Toward a Cloud Operating System

Fabio Pianese Peter Bosch Alessandro Duminuco Nico Janssens Thanos Stathopoulos† Moritz Steiner

Alcatel-Lucent Bell Labs

Service Infrastructure Research Dept. †Computer Systems and Security Dept.

{firstname.lastname}@alcatel-lucent.com

***Abstract*—Cloud computing is characterized today by a hotchpotch of elements and solutions, namely operating systems running on a single virtualized computing environment, middleware layers that attempt to combine physical and virtualized resources from multiple operating systems, and specialized application engines that leverage a key asset of the cloud service provider (e.g. Google's BigTable). Yet, there does not exist a virtual distributed operating system that ties together these cloud resources into a unified processing environment that is easy to program, flexible, scalable, self-managed, and dependable.**

In this position paper, we advocate the importance of a virtual distributed operating system, a *Cloud OS*, as a catalyst in unlocking the real potential of the Cloud—a computing platform with seemingly infinite CPU, memory, storage and network resources. Following established Operating Systems and Distributed Systems principles laid out by UNIX and subsequent research efforts, the Cloud OS aims to provide simple programming abstractions to available cloud resources, strong isolation techniques between Cloud processes, and strong integration with network resources. At the same time, our Cloud OS design is tailored to the challenging environment of the Cloud by emphasizing elasticity, autonomous decentralized management, and fault tolerance.

I. Introduction

The computing industry is radically changing the scale of its operations. While a few years ago typical deployed systems consisted of individual racks filled with few tens of computers, today's massive computing infrastructures are composed of multiple server farms, each built inside carefully engineered data centers that may host several tens of thousand CPU cores in extremely dense and space-efficient layouts [1]. There are several reasons for this development:

- Significant economies of scale in manufacturing and purchasing huge amounts of off-the-shelf hardware parts.
- Remarkable savings in power and cooling costs from the massive pooling of computers in dedicated facilities.
- Hardware advances that have made the use of system virtualization techniques viable and attractive.
- Commercial interest for a growing set of applications and services to be offloaded "into the Cloud".

The commoditization of computing is thus transforming processing, storage, and bandwidth into utilities such as electrical power, water, or telephone access. This process is already well under way, as today businesses of all sizes tend to outsource their computing infrastructures, often turning to external providers to fulfill their entire operational IT needs. The migration of services and applications into the network is also modifying how computing is perceived in the mainstream, turning what was once felt as the result of concrete equipment and processes into an abstract entity, devoid of any physical connotation: this is what the expression "Cloud computing" is currently describing.

Previous research has successfully investigated the viability of several approaches to managing large-scale pools of hardware, users, processes, and applications. The main concerns of these efforts were twofold: on one hand, exploring the technical issues such as the scalability limits of management techniques; on the other hand, understanding the real-world and "systemic" concerns such as ease of deployment and expressiveness of the user and programming interface.

Our main motivation lies in the fact that state-of-the-art management systems available today do not provide access to the Cloud in a uniform and coherent way. They either attempt to expose all the low-level details of the underlying pieces of hardware [2] or reduce the Cloud to a mere set of API calls—to instantiate and remotely control resources [3][4][5][6], to provide facilities such as data storage, CDN/streaming, and event queues [7][8][9], or to make available distributed computing library packages [10][11][12]. Yet, a major gap still has to be bridged in order to bond the Cloud resources into one unified processing environment that is easy to program, flexible, scalable, self-managing, and dependable.

In this position paper, we argue for a holistic approach to Cloud computing that transcends the limits of individual machines. We aim to provide a uniform abstraction—the Cloud Operating System—that adheres to well-established operating systems conventions, namely: (a) providing a simple and yet expressive set of Cloud metrics that can be understood by the applications and exploited according to individual policies and requirements, and (b) exposing a coherent and unified programming interface that can leverage the available network, CPU, and storage as the pooled resources of a large-scale distributed Cloud computer.

In Section II we elaborate our vision of a Cloud operating system, discuss our working assumptions, and state the requirements we aim to meet. In Section III we present a set of elements and features that we see as necessary in Cloud OS: distributed resource measurement and management techniques, resource abstraction models, and interfaces, both to the underlying hardware and to the users/programmers. We then briefly review the related work in Section IV, and conclude in Section V with our plans for the future.

II. The Cloud: a Unitary Computing System

The way we are accustomed to interact with computers has shaped over the years our expectations about what computers can and cannot achieve. The growth in processor power, the development of new human-machine interfaces, and the rise of the Internet have progressively turned an instrument initially meant to perform batch-mode calculations into the personal gateway for media processing and social networking we have today, which is an integral aspect of our lifestyle. The emergence of Clouds is about to further this evolution with effects that we are not yet able to foresee: as processing and storage move away from end-user equipment, the way people interact with smaller and increasingly pervasive pieces of connected hardware will probably change in exciting and unexpected ways.

To facilitate this evolution, we argue it is important to recognize that the established metaphor of the computer as a self-contained entity is now outdated and needs to be abandoned. Computer networks have reached such a high penetration that in most cases all of the programs and data that are ever accessed on a user machine have in fact been produced somewhere else and downloaded from somewhere else. While much more powerful than in the past, the CPU power and storage capacity of the hardware a user may have at her disposal pales compared to the power and storage of the hardware she is able to access over the Internet.

