CloudViews: Communal Data Sharing in Public Clouds

Roxana Geambasu Steven D. Gribble Henry M. Levy
University of Washington

1 Introduction

Web services are undergoing an exciting transition from in-house data centers to public clouds. Attracted by automatic scalability and extremely low compute, storage, and management costs, Web services are increasingly opting for public cloud deployment over traditional in-house datacenters. For example, Amazon's S3 provides storage and backup services for numerous applications [12, 17], a number of mature services have recently migrated to Amazon EC2 [18], and many startups are adopting the cloud as their sole viable solution to achieve scale [22]. While predictions regarding cloud computing vary, most of the community agrees that public clouds will continue to grow in the number and importance of their tenants [10].

This paper focuses on a new opportunity introduced by the cloud environment: specifically, rich data sharing among independent Web services that are *co-located* within the same cloud. In the future, we expect that a small number of giant-scale shared clouds – such as Amazon AWS, Google AppEngine, or Microsoft Azure – will result in an unprecedented environment where thousands of independent and mutually distrustful Web services share the same runtime environment, storage system, and cloud infrastructure. One could even imagine that most of the Web will someday be served from a handful of giant-scale clouds. What will that new shared-cloud environment look like? What are the opportunities and challenges created by this integration and consolidation? While challenges raised by the multi-tenant environment, such as isolation, security, and privacy, have received significant recent attention [20], we believe that identifying untapped opportunities is equally important, as it enables innovation and advancement in the new shared-cloud world.

In particular, we argue that co-location creates an auspicious environment for Web service composition, which in turn spawns immense opportunities for simplifying Web service construction. Three key technological features differentiate the shared-cloud world from the traditional in-house datacenter and enable these opportunities: (1) free, efficient, and plentiful network bandwidth that supports tighter and larger-scale Web service integration than is possible over wide-area networks; (2) a shared storage system that can provide powerful abstractions for convenient, efficient, and large-scale inter-service data sharing; and (3) the potential for a rich runtime ecosystem consisting of many "utility" Web services that act as building blocks for other services and facilitate their implementation greatly.

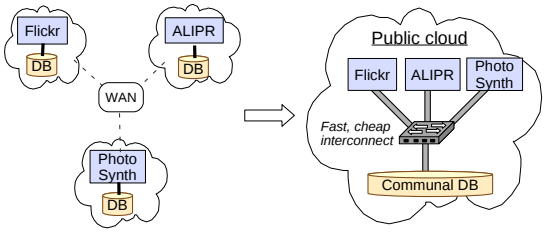
This paper is divided into two parts. The next section describes the opportunities for collaborative Web services in the new cloud world. It also examines the three technological features described above and both the potentials and the challenges that they bring. Section 3 presents CloudViews, a cloud storage system we are developing to facilitate collaboration through protected inter-service data sharing. CloudViews is one example of the kinds of functions public clouds must offer to facilitate Web service development. Overall, our goal is to provide a high-level glimpse of the potential impact of co-location on low-level cloud infrastructure and service development.

2 Opportunities in Public Clouds

The vision of facilitating Web service construction through seamless integration of existing Web services has existed since the dawn of Web 2.0. Similarly, Web hosting has existed for years. What is new is the intersection of three key technological features of the public cloud environment: a faster and cheaper network fabric; a common storage infrastructure; and the potential for a richer-than-ever runtime environment. After a brief background on composition, we analyze each feature and the new opportunities and challenges that it creates. Figure 2 provides a summary of these technology trends.

2.1 Web Service Composition Background

Web service composition is the defining feature of Web 2.0. The ability to combine existing Web services into new composite services (or *mashups*) has proven momentous for advancing the Web and deriving new value from it. The simplest mashup example is a map-based mashup like toEat.com, which overlays restaurant information on a Google map to provide the user with a convenient view of close-by restaurants. Such a mashup is termed a *client-side mashup*, as the contents from



(a) In-house DC environment (b) Shared-cloud environment

Figure 1: **Technological advantages of public clouds.** Three Web services seeking interaction (a) in the traditional one-datacenter-per-server model and (b) in the public-cloud environment. In public clouds, services enjoy a more efficient and free inter-web-service networking fabric, a common large-scale storage infrastructure ("Communal DB"), and a rich computing ecosystem (not shown).

the composing services are aggregated in the client's browser. *Server-side* mashups also exist. In their case, the mashup site aggregates information from other services and its own database and returns the result to the client. Server-side mashups are typically data-driven. Popular examples of server-side mashups include Facebook applications such as iLike.com and comparison shopping sites such as PriceGrabber.com. The focus of this paper is on server-side, data-centric mashups.

