netSCALE Technologies, Inc.

281 State Hwy 79, Morganville, NJ 07751-1157, Tel. 732-970-1441, http://www.net-scale.com

Lessons from Cloud Computing for End-To-End Security

WHITE PAPER

Revision: 206

Version: 1.0

July 8, 2009

Table of Contents

1. Summary

While the final goal for the government as well as for the private sector is a "a network-centric data strategy that streamlines data discovery and sharing" [1], the implementation of this vision is not a straight forward task. Neither the industry nor academia has provided a comprehensive and practical solution for this need. Concepts of grid and cloud computing have emerged in the last ten years as candidates, but as it will be shown in this document, their current implementations present significant flaws which detriment any serious adoption. Security gaps and lack of interoperability are among the most urgent issues which require attention [2].

This document is organized as follows. The first chapter introduces the concepts of grid and cloud computing, and compares their implementations. The following chapters explore the main implementations such as Google and Amazon, as well as commercial and academic alternatives. Conclusions are presented at the end.

2. Introduction

Economies of scale, anywhere access, new forms of collaboration and extended data storage are recognized driving forces for grid and cloud computing technologies. Both technologies attempt to address the need for distributed, out-sourced, managed computing resources.

Grids have been understood as systems with non centralized control, that coordinate resources using standard, open and general-purpose protocols and interfaces to deliver a "better than the sum of the parts" quality of service [3]. As such, grid computing aims to enable "organizations to share computing and information resources across departments and organizational boundaries in a secure, highly efficient manner" [4].

Clouds have been recently defined as a large pool of dynamically reconfigurable virtualized resources, which are easy to use and access, exploiting a pay-per-use model with guaranties supported by customized service level agreements [5]. This translates as "a way [for IT] to increase capacity or add capabilities on the fly without investing in new infrastructure, training new personnel, or licensing new software" [6]. Variations of cloud computing include:

- Software as a Service (SaaS), where applications run in the network and deliver to thousands of users through the browser.
- Utility Computing, where memory, storage and computational capacity represent a virtualized resource pool available over the network.
- Web Services or Service Oriented Architecture (SOA), where APIs are available to to exploit functionality over the Internet.
- Platform as a Service (PaaS), where customized applications are deployed, run and delivered from a provider's infrastructure.
- Managed Service Providers (MSP), where applications such as e-mail virus scanning are managed externally to the IT organization as a service.

Both technologies have adopted distinctive positions. Grid computing has evolved as an open, physical rather than virtual, academic outgrown concept. While cloud computing has emerged as a commercial, virtual and user centric concept. Even though both technologies attempt to attain the same goal, their approach to resource sharing, virtualization, security, high level services, architecture, software dependencies, scalability, self-management, usability, standardization and quality of service is different, as seen in Table 1.

Table 1: Comparison Grid vs. Cloud Computing[1]

	Grid Computing	**Cloud Computing**
Resource Sharing	Fair distribution across organizations.	Isolation through virtualization.
Resource Heterogeneity	Aggregation of heterogeneous hardware and software.	
Virtualization	At the data and computing resources level, where applications and data can expand over multiple physical resources.	At the hardware and software level, where one physical resource can be used for several OS virtualizations.
Security	Through credential delegations	Through isolation (setting up closed virtual environments).
High Level Services	Several offerings such as metadata search and data transfer.	Lack of them.
Architecture and Software Dependencies	Given the abstract layer between physical resources and applications, applications running on the grid must comply with this interface following a service oriented architecture. In addition, applications require a predefined workflow of services, to coordinate the jobs performed and their location.	Through hardware virtualization applications are architecture agnostic and domain independent. Workflows are mostly not required for the type of on-demand deployments.
Scalability and Self-Management	Scalability is achieved by increasing number of nodes. Cross administrative domains have presented some difficulties for the self-management and automatic reconfiguration of grids.	Scalability is achieved by also resizing virtualized hardware resources. Self-management and automatic reconfiguration is easier since the cloud deployments are single domain.
Centralized Control	Decentralized	Centralized
Usability	Hard to manage.	Easy to manage from user's perspective.
Standardization	Standardization and interoperability under Globus project	Lack for clouds interoperability.
Payment Model	Rigid	Flexible
QoS Guarantees	Limited support, often best-effort only.	Limited support, focused on availability and uptime.

1 Adapted from [5]

Probably the model will evolve as Grid Computing for the Cloud Providers and as Cloud Computing for application developers and users. However, independent of this evolution, security issues must be addressed for the adoption of these approaches in mission critical operations. Data integrity, recovery, privacy, regulatory compliance and auditing are some of the areas under risk [7]. Moreover, the service agreements provided by the service providers do not offer enough assurances. For example, options to terminate accounts "for any reason" and "at any time", or the option of the provider to "pre-screen, review, flag, filter, modify, refuse, or remove any content from the service" [8].