We feel that the times are ready for a new way to understand and approach the huge amount of distributed, interconnected resources that are already available on large-scale networks: the first Cloud computing infrastructures that start to be commercially available provide us with a concrete set of working assumptions on which to base the design of future computer systems. Research on distributed management of computer hardware and applications, together with the emergence of Internet-wide distributed systems, have provided a wealth of experiences and technical building blocks toward the goal of building and maintaining large-scale computer systems. However, users and developers still lack a definite perception about the potential of the Clouds, whose size and aggregate power are so large and hard to grasp: therefore we need to provide a new set of metaphors that unify and expose Cloud resources in a simple yet powerful way.

A. Assumptions on Cloud infrastructure

A Cloud is a logical entity composed of managed computing resources deployed in private facilities and interconnected over a public network, such as the Internet. Cloud machines (also called **nodes**) are comprised of inexpensive, off-the-shelf consumer-grade hardware. Clouds are comprised of a large number of **clusters** (i.e. sets of nodes contained in a same facility) whose size may range from a few machines to entire datacenters. Clusters may use sealed enclosures or be placed into secluded locations that might not be accessible on a regular basis, a factor that hinders access and maintenance activities. Clusters are sparsely hosted in a number of locations across the world, as their deployment is driven by such practical concerns as:

- Availability of adequate facilities in strategic locations
- High-bandwidth, multi-homed Internet connectivity
- Presence of a reliable supply of cheap electrical power
- Suitable geological and/or metereological properties (Cold weather, nearby lakes or glaciers for cheap cooling)
- Presence of special legal regimes (data protection, etc.)

The clusters' computing and networking hardware is said to be *inside the perimeter* of the Cloud. The network that connects the clusters and provides the expected networking services (IP routing, DNS naming, etc.) lies *outside* the Cloud perimeter.

The reliance on commodity hardware and the reduced servicing capability compel us to treat Cloud hardware as *unreliable* and prone to malfunction and failures. The network needs also to be considered unreliable, as a single failure of a piece of network equipment can impact a potentially large number of computers at the same time. A typical mode of network failure introduces *partitions* in the global connectivity which disrupt the end-to-end behavior of the affected transport-layer connections. Network issues may arise both inside the Cloud perimeter, where counter-measures may be available to address them quickly in a way that is transparent to most applications, and outside, where it is not possible to react as timely.

We do not intend to formulate any specific assumption on the applications that will run on the Cloud computer. In other words, we expect to satisfy the whole range of current applicative requirements, e.g. CPU-intensive number crunching, storage-intensive distributed backup, or network-intensive bulk data distribution (and combinations thereof). The Cloud operating system aims to be as general purpose as possible, providing a simple set of interfaces for the management of Cloud resources: all the policy decisions pertaining to the use of the available resources are left to the individual applications.

B. Cloud OS: a familiar OS metaphor for the Cloud

We formulate a new metaphor, the Cloud operating system, that may be adequate to support the transition from individual computers as the atomic "computational units" to large-scale, seamless, distributed computer systems. The Cloud OS aims to provide a familiar interface for developing and deploying massively scalable distributed applications on behalf of a large number of users, exploiting the seemingly infinite CPU, storage, and bandwidth provided by the Cloud infrastructure.

The features of the Cloud OS aim to be an extension to those of modern operating systems, such as UNIX and its successors: in addition to simple programming abstractions and strong isolation techniques between users and applications, we emphasize the need to provide a much stronger level of integration with network resources. In this respect, there is much to be learnt from the successor of UNIX, Plan 9 from Bell Labs [13], which extended the consistency of the "everything is a file" metaphor to a number of inconsistent aspects of the UNIX environment. More useful lessons on techniques of process distribution and remote execution can

be drawn from earlier distributed operating systems such as Amoeba [14].

While a traditional OS is a piece of software that manages the hardware devices present in a computer, the Cloud OS is a set of distributed processes whose purpose is the management of Cloud resources. Analogies to established concepts can therefore help us to describe the kind of features and interfaces we wish to have in a Cloud OS, ignoring for the moment the obvious differences of scale and implementation between the two scenarios:

- an OS is a collection of routines (scheduler, virtual memory allocator, file system code, interrupt handlers, etc.) that regulate the access by software to CPU, memory, disk, and other hardware peripherals; the Cloud OS provides an additional set of functionalities that give administrative access to resources in the Cloud: allocate and deallocate virtual machines, dispatch and migrate processes, setup inter-process communication, etc.
- an OS provides a standard library of system calls which programs can use to interact with the underlying hardware; the Cloud OS provides a set of network-based interfaces that applications can use to query the management system and control Cloud resources.
- an OS includes a standard distribution of libraries and software packages; the Cloud OS includes software support for the autonomous scaling and opportunistic deployment of distributed applications.