2.2 Technological Shifts, Opportunities, and Challenges

Figure 1 compares the traditional in-house datacenter environment to public, shared clouds, pointing out the technological advances brought on by the cloud. The figure shows three existing photo-related Web services: Flickr (photo sharing service storing over 3 billion photos [14]), Photosynth [15] (3D scene re-composition from photos), and ALIPR [1] (automatic tagging of photos using image recognition). All of these services currently operate inside their own private datacenters (Figure 1(a)). Composition is extremely valuable for these services. For example, Photosynth leverages Flickr to create 3D models from photos of tourist attractions. Similarly, Flickr could benefit from ALIPR, which could automatically add tags to its photos.

2.2.1 The Free and Fast Network

In the traditional in-house datacenter model (Figure 1(a)), each service runs inside its own datacenter and stores its data within its own storage system (labeled DB in the figure). Communications among services are done over high-delay, low-bandwidth, and expensive WANs (denoted by dashed lines). These impose severe constraints on the scale, frequency, and latency achievable by Web service compositions. Upon their migration to a public cloud (Figure 1(b)), the services become residents of the same cloud environment. Consequently, services now enjoy a largely free, fast, and high-bandwidth inter-web-service network. For example, 10Gbps switches are common in today's datacenters [11]. Additionally, much work is being done to ensure uniform bandwidth and latency between any two machines in a datacenter [11].

Technological change	Opportunity	Challenges
Efficient and cheap networks	Tight, large-scale integration.	Datacenter-level co-location.
Common storage infrastructure	Convenient composition through storage.	Sharing abstraction, protection, perf. isolation, billing.
Rich runtime ecosystem	Utility services provide building blocks.	Business models and incentives, common formats.

Figure 2: **Technological advantages, opportunities, and challenges in the public-cloud environment.**

We argue that this free, high-quality networking fabric will enable a new generation of large-scale, finely-intertwined Web service compositions that was infeasible over WANs. For example, in today's world, leveraging ALIPR to tag Flickr's 3B photos or Facebook's 10B photos would be utterly slow and expensive. Inside the cloud, however, these transfers can be absolutely free and significantly faster on the parallel high-speed network.

Challenges. To take full advantage of fast cloud networks, we must ensure that composing services co-reside in the same datacenter. Co-location is thus an important problem (first row of Figure 2). We believe that the high code mobility enabled by today's virtualized clouds, coupled with years of research in location-aware schedulers [16], will allow the mitigation of this challenge. Moreover, an extremely coarse-grained co-location system, which schedules composing services within the same region[1], is enough to keep transfer costs zero.

2.2.2 The Common Storage Infrastructure

In the traditional in-house datacenter, each service stores its data within its own database, however services in public clouds store all of their data in the cloud's common storage service (*e.g.*, S3, Bigtable, etc.). Apart from the obvious benefit of not having to implement and manage an in-house, scalable, fault-tolerant storage system, the common storage infrastructure gives way to a second valuable opportunity. Specifically, the storage system can simplify data-centric Web service compositions by handling many sharing-related issues, such as protection, performance isolation, and billing. In the in-house datacenter environment, all of these difficult functions require in-house solutions.

As an example, our CloudViews system (described in Section 3) allows Flickr to conveniently grant access to all or a subset of its photos, *e.g.*, to Photosynth. From that moment on, the storage infrastructure handles all of Photosynth's requests, ensuring that it can only access shared photos, that no sudden spike in its requests will disrupt Flickr's operation, and that Photosynth pays for the con-

[1] We refer here to Amazon's continent-wide regions, inside of which traffic is free.