3. Google

3.1. Description

Taking advantage of the Google infrastructure, Google offers the Google's App Engine, which allows users write simple applications in Python or Java and store the data in Google's database. This framework is not as mature as the Amazon offerings [8]. However, their underlying infrastructure that supports Google's massive search business is worth describing.

Google's architecture includes the following main elements [9]:

- A hardware infrastructure, which includes in-house rack design, PC-class motherboards, low-end storage and networking hardware, Linux OS and in-house software.
- A distributed and clustered systems infrastructure, which is composed of:
    - A scheduling system.
    - Google File System (GFS).
    - BigTable for data storage.
    - MapReduce for data processing.

A typical cluster at Google is shown below:

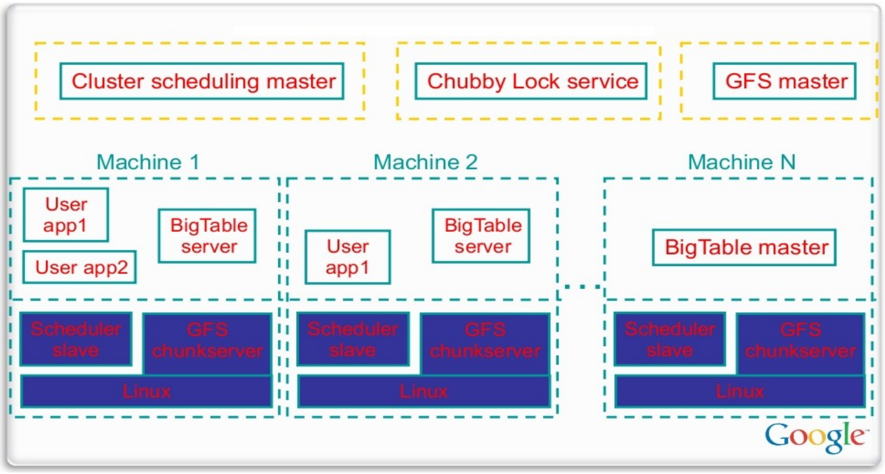

Figure 1: Typical Google cluster.

3.1.1. Google Cluster Architecture

Google's software architecture grew out of two basic principles: i) reliability depends on software rather than on server-class hardware, and ii) design is adjusted for best aggregate request throughput and not for peak server response (response times are managed by parallelizing individual requests) [10]. Services are replicated across different machines, and failures are detected and handled automatically. Machines are organized in clusters distributed worldwide. Load balancers at different levels distribute the work: a geography-based DNS load balancer assigns a cluster to user's requests, and a hardware-based load balancer in each cluster performs the local load balancing. The system has been structured so that most of the data accesses are read-only queries and the updates are relatively infrequently, which reduces the issues of consistency typically found in general-purpose databases. Finally, parallelism is highly exploited.

3.1.2. Google File System

The Google File System (GFS) is a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on commodity hardware, and it delivers high aggregate performance to a large number of clients [11].

The specific characteristics of GFS reside on its design assumptions:

- The system is built from many inexpensive hardware components that often fail. Therefore, monitoring, detection, tolerance and prompt recovery must be done on a regular basis.
- The file size distribution in the system is typically from 100MB to multi-GB. Optimization is targeted to efficiently handling these large files.
- Two kinds of reads constitute the primary workloads: i) large streaming reads and ii) small random reads. In either case, operations should be organized in such a way that they can steadily advance through a file. For example, short reads are often batched and sorted out before executed.
- Write operations are also part of the workloads, as large, sequential writes that append data to files. Once written, files are seldom modified. Small random writes are supported, but not optimized.
- System efficiently supports multiple clients simultaneously appending to the same file. Thus, append operation is atomic.
- Priority is set on high sustained bandwidth rather than low latency.

The usual file system operations are supported, and two more have been added to deal specifically with large distributed applications: *snapshot* and *record append*. Snapshot creates a copy of a file or a directory tree at low cost. Record append is an atomic append operation, where the client specifies only the data, but not the offset.

The main architectural components of GFS are: the GFS cluster and the GFS client. A GFS cluster consists of a single master and multiple chunkservers, and is accessed by many GFS clients. Files are divided into fixed-size chunks. Each chunk is uniquely identified, and its ID is assigned by the master at the time of creation. Chunkservers stores chunks locally and read or write chunk data. Each chunk is replicated on multiple chunk servers. Masters maintain all file system metadata and control system-wide activities. Only metadata is cached in the clients, which greatly simplifies the implementation. Finally, data consistency has been relaxed and GFS applications must accommodate for it.