C. Toward seamless access to networked resources

The Cloud OS interface implements, as a straightforward extension of a traditional OS interface, those additional functions and abstractions that will facilitate the access to remote resources. Instances of software running on the Cloud are expected to communicate over a variety of scopes (same node, same cluster, between nodes in remote clusters) using network links whose behavior and properties can be very diverse. A major challenge of the Cloud environment is thus characterizing the network infrastructure in a *much more expressive* way compared to existing distributed operating systems: in order to support applications that are sensitive to inter-node latencies or that require heavy network activity, the *cost*[1] (in terms of link latency, expected transmission delay, delay required to initialize a new virtual machine, etc.) of accessing a Cloud resource needs to be advertised.

Our work participates in the current attempts at rethinking the relationship between processing and networking at any scale—ranging from individual multi-processor systems [15] to datacenters [16]—in order to encompass large-scale Cloud architectures. Baumann *et al.* [15] observe that current multi-core processors are actually networked systems, and instead of building a traditional kernel for such processors with processor cores competing for shared locks, they adopt a message-based, distributed approach that ultimately reaches a higher efficiency. Costa *et al.* [16] argue for a higher degree of integration between services and networks in the datacenter by exposing detailed knowledge about network topology and performance of individual links, which allows the adoption of automated network management techniques such as application-aware multi-hop routing or efficient local-scope multicast.

The main purpose of our Cloud OS is the introduction of a new set of abstractions to represent the previously hidden cost required to access networked Cloud resources in a synthetic and sufficiently accurate way. It is our primary concern to provide to users and developers alike with a consistent view over the Cloud resources so as to encourage them to write new forms of distributed applications. Developers in particular typically expect to focus primarily on the functional requirements of the applications, rather than on the complicated trade-offs of distributed programming. The Cloud OS therefore will provide a library of standard functionalities, such as naming, consistent replication and elastic application deployment that attempt to cover the common requirements of distributed applications.

D. Cloud OS requirements

Whereas current datacenter setups can offer a fine-grained amount of control and pervasive management capabilities, the Cloud environment is much less predictable and harder to control: the environment imposes therefore several restrictions to the Cloud OS design, such as the reliance on coarse-grained knowledge about Cloud resource availability, the need to detect and tolerate failures and partitions, and a lack of global view over the system state. Despite these limitations, our design aims to meet the following general requirements:

a) The Cloud OS must permit autonomous management of its resources on behalf of its users and applications: Our main purpose is providing an abstraction of the Cloud as a coherent system beyond the individual pieces of hardware from which it is built. The Cloud OS should therefore expose a consistent and unified interface that conceals whenever possible the fact that individual nodes are involved in its operations, and what those low-level operations are.

b) Cloud OS operation must continue despite loss of nodes, entire clusters, and network partitioning: Conforming to our assumptions, we expect that every system component, including networks, may unexpectedly fail, either temporarily or permanently. Guaranteeing continued operation of the Cloud management processes in these conditions involves mechanisms for quickly detecting the failures and enacting appropriate measures. Note that fault-tolerance at the Cloud level does not imply any guarantee about the fault-tolerance of individual applications: the state of any process could suddenly disappear because of any of the previous events, therefore Cloud applications should be designed with this in mind. Several Cloud libraries that implement common fault-tolerance and state recovery features are provided out of the box.

c) The Cloud OS must be operating system and architecture agnostic: The network is the common interface boundary between the various software elements of the Cloud. The reason for this choice is that we want to enable the broadest

[1] In commercial Cloud deployments, cost may also include the actual pricing of the resources consumed by the applications.

compatibility between hardware and software configurations, while providing at the same time an easy way for future evolution of the Cloud system, both at a global and at an individual subsystem level. Experience shows that protocols are able to withstand time much better than ABIs, standard library specifications, and file formats: long-lived protocols such as the X protocol and HTTP are good examples in this regard. While it is wise from an operational standpoint to consolidate the number of architectures supported and standardize around a small number of software platforms, the Cloud OS operation does not depend on any closed set of platforms and architectures.

d) The Cloud must support multiple types of applications, including legacy: In the assumptions above, we purposefully did not specify a target set of applications that the Cloud is supposed to host. Rather than optimizing the system for a specific mode of operation (e.g. high performance computing, high data availability, high network throughput, etc.), we aim to address the much broader requirements of a general-purpose scenario: applications of every type should ideally coexist and obtain from the system the resources that best match the application requirements.

e) The Cloud OS management system must be decentralized, scalable, have little overhead per user and per machine, and be cost effective: The use of such a soft-state approach takes inspiration from recent peer-to-peer techniques: these systems are capable of withstanding failures and churn at the price of a reasonable amount of network overhead, and provide enough scalability to meet and surpass the magnitudes of today's datacenters and large-scale testbeds. Moreover, apart from initial resource deployment and key distribution, no human intervention should be required to expand the Cloud resources. Likewise, user management should only entail the on-demand creation of user credentials, which are then automatically propagated throughout the Cloud.

f) The resources used in the Cloud architecture must be accountable, e.g. for billing and debugging purposes: The cost of an application's deployment across the Cloud is also a part of the end-to-end metrics that may influence the scheduling of resources as per an application's own policy. Moreover, dynamic billing schemes based e.g. on resource congestion [17] could be an effective way to locally encourage a proportionally fair behavior among users of the system and increase the cost of attacks based on maliciously targeted resource allocation [18].