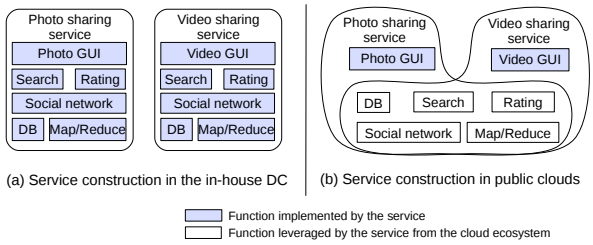


Figure 3: **Simplifying Web service construction in public clouds.** Web service construction in (a) in-house and (b) public-cloud environments. In the in-house datacenter, services cannot outsource their modules to other services over the WAN, leading to many services implementing entire software stacks redundantly. In public clouds, a Web service should be able to seamlessly leverage other services living in the cloud's ecosystem (e.g., media indexing and search, social networking data, etc.).

sumed data (assuming a billing contract was established). This way, Flickr can focus on its specific functions, ignoring all the complexity typically associated with maintaining an API in support of this sharing.

Several Amazon S3 offerings have similar flavors as CloudViews. For example, with "Requester Pays" [3], an S3 user can "rent" some of his data and have whomever accesses that data pay a fee for it; with the "Authenticated query" feature [2], a user can provide temporary direct access to a bucket to another user. We believe that these features are rudiments of the properties that a cloud storage system must provide in support of mashups.

Challenges. We identify several requirements of the cloud storage that are essential to facilitating Web service compositions, but that receive inadequate support in today's clouds:

- *A flexible sharing abstraction.* Picking the appropriate abstraction is key to stimulating composition. The abstraction must allow services to efficiently share data at any granularity, from a few objects (*e.g.*, the list of friends for a user on Facebook) to terabytes and billions of objects (*e.g.*, Flickr shares all of its photos with ALIPR or Photosynth). Additionally, the sharing abstraction must provide some degree of logical independence from the specific formats and data layouts. Today's cloud sharing abstractions (*e.g.*, buckets in S3) fall short in this latter requirement.

- *Naming and protection.* Naming and protection are decisive yet challenging aspects of any shared system. The protection mechanism must scale to a huge number of sharers and sharees (tens of thousands). None of today's cloud storage systems are equipped to handle such sharing. For example, Bigtable has a limited protection notion based on coarse, per-column-family ACLs. Similarly, S3's ACL mechanism limits the number of sharees for each bucket to 100.

- *Resource allocation.* Resource allocation is another key property of a shared system. Broadly speaking, resource allocation in today's cloud storage systems is done by partitioning data by service and assigning partitions evenly among storage servers [5, 9]. This works well, provided no sharing is involved. However, when two services are operating directly on each other's data, how do we prevent a workload spike in one from disrupting the other's performance? Our preliminary experiments on Hadoop's Hbase indicate that data sharing does cause significant performance interference due to the absence of appropriate resource allocation.

These requirements represent important but hard challenges that the storage system must overcome. Section 3 presents the design of a system we are working on, called CloudViews, which attempts to address these issues.

2.2.3 The Rich Runtime Ecosystem

The previous two technological features of public clouds – a fast network and a common storage – support convenient, large-scale, and efficient Web service compositions. This last feature paves the way for our ultimate goal: simplifying the development of Web services. We argue that a new generation of utility services (or building blocks for other services) is likely to emerge and flourish in public clouds. Evidence of this trend is visible in most clouds, where the cloud providers and sometimes third-parties already provide some utility services in support of consumer services (*e.g.*, RightScale, Amazon's S3 and Map/Reduce, and Google's Bigtable).

Leading this trend much further, we foresee the emergence of a rich cloud ecosystem, consisting of tens of thousands of utility services, each of which offers some function that other services can rent and use. Figure 3 uses an example to illustrate the enormous opportunities for simplifying Web service development hiding behind this concept. In this example, we use two services – one for sharing photos and the other one for sharing videos – and identify common functions between them. We then compare the opportunities for code reuse in the in-house datacenter model versus in public clouds.

In-house datacenters can tolerate only limited outsourcing of functions to one another over the WAN. For example, suppose the photo service opened an API to its general media search engine, allowing other services to plug in their own data and search for it. Would the video service be able to use that API? As the entire video collection is stored inside its local database and is separated from the search engine by thin WAN links, this would be prohibitively expensive. Thus, services running in the traditional in-house environment have no choice but to implement entire software stacks. In fact, today's Web as a whole is architected as a stovepiped system. This