3.1.3. BigTable

BigTable is a distributed storage system for managing structured data of the size of petabytes across thousands of servers [12]. Under different demands of data size and latency, BigTable provides a flexible and high-performance solution for many Google products such as web indexing and Google Earth.

BigTable resembles a database, but it does not support a full relational data model. This data model was driven by Google's applications, which needed to store large collections of web pages and information associated with them. In BigTable, data is indexed using row and column names and timestamps for versioning, and is treated as uninterpreted strings. Every read or write of data under a single row is atomic, regardless of the different columns read or written in the row. A row range (called tablet) is the unit of distribution and load balancing. Column names are grouped into sets (column families), which form the basic unit of access control. In addition, BigTable schema parameters let clients dynamically control whether to serve data out of memory or disk.

BigTable uses the distributed GFS to store log and data files. It depends on a cluster management system for scheduling jobs, managing resources on shared machines, dealing with machine failures, and monitoring status. BigTable data is stored in Google's SSTable file format, which provides a persistent, ordered immutable map from names to values. BigTable relies on Chubby [13], a highly-available and persistent distributed lock service, for several tasks: i) to ensure existence of at least one active master at any time, ii) to store the bootstrap location of BigTable data, iii) to discover tablet servers, iv) to store BigTable schema information, and v) to store access control lists. A Chubby service consists of five active replicas and uses the Paxos [14] consensus algorithm to keep their consistency in case of failure.

3.1.4. MapReduce

MapReduce is a programing model and an associated implementation for processing and generating large data sets [15]. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges these intermediate values associated with a intermediate key. Through this programming model, parallelism becomes natural.

3.2. Lessons Learned

After many years of experience dealing with large data sets, Google learned to operate with large, distributed and data-intensive applications. Early on, they realized that it wouldn't be economically feasible to rely on server-class hardware to deploy their applications, so they included in their design the concept of software reliability and fault tolerance. As valuable as these design principles are, Google was not set to address the issues of distributed storage systems over wide area networks. In this regard, highly variable bandwidth, untrusted participants, frequent reconfiguration, decentralized control and Byzantine fault tolerance are not Google's concerns, but they are for a global scale distributed and persistent storage system.

4. Amazon

4.1. Description

Amazon offers two services: Elastic Computing Cloud (EC2) and Simple Storage Service (S3). As the name specify, EC2 is for computational resources, while S3 is for storage resources.

S3 is supported by a large number of computers distributed across multiple data centers in the US and Europe, and it's expected to offer low data access latency, infinite data durability and 99.99% availability[2]. Since its launch, S3 has acquired a large user base including home users and small businesses to large enterprises [16]. It stores over 5 billion objects and handles over 900 million user request per day [17].

In S3 data is stored in buckets, at the top level, and within these buckets, in data objects. Buckets are equivalent to folders or containers, and have a unique global identifier or name. Every account can have up to 100 buckets. A bucket can store an unlimited number of data objects, each one containing a name, up to 5GB and metadata up to 4KB of user-specified name/value pairs. Data storage and data transfers are charged, and no metadata or content-based search capabilities are provided.

Clients authenticate using a public/private key scheme and a keyed-has message authentication code (HMAC). Keys are created at the moment of account creation and permanently stored at Amazon. Users can download the keys from Amazon's website, making Amazon's security model equivalent to a simple password system. Finally, access control is specified using access control lists (ACL) at the bucket and object levels for up to 100 identities.

S3 support three data access protocols: SOAP, REST and BitTorrent. This latter protocol allows for substantial bandwidth savings if multiple concurrent clients demand the same set of objects.

Lastly, data transfers between S3 and EC2 are not charged, allowing to run computations in EC2 and store and/or retrieve data from S3.

Even though the exact architecture of S3 is not disclosed, there are several references, which indicate that elements from Dynamo are used in S3 [18]. Dynamo is a highly available, scalable and distributed key-value storage system used by some of Amazon's core services [19]. To achieve scalability and availability, Dynamo uses techniques such as:

- Consistent hashing for data partitioning and replication.
- Object versioning to facilitate consistency.
- Quorum-like techniques and a decentralized replica synchronization protocol to maintain consistency among replicas during updates.
- Gossip-based technique and membership protocol to detect failure.
- Mostly automatic decentralized organization, which allows adding and removing storage nodes without any manual partitioning or redistribution.

It should be noticed that the success from Dynamo partly resides on the focused scopes of its requirements:

2 Amazon Web Services http://s3.amazonaws.com

- A query model with simple read and write operations to a data item that is uniquely identified by a key, and states are stored as binary objects also identified by unique keys. No operation span multiple data items and there is no need for relational schema. Dynamo target to store objects usually less than 1MB.
- Adequate compromise in ACID (Atomicity, Consistency, Isolation, Durability) principles. Dynamo relaxes consistency - "C" - to favor availability.
- Efficiency to constantly achieve the stringent latency and throughput requirements, while running on commodity hardware. The trade-offs are in performance, cost efficiency, availability and durability guarantees.
- Dynamo runs internally within Amazon's trusted environment, and thus eliminating any requirements in security associated to authentication and authorization.