III. Toward a Cloud Operating System

In this section, we present the architecture and functional building blocks of the Cloud OS. Our current design approach leverages decades of experience in building networked systems, from the origins of the Internet architecture [19] to subsequent achievements in distributed operating systems research [13] and large-scale network testbed administration [20]. An additional inspiration, especially concerning the implementation of Cloud OS, comes from the last decade of advances in distributed algorithms and peer-to-peer systems.

A. Logical architecture of the Cloud OS

Figure 1 represents a logical model of Cloud OS. We define the **Cloud object** as a set of local OS processes running on an single node, which are wrapped together and assigned locally a random identifier of suitable length to minimize the risk of system-wide ID collisions. A **Cloud process (CP)** is a collection of Cloud objects that implement the same (usually distributed) application.

We refer to the small number of CPs that regulate physical allocation, access control, accounting, and measurements of resources as the *Cloud kernel space*. Those CPs that do not belong to kernel space pertain to the *Cloud user space*. User space CPs that are executed directly by users are called **User Applications**, while **Cloud Libraries** are CPs typically called upon by Applications and other Libraries. Applications can interface with Libraries and kernel CPs over the network through a set of standard interfaces called **Cloud System Calls**[2]. The assumptions stated above pose very few constraints about the features that the underlying Cloud hardware is expected to provide. Basically, the ability to execute the Cloud kernel processes, together with the availability of appropriate trust credentials, is a sufficient condition for a node to be part of the Cloud[3]. A limited access to Cloud abstractions and interfaces is thus also achievable from machines that belong to administrative domains other than that of the Cloud provider, with possible restrictions due to the extent of the management rights available there.

All objects in the Cloud user space expose a Cloud system call handler to catch signals from the Cloud OS, i.e. they can be accessed via a network-based interface for management purposes. The association between object names and their network address and port is maintained by the **process management** and **virtual machine management** kernel CPs, and the resulting information is made available throughout the Cloud via the **naming** Library. The naming library also keeps track of the link between User Application CPs and the objects they are composed of. The access rights necessary for all management operations are granted and verified by the **authentication** kernel CP. **Measurement** kernel CPs are always active in the Cloud and operate in both on-demand and background modes.

B. Implementation of the Cloud kernel processes

1) Resource measurement: The Cloud OS needs to maintain an approximate view of the available Cloud resources. Our current approach involves performing local measurements on each Cloud node. This technique provides easy access to end-to-end variables such as latency, bandwidth, packet loss rate, etc., which are precious sources of knowledge that are directly exploitable by the applications. More detailed knowledge

[2]The network protocols used among objects that belong to a same CP are outside the scope of the Cloud OS interface definition.

[3]Nodes that are not part of the Cloud are still capable to access Cloud resources using the network-based system call interface; however, without a full OS-level support for Cloud abstractions, they won't provide seamless integration between the local and the Cloud environment.

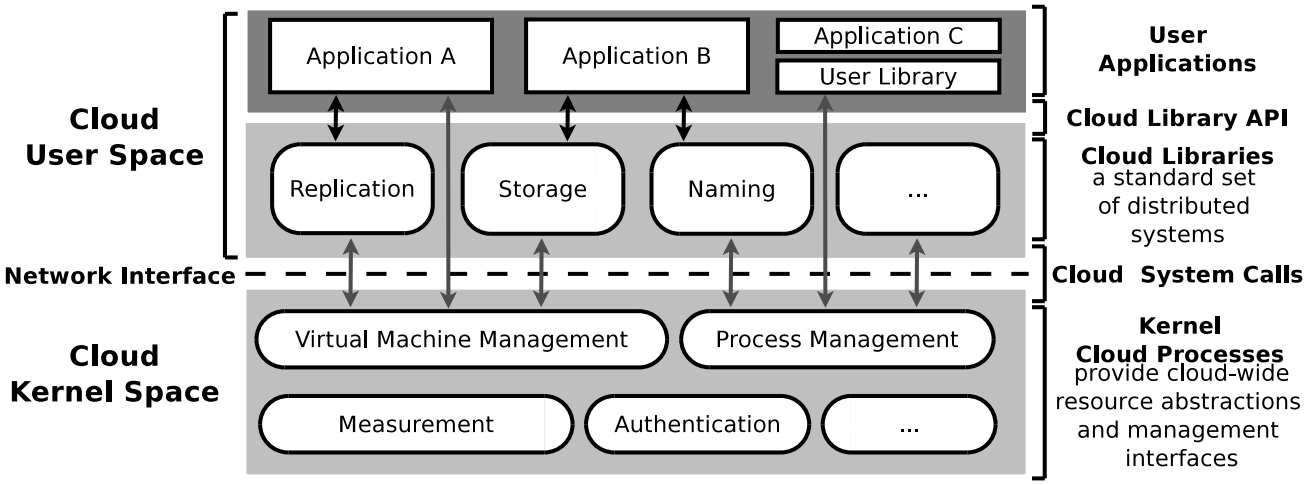

Figure 1. Logical model of Cloud OS, featuring the division between Cloud kernel / Cloud user space and the system call and library API interfaces.

requires *complete control* over the network infrastructure, but it may be used in certain cases to augment the accuracy of end-to-end measurements (e.g., with short-term predictions of CPU load or networking performance [21]) in Clouds that span several datacenters.