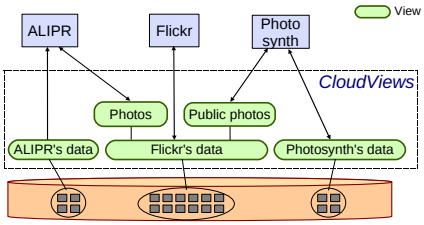


Figure 4: **Data sharing in CloudViews.** Services access their data using views; for example, Flickr uses the view denoted "Flickr's data," which contains all of its data. Services can share restricted views of their data with other services. Flickr has shared with ALIPR the view denoted "Photos," which contains all of its photos, but not other data (*e.g.*, user information). Flickr has also shared with Photosynth the view "Public photos," containing only the public photos.

may well be the reason why building even an intuitively simple service like Twitter is challenging at scale [8]. We now have a chance to correct this situation in public clouds, where a Web service developer can seamlessly leverage utility services from the cloud ecosystem.

Challenges. The success of the cloud ecosystem relies on two issues. First, collaboration is highly dependent on common or at least open data formats. For example, if Flickr stored its photos in a private format, then ALIPR would not be able to understand and tag them. Second, the emergence of utility services powering the cloud ecosystem is contingent upon the presence of incentives for providing functions "behind the scenes." Fortunately, business models are receiving significant attention from the cloud computing community [6], and unified APIs and common data formats are seeing ever-increasing adoption (*e.g.*, microformats, OpenSocial, etc.).

2.3 Summary

This section described three technological shifts that arise in public clouds: faster, cheaper networks, a common storage infrastructure, and the potential for a rich ecosystem of utility services. These changes give rise to a set of new opportunities for convenient, large-scale Web service compositions within public clouds and, ultimately, to much simplified creation of Web services. In the end, our hope is that creating a large-scale Web service can be made as easy as building some service-specific functionality (*e.g.*, the GUI), picking out some building blocks from the cloud's ecosystem, and combining those with a few "glue" scripts. The next section presents CloudViews, a system we are currently building as a first step in this direction.

3 CloudViews

We are currently designing and implementing CloudViews, a storage system that addresses several of the challenges identified in Section 2. In particular, CloudViews aims at facilitating data-centric Web service compositions by enhancing communal cloud storage systems with a database-style view abstraction for flexible, protected, efficient, and performance-isolated data sharing. We are currently building a prototype of CloudViews on top of Hadoop HBase.

The view abstraction. CloudViews facilitates composition by allowing services to create and share views over the common storage infrastructure. Figure 4 illustrates how services access and share views in CloudViews using the Flickr–ALIPR–Photosynth example. Specifically, Flickr accesses its data stored within the communal storage system via its base view, called "Flickr's data." The view is a query on the underlying database selecting all of the data whose owner is Flickr (*e.g.*, photos, tags, user information, and other data). Similarly, ALIPR and Photosynth access their data using their own views. Views protect the data, *i.e.*, the services cannot access each other's data by default.

However, services can share restricted views of their data with other services. For example, in the figure, Flickr has shared with ALIPR the view denoted "Photos," which contains its photo information, but excludes any other data, such as Flickr's user database. ALIPR uses this view to retrieve Flickr's photos and update their tags in the database. Finally, services can share different views with different services. For example, Flickr has shared with Photosynth the view "Public photos," which contains only the public photos.

Views have long been recognized by the database community to enable flexible, scalable sharing and logical data independence. CloudViews leverages the view notion, applies it onto the cloud environment from which they are absent, and enhances it with additional properties, such as a scalable protection scheme and a resource management mechanism.

Protection. Protection in CloudViews is a primary goal. A key challenge of the protection mechanism is scalability. CloudViews aims at handling tens of thousands of Web services sharing data through a mechanism called *signed views*. A signed view is an unforgeable token consisting of the view's query plus some metadata (including expiration, billing, and resource management information). Unforgeability is achieved through cryptographic signatures. A signed view is self-certifying, *i.e.*, a service's mere possession of it is proof that it has the right to access the data specified by that view. A signed view is also self-sufficient, *i.e.*, it carries all of the information required to be executed, tracked, billed for, etc. Intuitively, self-certification and self-sufficiency eliminate the need for centralized protection state, making protection based on signed views amenable to large scales.

Signed views are similar in flavor to S3's "authenticated queries" [2] and, more generally, to capabilities [13, 21]. Signed views differ from those mechanisms with additional properties, which we discuss next.