4.2. Lessons Learned

Although Dynamo's architecture may not solve the greater problem of a global-scale trusted distributed storage system, its implementation in a real world setting, leaves us with several lessons to be learned:

- Focused requirements that lead to a simple architectural framework, which then can be configured to address variations of the use cases.
- Clear recognition of the trade-offs and use of them to achieve the best user experience for the use cases at hand.

From a technical point of view, Dynamo's implementation explored strategies in the following areas, which are important to keep in mind when designing a large storage system:

- Balancing performance and durability.
- Ensuring uniform load distribution.
- Identification of the cause of divergent versions and how to reconcile them.
- Client-driven vs. server-driven coordination.
- Balancing background vs. foreground tasks.

Finally, from a user's perspective, the following table summarizes an evaluation of S3 [20], which could be applicable to a global-scale distributed storage system.

Evaluation of S3 for Data Intensive Applications	
Data Durability	**Acceptable**, as indicated by non permanent data loss during a period of 12 months, with 10,000 files which sizes varied from 1B to 1GB, and over 137K requests done during that period.
Data Availability	**From EC2 was high**, as indicated by 5 PUT retries , 23 PUT timed-out, and 5 GET retries out of 107,556 tests. Considering 1MB or greater files, the throughput for writes was less than 10KB/s and only 0.03% was less than 100KB/s. **From S3 was 95.89%** without retry and 100% after the 5th retry for 15,456 downloads to Univ of South Florida.

Data Access Performance	Good availability, variable download time depending on location, file size and time of the day, and a fair use of the BitTorrent protocol. Specifically, from EC2 to S3 for single threads, downloading 1B, S3 can sustain a maximum of 100 transactions per sec – reflecting a dominance by S3 transaction overhead -, while downloading 100 MB, the maximum bandwidth was 21 MB/s. Uploading data to S3 was similar. From EC2 to S3 for concurrent threads, as the number of threads increased - to 6 - the per-thread bandwidth decreased but the aggregated bandwidth increased. For remote access, data bandwidth varies depending on geographical location and the time of the day.
Support for Security and Privacy	**Limited** because: ▪ Crude access control scheme, which does not scale well. ▪ Lack of support for fine-grained delegation. ▪ Implicit trust; no support for non-reputability (S3 must be trusted). ▪ Unlimited risk: S3 does not offer support for user-specified usage limits.
Cost	Given Amazon pricing structure, its usage may make sense for the costly tasks of high data availability and durability, while caching data at the edges of the system should be used to reduce the access volume and improve performance. Having said that, its costs for global-scale storage is prohibitive.
Usability	Given its security model and billing structure, the system is i**nadequate** to support complex, collaborative environments.

5. CleverSafe

5.1. Description

CleverSafe offers remote storage through a dispersed storage network (dsNet) [21]. dsNet uses Information Dispersal Algorithms (IDAs) [22] that allow for forward error correction and recovery.

IDAs separate data into unrecognizable slices, which are distributed to different storage devices. No single copy of the data resides in one location, and only a subset of the nodes needs to be available to fully retrieve the data. In CleverSafe dsNet, slices are distributed, via secure Internet connections, to storage locations in premises or throughout the world.

By coding and dispersing information, the reliability, security and efficiency of data storage in IDA systems can be vastly improved over traditional replication and parity-based systems.

Data distributed over ***p*** storage nodes, where any ***m*** of the nodes can fail, would require a storage overhead of ***p/(p-m)*** + ~12% associated with block file device. If p = 16 and m = 4, the storage overhead would be 1.33 + .12 = 1.45.

In comparison with replication systems, where data is mirrored on n storage devices, dsNet offers a more efficient approach to achieve robustness. In our previous example, n would be 5, and therefore the storage overhead is 5 compared to 1.45.

Similarly, dsNet would offer a better solution for storage than the parity-based systems, such as RAID disks. While parity-based systems are less wasteful than the replication systems (where n > 1), they are less reliable, since only one copy can fail without data loss.

IDA systems in combination with appropriate monitoring can repair data corrupted or destroyed based on inspection of the other uncorrupted slices, thus increasing the mean time between failure significantly.