Measurements can target either *local* quantities, i.e. inside a single Cloud node, or *pairwise* quantities, i.e. involving pairs of connected machines (e.g. link bandwidth, latency, etc.). Complete measurements of pairwise quantities cannot be performed in large-scale systems, as the number of measurement operations required grows quadratically with the size of the Cloud. Several distributed algorithms to predict latencies without global measurement campaigns have been proposed: Vivaldi [22] collects local latency samples and represents nodes as points in a coordinate system. Meridian [23] uses an overlay network to recursively select machines that are the closest to a given network host. Bandwidth estimation in Cloud environments remains an open problem: despite the existence of a number of established techniques [24], most of them are too intrusive and unsuitable for simultaneous use and to perform repeated measurements on high capacity links.

2) Resource abstraction: Modern OS metaphors, such as the "everything is a file" model used by UNIX and Plan9, provide transparent network interfaces and completely hide their properties and specificities from the applications. However, characterizing the underlying network is a crucial exigence for a Cloud OS, for network properties such as pairwise latencies, available bandwidth, etc., determine the ability of distributed applications to efficiently exploit the available resources. One major strength of a file-based interface is that it is very flexible and its shortcomings can be supplemented with an appropriate use of naming conventions. We are considering several such mechanisms to present abstracted resource information from measurements to the applications, e.g. via appropriate extensions of the */proc* interface or via POSIX-compatible semantic cues [25].

In order to present information about resources to the user applications, the Cloud OS needs first to collect and aggregate them it in a timely way. Clearly, solutions based on centralized databases are not viable, since they lack the fault-tolerance and the scalability we require. The use of distributed systems, such as distributed hash tables (DHTs), has proved to be very effective for publishing and retrieving information in large-scale systems [26], even in presence of considerable levels of churn [27]. However, DHTs offer hash table (key, value) semantic, which are not expressive enough to support more complex queries such as those used while searching for resources. Multi-dimensional DHTs [28][29] and gossip-based approaches [30] extended the base (key, value) semantic in order to allow multi-criteria and range queries.

3) Distributed process and application management: The Cloud OS instantiates and manages all objects that exist across the Cloud nodes. A consolidated practice is the use of virtual machines (VMs), which provide an abstraction that flexibly decouples the "logical" computing resources from the underlying physical Cloud nodes. Virtualization provides several properties required in a Cloud environment [31], such as the support for multiple OS platforms on the same node and the implicit isolation (up to a certain extent) between processes running on different VMs on the same hardware. Computation elasticity, load balancing, and other optimization requirements introduce the need for dynamic allocation of resources such as the ability to relocate a running process between two nodes in the Cloud. This can be done either at the Cloud process level, i.e. migrating single processes between nodes, or at virtual machine level, i.e checkpointing and restoring the whole VM state on a different node. The combination of process and VM migration, such as it was introduced by MOSIX [32], is very interesting as a Cloud functionality as it allows to autonomously regroup and migrate bundles of related Cloud objects with a single logical operation[4].

[4]Another compelling approach is Libra [33], which aims to bridge the distance between processes and VMs: the "guest" operating system is reduced to a thin layer on top of the hypervisor that accesses the functions exposed by the "host" operating system through a file-system interface.

The Cloud operating system must also provide an interface to manage processes from a user perspective. This requires the introduction of an abstraction to aggregate all the different computational resources is a single view. A good example on how to do this is the recent Unified Execution Model (UEM) proposal [34], which structures the interface as a directory tree similar to the Plan 9 */proc* file system and provides an intuitive way to create, copy, and move processes between Cloud nodes. The novelty of the UEM approach is the possibility to "switch the view" on the Cloud process namespace, using a simple extension of common shell built-in commands, inducing a transition e.g. from a per-node view of the process environment to an application-based list of running Cloud processes and objects.

4) Access control and user authentication: Providing seamless support for large numbers of simultaneous users requires a distributed authentication method to avoid single points of failure, resulting in the complete or partial inaccessibility to Cloud resources. Plan 9 provides a very interesting distributed security model [35] which is based on a *factotum* server, running on every machine, that authenticates the users providing a single-sign-on facility based on secure capabilities. Authentication of users by factotum needs to be supported by a Cloud-wide system for securely distributing and storing user credentials, which could be seen as a scaled-up, distributed version of the Plan 9 *secstore* service.

C. Features provided by the Cloud user space

In order to fully exploit the potential of a general purpose Cloud OS, developers should be given access to a set of standard ways to satisfy common requirements of distributed large-scale applications. Cloud libraries provide a standard API with features such as:

- access to Cloud-wide object and process naming via DNS and/or other distributed naming services [36],
- distributed reliable storage functionality [25][37],
- automated Cloud application deployment, horizontal scaling, and lifecycle management [38],
- high availability failover support with checkpointed replicated process execution.

As a general principle, the Cloud libraries provided by the Cloud OS should allow the developers to control the required level of data replication, consistency, and availability, and also the way failure handling is performed when application requirements are not satisfied. This way, an application developer can concentrate her attention on the specific properties of the application, knowing that the system will try its best to accommodate the stated requirements. For instance, when an application demands high availability and is capable of dealing with temporary inconsistencies, the library may provide eventual consistency support, instead of stronger consistency.