Resource allocation. In a world where services can access each other's data directly from the database, resource management is vital. In CloudViews, views are the unit of scheduling. At a conceptual level, queries atop each view are placed in a separate queue upon their entrance into the system. Queues are then scheduled for execution fairly. We are currently investigating the use of a distributed lottery scheduling scheme [19] to achieve fairness.

Notifications. Update notifications are key when large amounts of data are being shared. As one example, the ALIPR or Photosynth would greatly benefit from being able to receive notifications whenever new photos are uploaded into Flickr. This would save them from doing inefficient scans of Flickr's giant photo set. As another example, imagine a future in which a significant portion of the Web resides in a cloud and a search engine service is co-located in the same cloud. What would that cloud search engine look like? One possibility is that Web sites share views of their public data with the search engine, which would use those views to access their pages. A tremendously valuable property of those views would be to allow the search engine to register for view update notifications. Although we are currently targeting smaller scales than those imagined in this example, we are considering leveraging techniques from pub/sub systems [4, 7] to enable update notifications for our views.

4 Conclusions

The transition to public clouds is consolidating a large number of Web services into a few giant-scale clouds. Consolidation creates an auspicious environment for Web service collaboration. The cheap, high-bandwidth, and low-latency networking fabric, the common storage infrastructure, and the potential for a rich computing ecosystem enable immense opportunities for sharing and composition in public clouds. This paper identified those opportunities, as well as the challenges associated with them. We argued that, through carefully crafted abstractions, the cloud can stimulate a new generation of tightly-coupled, large-scale Web service compositions. We presented CloudViews, a system we are currently designing to support convenient, efficient, and performance-isolated data sharing in public clouds. By doing so, we make a first step toward a new and simplified Web service construction model: seamlessly composing Web services living inside the cloud's ecosystem.

References

[1] ALIPR. ALIPR: Automatic Photo Tagging and Visual Image Search. http://alipr.com/, 2009.

[2] Amazon.com. Query string authentication. http://docs.amazonwebservices.com/AmazonS3/latest/index.html?RESTAuthentication.html, 2006.

[3] Amazon.com. Requester pays buckets. http://docs.amazonwebservices.com/AmazonS3/latest/index.html?RequesterPaysBuckets.html, 2009.

[4] L. F. Cabrera, M. B. Jones, and M. Theimer. Herald: Achieving a global event notification service. In *Proc. of HotOS*, 2001.

[5] 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 *Proc. of OSDI*, 2006.

[6] T. Claburn. Amazon Web Services Intros Business Model For Content Owners. http://www.intelligententerprise.com/showArticle.jhtml?articleID=212701060, 2009.

[7] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. PNUTS: Yahoo!'s hosted data serving platform. In *Proc. of VLDB*, 2008.

[8] N. Cubrilovic. Twitter at scale: Will it work? http://www.techcrunch.com/2008/05/22/twitter-at-scale-will-it-work/, 2008.

[9] 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. In *Proc. of SOSP*, 2007.

[10] J. Foley. 10 cloud computing predictions for 2009. http://www.informationweek.com/news/services/saas/showArticle.jhtml?articleID=212901104&pgno=1&queryText=&isPrev=, 2009.

[11] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Towards a next generation data center architecture: scalability and commoditization. In *Proc. of PRESTO*, 2008.

[12] Jungle Tools. JungleDisk – reliable storage on Amazon S3. http://www.jungledisk.com/, 2007.

[13] H. Levy. *Capability-Based Computer Systems*. Digital Press, 1984.

[14] Michael Arrington. Three Billion Photos At Flickr. http://www.techcrunch.com/2008/11/03/three-billion-photos-at-flickr/, 2008.

[15] Microsoft. Photosynth. http://livelabs.com/photosynth/, 2009.

[16] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-aware request distribution in cluster-based network servers. In *Proc. of ASPLOS*, 1999.

[17] SmugMug. Smugmug – the ultimate in photo sharing. http://www.smugmug.com/, 2005.

[18] Techout. Techout index page. http://www.techout.com, 2005.

[19] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Proc. of OSDI*, 1994.

[20] R. Westervelt. Cloud computing group to tackle security concerns. http://searchsecurity.techtarget.com/news/article/0,289142,sid14_gci1352540,00.html, 2009.

[21] W. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack. HYDRA: The kernel of a multiprocessor operating system. *Comm. of the ACM*, 17(6), June 1974.

[22] Xignite. Xignite page. http://www.xignite.com/, 2009.