5.1.1. Current Infrastructure

CleverSafe Dispersed Storage™ (dsNet) contains 3 layers: a Source computer, an Accesser and Slicestors for storage as shown in Figure 2[3]:

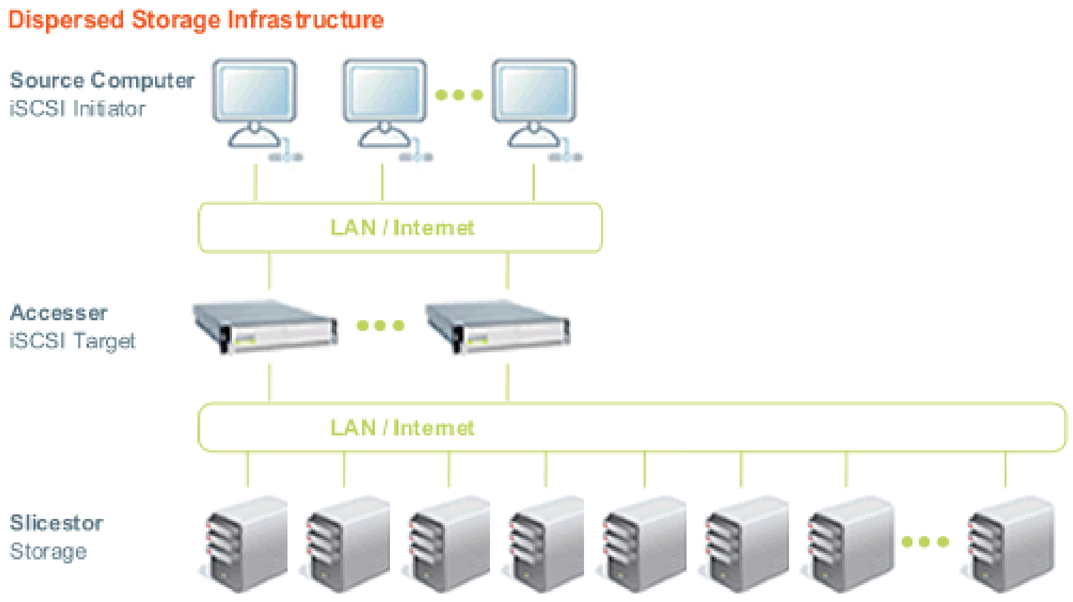

Figure 2: CleverSafe Dispersed Storage™ Infrastructure.

The dsNet storage network currently has been implemented as a block file device with an iSCSI Device interface. Additional interfaces such as NFS, CIFS and CAS have been announced.

For the Source device, the dsNet is equivalent to a file system, and the Accesser is mounted as a drive to the Source computer.

The Accesser software is responsible for storing and retrieving the slices to the Slicestors, as well as for providing the file system-type front end to the source computer. In addition the Accesser performs the encoding and decoding prior to the slices distribution.

The Slicestors store the slices produced by the Accessers.

3 From CleverSafe web-site: http://www.cleversafe.org/dispersed-storage/infrastructure

5.2. Lessons Learned

Since its foundation in 2004, CleverSafe has been recognized as an innovative technology company, which delivers cost effective solutions[4], and most certainly their strategy for storage makes sense and it seems that they have delivered. However the following points should be considered:

- Their focus is storage and not end-to-end security.
- Their technology improves traditional storage systems, but not an elastic storage systems which expands as the demand for storage increases.
- The storage nodes where the data is dispersed are set a-priori and not on-the-fly as needed for a systems where nodes can be added or removed constantly.

Within the constraints of the problem statement they tried to solve, it seems to work, but it does not solve the issue of a global-scale distributed, secure and persistent storage system.

6. OceanStore

6.1. Description

OceanStore is a global-scale distributed and persistent storage system, built over a peer-to-peer infrastructure of untrusted and changing components, where data constantly moves to improve performance and availability [23]. Data is protected through redundancy and cryptographic techniques. Servers are active components that participate in protocols for distributed consistency management, and even though some of them may crash without warning, it's assumed that a number of them are working correctly most of the time and can be trusted to carry out these protocols and therefore can guarantee quality of service. Performance is improved through allowing data caching anywhere, anytime – so called promiscuous caching. Moreover, monitoring of usage patterns not only permits adaptation to regional outages and denial of service attacks, but also enhances performance through proactive movement of data.

The persistent object is the fundamental unit. Each object is named by a globally unique identifier (GUID). Objects are replicated and stored in multiple servers. Since a given replica is independent of the server on which it resides at any given time, they are called floating replicas. A replica is located through: i) a fast, probabilistic algorithm applied near the requesting machine, and ii) a slower, deterministic algorithm. This location algorithm is called decentralized object location and routing (DOLR). Objects are modified through updates, which create a new version, and provide the mechanism for consistency. Objects exist in both active (the latest version with a handle for update, floating replica) and archival (permanent, read-only version) forms. Archival versions are encoded with an erasure code and spread over hundreds of servers. Write applications interact with the system through a number of sessions guaranties in the style of the Bayou system [24], dictating the level of consistency from extremely loose to ACID (Atomicity, Consistency, Isolation, Durability) semantics.