The advantages of this configurability are twofold. On one hand, an application developer doesn't need to implement her own customized (application-specific) libraries, but instead can customize the level of support she needs from the Cloud library API. This reduces the complexity and error proneness of application development. On the other hand, the approach above promotes re-usability of Cloud application components, which can be easily adapted to satisfy different specifications just by updating a minor amount of embedded Cloud library parameters.

IV. Related work

Our work draws from a number of different fields and research topics: distributed operating systems, remote application management, and large-scale peer-to-peer systems. With Cloud OS, we aim to achieve an original synthesis of the three: we propose an architecture-agnostic functional platform to support the efficient implementation and seamless deployment of new distributed applications.

A. Grid and Cloud middleware

Over the past few years, the adoption of cloud computing in enterprise development has been technologically supported by a very heterogeneous set of cloud services. IBM Research Center recently stated that the heterogeneous nature of this solution space hinders the adoption of Cloud technologies by major enterprises [39]. In [40], the authors argue that a "Cloud middleware" should solve this by homogenizing the cloud interface. To justify this observation, IBM developed the Altocumulus middleware, which offers a uniform API for using Amazon EC2, Eucalyptus, Google AppEngine, and IBM HiPODS Cloud, aiming to provide an API which is Cloud agnostic (i.e., the integration of new public clouds requires the development of specific adaptors). Although our work aims to offer a homogeneous cloud interface as well, our work adopts a bottom-up instead of a top-down approach: instead of providing means to move existing solutions under one single Cloud umbrella, we seek to offer a Cloud OS that is able to seamlessly control private and public cloud resources.

The Globus toolkit [3] is a set of libraries and services that address security, information discovery, resource management, data management, communication, fault-detection and portability in a Grid environment. The lack of consistent interfaces make the toolkit difficult to install, manage, and use. While the generalization done by Globus is essentially at the application level (indeed services virtualize components that live in the application level), we try to operate at a lower level, trying to provide the user with an operating system abstraction. In our approach, what is provided to users are raw resources (CPU, network bandwidth, storage) and an interface to operate on processes, instead of higher-level services.

Other Cloud middleware initiatives like SmartFrog [38] have focused on managing the usage of private and public cloud computing infrastructure. Similar to our Cloud OS, SmartFrog supports transparent deployment of client services and applications on the available resources, allows for on-demand scaling, and provides failure recovery features. Note, however, that the SmartFrog middleware assumes Cloud applications are written in Java. Our Cloud OS, in contrast, provides a language and technology agnostic API including a set of

network protocols. This way, the usability of our Cloud OS will not be restricted by technology lock-in.

Among the related literature, the goals of XtreemOS [41] are the closest to ours. It aims at the design and implementation of an open source Grid operating system which would be capable of running on a wide range of underlying platforms, spanning from clusters to mobiles. It implements a set of system services that extend those found in a typical Linux system. This is different from our approach, which focuses on network protocols and has no specific requirements concerning architecture or operating system running on a node. XtreemOS aims to be completely transparent to the applications, whereas we want to provide additional information and hooks to the application developers.

B. Global resource management

Early systems for monitoring and managing resources, such as Remos [42] and Darwin [43] used a centralized architecture that suffered from clear scalability and reliability problems. Later, Astrolabe [44] adopted a distributed approach focused on scalability: Astrolabe creates a hierarchy of zones and inside each of them the availability of resources is advertised via a gossiping protocol. At each level of the hierarchy, the state of the resources gets incrementally aggregated. This approach hinders the efficient compilation of the list of nodes that match a given job requirement. Like Astrolabe, [30] also relies on gossiping to locate appropriate nodes, but doesn't use a hierarchical infrastructure: gossip messages are routed via subsequent hops across a multidimensional space to reach nodes that satisfy all the specified constraints. The SWORD [45] resource discovery service builds upon a DHT, a structured overlay [46]. SWORD generates a different key for each attribute based on its current value. Range queries are efficient in SWORD, since each node is responsible for a continuous range of values [29].

C. Remote application management

The first attempts to manage the computing resources of a large number of remote machines were SETI@Home [47] in the mid-nineties, followed by its successor BOINC [48] and many other similar projects. The immense computing resource that could be harnessed via a simple, centralized mechanism first tangibly illustrated the power of large-scale distributed batch computing.

Distributed application management framework soon abandoned the initial centralized architecture, leveraging techniques from peer-to-peer search and file-sharing networks: these systems need in fact to integrate resource discovery, application monitoring, and remote management [49][50]. The first generation of resource management systems was based on early peer-to-peer architectures. Zorilla [51] is very similar to Gnutella, thus suffering from the same scalability issues [52]. Job advertisements are flooded to other nodes in the system. Upon reception of an advertisement, the node checks if all the requirements of that job are met at the current time, and acknowledges or ignores the advertisement. In case of a node failure, Zorilla will find a new node and notify the application, leaving to the application the implementation of mechanisms for failure recovery.

Plush [53] makes use of SWORD to provide an application management infrastructure service. It consists of tools for automatic deployment and monitoring of applications on large-scale testbeds such as PlanetLab. To do so it uses a set of application controllers that run on each node, whereas, the execution of the distributed application is controlled by a centralized manager. While one requirement of our Cloud OS is to be scalable, Plush has been able to manage only up to 1000 nodes in emulations and 500 nodes in a real-world deployment.

