10] Interview with Kate Keahey, “Converging Virtualization with Distributed Computing.” [Online]. Available: http://www.gridtoday.com/grid/1086063.html

[11] I. Wladawsky-Berger, “OGF-22 Keynote: Cloud Computing, Grids and the Coming IT Cambrian Explosion,” February 2008. [Online]. Available: http://www.ogf.org/gf/event_schedule/?id=1137

[12] S. Jha, H. Kaiser, A. Merzky, and O. Weidner, “Grid Interoperability at the Application Level Using SAGA,” *IEEE International Conference on e-Science and Grid Computing*, pp. 584–591, 2007.

[13] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, “The physiology of the Grid,” *Grid Computing: Making the Global Infrastructure a Reality*, 2003.

[14] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” *Operating Systems Design and Implementation (OSDI ’04)*, 2004.

[15] e. a. Christopher Moretti, “All-Pairs: An Abstraction for Data-Intensive Cloud Computing,” *submitted to IPDPS08)*, 2008. [Online]. Available: http://www.nd.edu/~dthain/papers/allpairs-ipdps08.pdf

[16] L. Branscomb, T. Belytschko, P. Bridenbaugh, T. Chay, J. Dozier, G. S. Grest, E. F. Hayes, B. Honig, N. Lane, J. William A. Lester, G. J. McRae, J. A. Sethian, B. Smith, and M. Vernon, “NSB 93-205 – NSF Blue Ribbon Panel on High Performance Computing,” October 1993.