Pond [25] was the prototype of OceanStore, deployed on PlanetLab employing over 200 machines at 100 sites. Its implementation included most of the features of a complete system such as: location-independent routing, Byzantine update commit-

4 http://www.cleversafe.com/news-reviews/reviews

ment, push-base update of cached copies through an overlay multicast network, and continuous archiving to erasure-coded form. Compared to NFS, Pond outperformed it by a factor of 4.6 in read-intensive phases, but underperformed by 7.3 on write intensive phases, mainly limited by the speed of erasure coding and threshold signature generation.

OceanStore's initial architecture was quite ambitious and its implementation showed its limitations. However the issues it was designed to address are still very pressing to resolve.

6.2. Lessons Learned

After several years of implementing the concepts laid out in the original design from OceanStore, Kubiatowicz reflected on the lessons learned and pointed out four main problems [26]:

1. In practice, DOLR showed to be unstable, where nodes might be lost and never reintegrate, and routing state might become stale or be lost. This was caused by the complexity of the algorithms and, in hindsight, a wrong design paradigm. As a result a new, simpler, targeted design, Bamboo, showed to be more stable and with better bandwidth utilization [27], [28].
2. Write latencies resulted to be very large. For small files, signature dominates the timing, while for large files encoding becomes dominant. Realistic alternatives have not been found.
3. Efficiency for information retrieval turned out to be problematic. Small blocks of data spread out widely with every block of every file residing on a different set of servers. The suggested solution involved a two-level naming, placing the data in larger chunks, and accessing blocks by name within a chunk. This is realized in the Antiquity prototype [29].
4. The complexity of the algorithms combined with a reactive approach to fix implementation problems made it difficult to address the implementation issues. As a lesson learned, it's better to keep in mind the vision and address the issues as part of the design, instead of performing incremental improvements.

Kubiatowicz also mentioned other issues such as: i) high cost of archival repair, ii) resource management for denial of service or over utilization of storage nodes, iii) complexity with Byzantine agreement, distribution of key generation and access control at the level of the inner rings, iv) handling of low-bandwidth links and partial disconnection, and v) better replica placement.

Finally, secure and efficient sharing was not addressed. Thoughts on cryptographic hardware for revocation in the network do not seem realistic. The use of cryptographic links seems to offer a more practical approach.

These reflexions, even though they are not complete, provide a valuable guidance for the future of implementing and deploying such a system, and prove that after almost 10 years of research and experimentation, the problem of secure, persistent, of global-scale, distributed storage is still not solved.

7. Other Examples

7.1. Wikipedia

Since its origins in 2001 as a Perl CGI script running on a single server, Wikipedia has grown into a distributed platform with three data centers (Tampa, Amsterdam and Seoul) and 350 servers, managed by 6 persons. Performance and efficiency have been prioritized over high availability and security of operation [30]. The main components are:

- Linux for operating system (Fedora and Ubuntu)
- PowerDNS for geo-based request distribution
- Linux Virtual Service (LVS) for efficient load balancing i) in front of the Content Distribution Network (CDN), ii) between the CDN and Application, and iii) between Application and Search.
- Squid for content acceleration and distribution.
- lighttpd for static file serving
- Apache for application HTTP server
- PHP5 for core language
- MySQL for Database
- MediaWiki is the main application and it was developed in-house.
- Lucene and Mono for search engine.
- Memcached for various object caching.

The interaction of these components are shown in Figure 3 [31].