V. Future directions

The existence of simple yet powerful and expressive abstractions is essential in realizing the full potential of Cloud Computing. To this purpose we introduced the Cloud operating system, *Cloud OS*. Cloud OS aims to provide an expressive set of resource management options and metrics to applications to facilitate programming in the Cloud, while at the same time exposing a coherent and unified programming interface to the underlying distributed hardware. This unified interface will provide developers with a quick and transparent access to a massively scalable computing and networking environment, allowing the implementation of robust, elastic, and efficient distributed applications. Our next steps beyond laying out the architecture of CloudOS include, first, a detailed definition of functional elements and interfaces of the kernel-space Cloud processes and of the user-space libraries, and second, the design and implementation of the aforementioned elements with emphasis on fault-tolerance, security, and elasticity.

References

[1] "HP performance-optimized datacenter (POD)." Data Sheet, 2008.

[2] IBM, "Tivoli netview." [Online] http://www-01.ibm.com/software/tivoli/products/netview/.

[3] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems integration," in *Open Grid Service Infrastructure WG, Global Grid Forum*, 2002.

[4] "Amazon EC2." [Online] http://aws.amazon.com/ec2.

[5] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proc. of Cloud Computing and Its Applications*, Oct. 2008.

[6] B. Rochwerger, A. Galis, D. Breitgand, E. Levy, J. Cáceres, I. Llorente, Y. Wolfsthal, M. Wusthoff, S. Clayman, C. Chapman, W. Emmerich, E. Elmroth, and R. Montero, "Design for future internet service infrastructures," in *Future Internet Assembly 2009, Prague, Czech Republic*, In "Towards the Future Internet - A European Research Perspective", pp. 227-237, IOS Press, May 2009.

[7] "Amazon S3." [Online] http://aws.amazon.com/s3/.

[8] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[9] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," in *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[10] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Sixth Symposium on Operating System Design and Implementation (OSDI)*, 2004.

[11] "Apache Hadoop project." [Online] http://hadoop.apache.org/.

[12] "Google AppEngine." [Online] http://appengine.google.com.

[13] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom, "Plan 9 from Bell Labs," *Computing Systems*, vol. 8, no. 3, pp. 221–254, 1995.

[14] S. J. Mullender, G. van Rossum, A. S. Tanenbaum, R. van Renesse, and H. van Staveren, "Amoeba: A distributed operating system for the 1990s," *IEEE Computer*, vol. 23, no. 5, pp. 44–53, 1990.

[15] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania, "The multikernel: A new OS architecture for scalable multicore systems," in *Proceedings of the 22nd ACM Symposium on OS Principles (SOSP)*, 2009.

[16] P. Costa, T. Zahn, A. Rowstron, G. O'Shea, and S. Schubert, "Why Should We Integrate Services, Servers, and Networking in a Data Center?," in *Proceedings of the International Workshop on Research on Enterprise Networking (WREN'09), co-located with ACM SIGCOMM'09*, (Barcelona, Spain), pp. 111–118, ACM Press, August 2009.

[17] R. Gibbens and F. Kelly, "Resource pricing and the evolution of congestion control," *Automatica*, vol. 35, pp. 1969–1985, 1999.

[18] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proceedings of the ACM Conference on Computer and Communications Security*, (Chicago, IL), November 2009.

[19] D. D. Clark, "The design philosophy of the darpa internet protocols," in *Proc. SIGCOMM '88, Computer Communication Review Vol. 18, No. 4, August 1988*, pp. 106–114, 1988.

[20] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir, "Experiences building Planetlab," in *In Proceedings of the 7th USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, 2006.

[21] P. A. Dinda and D. R. O'Hallaron, "An extensible toolkit for resource prediction in distributed systems," Tech. Rep. CMU-CS-99-138, School of Computer Science , Carnegie Mellon University, 1999.

[22] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *In Proc. of ACM SIGCOMM*, 2004.

[23] B. Wong, A. Slivkins, and E. G. Sirer, "Meridian: a lightweight network location service without virtual coordinates," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 85–96, 2005.

[24] R. S. Prasad, M. Murray, C. Dovrolis, and K. Claffy, "Bandwidth estimation: Metrics, measurement techniques, and tools," *IEEE Network*, vol. 17, pp. 27–35, 2003.

[25] J. Stribling, Y. Sovran, I. Zhang, X. Pretzer, J. Li, M. F. Kaashoek, and R. Morris, "Flexible, wide-area storage for distributed systems with wheelfs," in *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, (Berkeley, CA, USA), pp. 43–58, USENIX Association, 2009.

[26] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, (London, UK), pp. 53–65, Springer-Verlag, 2002.

[27] M. Steiner, T. En-Najjary, and E. W. Biersack, "A global view of KAD," in *IMC 2007, ACM SIGCOMM Internet Measurement Conference, October 23-26, 2007, San Diego, USA*, 10 2007.

[28] P. Ganesan, B. Yang, and H. Garcia-Molina, "One torus to rule them all: multi-dimensional queries in p2p systems," in *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases*, (New York, NY, USA), pp. 19–24, ACM, 2004.