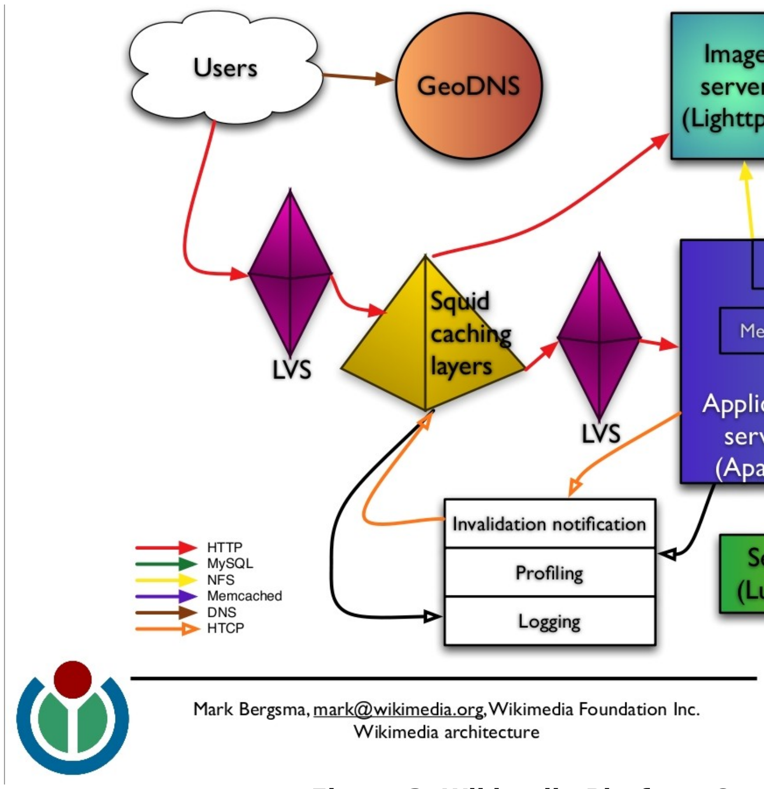

Figure 3: Wikipedia Platform Overview.

7.2. Akamai Technologies

Akamai Technologies offers a distributed content delivery system, which expands over 15,000 servers deployed on 1,200+ networks in 60+ countries [32]. By caching content at the Internet's edge, it reduces demand on content providers' infrastructure and provides faster service to users [33]. Akamai's infrastructure distributes resources by allocating more servers to the sites experiencing higher loads, while serving clients' requests from the nearest available server likely to have the requested content. Nearest is a function of network topology and dynamic link characteristics. Availability is a function of load and network bandwidth. Likely is a function of the servers carrying the content for each customer in the data center. Though mapping technology using border gateway protocol (BGP) information to determine network topology, the Akamai system directs requests to content servers, employing a dynamic, fault-tolerant DNS system. Server name resolution depends on service requested, server health, server load, network condition, client location and content requested.

This DNS-based load balancing system continuously monitors the state of services, and their servers and networks. This information is collected at the Network Operations Command Center (NOCC), from where corrective actions are dispatched if necessary. The system's end-to-end health is registered by agents simulating end-user downloading web content and measuring their failure rates and times.

Akamai servers deliver static and dynamic content as well as streaming of audio and video. Dynamic content is cached using Edge Side Includes technology, which let content providers break a dynamic page into pieces with independent cacheability properties. This reduced bandwidth by 95 to 99 % of the sites studied.

More recently, Akamai provided a distributed application platform (EdgeComputing) [34] where enterprise business web applications can be deployed. This involves virtualizing system resources and allocating them dynamically to different applications. To use EdgeComputing, a site developer must split the application into: i) an edge component (deployed in Akamai edge network), and ii) an origin component (deployed within the central data center as it traditionally would). Thus, the best suited split is to deploy the presentation components of an application onto the Akamai edge servers and to cache access to original data. In addition, the EdgeComputing platform provides caching mechanisms for persistent application data to minimize the interaction and load on the originating infrastructure on a request-by-request basis, and an IBM Java Cloudscape database management system to host infrequently changing, read-only data in an edge database.

7.3. CoDeeN Content Distribution Network

CoDeeN is an academic Content Distribution Network (CDN) built on top of PlanetLab to provide a fast and robust web content delivery service to its users. It consists of a network of high-performance proxy servers, which behave both as request redirectors and server surrogates [35].

In contrast to commercial CDNs which operate in a transparent way to end users, CoDeeN engages these users making the system demand-driven. Clients configure their browsers to use a CoDeeN node, which acts as a forward proxy. Cache misses are deterministically hashed and redirected to another CoDeeN proxy, which acts as a reverse-mode proxy, concentrating requests for a particular URL. Thus, fewer requests are forwarded to the original site.

Running on PlanetLab, CoDeeN does not operate on dedicated nodes with reliable resources, and its distributed in nature. Thus, CoDeeN includes significant node health monitoring facilities. Also, for its operation, CoDeeN added several security policies based on the client IP addresses, which involved rate limiting and privilege separation.

Recently HashCache has been developed as a configurable cache storage engine, which uses 6 to 20 times less memory than current network cache techniques and offers comparable or better performance [36]. The design criteria for HashCache includes: i) removing the in-memory index, ii) controlling collisions, iii) avoiding seeks for cache misses, iv) optimizing cache writes and v) prefetching cache reads. Currently HashCache is deployed at two locations in Africa, showing significant memory and CPU usage reduction compared to Squid.

7.4. PlanetLab

PlanetLab is a global platform for deploying and evaluating network services [37]. Since its origins in 2002, a set of 100 machines distributed over 40 sites, it has grown to 694 machines over 335 sites. This distributed system supports a series of large-scale network services such as content distribution, anycast, DHTs, robust DNS, large-file distribution and others.