[29] A. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting scalable multi-attribute range queries," in *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 353–366, 2004.

[30] P. Costa, J. Napper, G. Pierre, and M. V. Steen, "Autonomous Resource Selection for Decentralized Utility Computing," in *Proceedings of the 29^{th} IEEE International Conference on Distributed Computing Systems (ICDCS 2009)*, (Montreal, Canada), June 2009.

[31] R. Figueiredo, P. Dinda, and J. Fortes, "A case for grid computing on virtual machines," in *International Conference on Distributed Computing Systems (ICDCS)*, vol. 23, pp. 550–559, 2003.

[32] T. Maoz, A. Barak, and L. Amar, "Combining virtual machine migration with process migration for hpc on multi-clusters and grids," in *IEEE Cluster 2008, Tsukuba*, 2008.

[33] G. Ammons, J. Appavoo, M. Butrico, D. Da Silva, D. Grove, K. Kawachiya, O. Krieger, B. Rosenburg, E. Van Hensbergen, and R. W. Wisniewski, "Libra: a library operating system for a jvm in a virtualized execution environment," in *VEE '07: Proceedings of the 3rd international conference on Virtual execution environments*, (New York, NY, USA), pp. 44–54, ACM, 2007.

[34] E. V. Hensbergen, N. P. Evans, and P. Stanley-Marbell, "A unified execution model for cloud computing," in *In Proc. of the 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware (LADIS)*, 2009.

[35] R. Cox, E. Grosse, R. Pike, D. L. Presotto, and S. Quinlan, "Security in plan 9," in *Proceedings of the 11th USENIX Security Symposium*, (Berkeley, CA, USA), pp. 3–16, USENIX Association, 2002.

[36] V. Ramasubramanian and E. G. Sirer, "The design and implementation of a next generation name service for the internet," in *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, (New York, NY, USA), pp. 331–342, ACM, 2004.

[37] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, 2007.

[38] P. Goldsack, J. Guijarro, S. Loughran, A. Coles, A. Farrell, A. Lain, P. Murray, and P. Toft, "The smartfrog configuration management framework," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 1, pp. 16–25, 2009.

[39] E. M. Maximilien, A. Ranabahu, R. Engehausen, and L. C. Anderson, "Toward cloud-agnostic middlewares," in *Proceedings of OOPSLA '09*, (New York, NY, USA), pp. 619–626, ACM, 2009.

[40] E. M. Maximilien, A. Ranabahu, R. Engehausen, and L. C. Anderson, "IBM Altocumulus: a cross-cloud middleware and platform," in *Proceedings of OOPSLA '09*, (New York, NY, USA), pp. 805–806, ACM, 2009.

[41] M. Coppola, Y. Jegou, B. Matthews, C. Morin, L. Prieto, O. Sanchez, E. Yang, and H. Yu, "Virtual organization support within a grid-wide operating system," *Internet Computing, IEEE*, vol. 12, pp. 20–28, March-April 2008.

[42] T. DeWitt, T. Gross, B. Lowekamp, N. Miller, P. Steenkiste, J. Subhlok, and D. Sutherland, "Remos: A resource monitoring system for network-aware applications," Tech. Rep. CMU-CS-97-194, School Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1997.

[43] P. Chandra, A. Fisher, C. Kosak, T. Ng, P. Steenkiste, E. Takahashi, and H. Zhang, "Darwin: Customizable resource management for value-added network services," *IEEE NETWORK*, vol. 15, pp. 22–35, 2001.

[44] R. Van Renesse, K. Birman, D. Dumitriu, and W. Vogel, "Scalable management and data mining using Astrolabe," in *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS'01)*, 2001.

[45] J. Albrecht, D. Oppenheimer, D. Patterson, and A. Vahdat, "Design and Implementation Tradeoffs for Wide-Area Resource Discovery," *ACM Transactions on Internet Technology (TOIT)*, vol. 8, September 2008.

[46] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Survey and Tutorial*, vol. 7, pp. 72–93, March 2004.

[47] SETI@Home. [Online] http://setiathome.berkeley.edu.

[48] BOINC. [Online] http://boinc.berkeley.edu.

[49] G. Fox, D. Gannon, S.-H. Ko, S. Lee, S. Pallickara, M. Pierce, X. Qiu, X. Rao, A. Uyar, M. Wang, and W. Wu, *Peer-to-Peer Grids*. John Wiley and Sons Ltd, 2003.

[50] I. Foster and A. Iamnitchi, "On death, taxes, and the convergence of peer-to-peer and grid computing," in *In 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, pp. 118–128, 2003.

[51] N. Drost, R. van Nieuwpoort, and H. Bal, "Simple locality-aware co-allocation in peer-to-peer supercomputing," in *Proceedings of the 6th IEEE/ACM CCGrid Symposium*, 2006.

[52] J. Ritter, "Why gnutella can't scale. No, really." [Online] http://www.darkridge.com/ jpr5/doc/gnutella.html., Feb 2001.

[53] J. Albrecht, R. Braud, D. Dao, N. Topilski, C. Tuttle, A. C. Snoeren, and A. Vahdat, "Remote Control: Distributed Application Configuration, Management, and Visualization with Plush," in *Proceedings of the Twenty-first USENIX LISA Conference*, November 2007.