PlanetLab operates through distributed virtualization, where each service runs in a slice (network-wide container) of the global resources. These slices are created through operations available on a centralized front-end, that communicates remotely with local node managers to establish and control virtual machines on each node. A set of these virtual machines defines a slice.

One of the design principles of PlanetLab included the concept of unbundled management, which consisted of managing the system through services deployed onto slices like any other service. While this concept fostered evolution and extensibility of its functionality, its implementation coped with the challenges of provisioning for the adequate trust mechanisms, the appropriate resource allocation, and slice isolation yet allowing some slices to manage others.

The trust model depends on the functionality of the centralized front-end called PlanetLab Central (PLC), where the node owners trust PLC to manage the behavior of the virtual machines running on them, and the users trust PLC to provide them access to the nodes to run their services. The security of this model is enforced by traditional authentication mechanisms from the nodes to the boot server running at PLC, and by off-line vetting mechanisms to ensure the identity of the users who can create accounts and then upload their private keys. PLC then installs these keys in the virtual machines it create for that user for him or her to connect via SSH. In addition, auditing services record all the network traffic flowing out of the nodes, which allow node owners to monitor and determine the origin of any unwanted traffic.

The resource allocation is decoupled from the slice creation. Thus, virtual machines are created irrespective of available resources, but during run time they are given a fair share of the obtainable resources. With this, there are slice creation services and brokerage services, which can be of the type market-based for buying and selling resources or scheduled-based.

In order to facilitate growth, interoperability of multiple independent PlanetLab-like systems should be possible. This was achieved by providing well-defined interfaces by which independent PLC invoke operations on each others.

In summary the experience from PlanetLab provided not only the design but also a stable and operational infrastructure for planet-large scale and distributed platform.

8. Conclusions

As pointed out by Nelson [38], the Internet is entering an exciting third phase, where it's becoming a platform for computing as well as communications. The overview of the different implementations described in this paper corroborate this assessment. However these island solutions or failed attempts do not solve the bigger issue of a global-scale distributed and persistent storage system.

As seen by Google, Amazon, Akamai and Wikipedia, they set to address very confined requirements for large distributed data-intense system, but untrusted participants or untrusted components or highly variable bandwidth or decentralized control or persistence beyond the life of their own companies were not considered.

OceanStore set its goal in this direction, but run into obstacles on the design and implementation side.

In summary, the main conclusions from this review are:

- For the design and implementation phase, the requirements and assumptions must be clear and as concise as possible.

- The framework for development must be well structured and robust. Failure detection and system recovery must be built into the system from the ground-up.
- A system has to be able to expand and be able to incorporate new technologies.
- Security, decentralized control and persistence over many generations are still a challenge.

9. Bibliography

[1] Lais S., DOD rolls out net-centric data strategy, 2009, http://defensesystems.-com/articles/2009/05/14/netcentric-data-is-linchpin-to-transformation.aspx?s=networking_190509

[2] Brodkin J., Security gaps in cloud computing demand more scrutiny, Forrester says., 2009, http://www.computerworlduk.com/management/it-business/services-sourcing/news/index.cfm?newsid=14720

[3] Foster I., What is the grid? - a three point checklist., 2002, http://www-fp.mc-s.anl.gov/~foster/Articles/WhatIsTheGrid.pdf

[4] Open Grid Forum, Understanding Grids, 2009, http://www.ogf.org/UnderstandingGrids/grid_understand.php

[5] Vaquero L.M., Rodero-Merino L., Caceres J., Lindner M.. A Break in the Clouds: Towards a Cloud Definition. ACM SIGCOMM Computer Communication Review. vol. 39. no. 1. 2009. http://ccr.sigcomm.org/online/files/p50-v39n1l-vaqueroA.pdf

[6] KnorrE., Gruman G., What cloud computing really means., 2008, http://www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031

[7] Brodkin J., Gartner: Seven cloud-computing security risks., 2008, http://www.infoworld.com/d/security-central/gartner-seven-cloud-computing-security-risks-853

[8] Wayner P., Cloud versus cloud: A guided tour of Amazon, Google, AppNexus and GoGrid, 2008, http://www.infoworld.com/d/cloud-computing/cloud-versus-cloud-guided-tour-amazon-google-appnexus-and-gogrid-122

[9] Dean J., Handling Large Datasets at Google: Current Systems and Future Directions, 2008, http://research.yahoo.com/files/6DeanGoogle.pdf

[10] Barroso L., Dean J., Hölzle U.. Web Search for a Planet: The Google Cluster Architecture . IEEE Micro. vol. 23. no. 2. 2003. http://www.computer.org/micro/mi2003/m2022.pdf

[11] Ghemawat S., Gobioff H., Leung S-T.. The Google File System. Proceedings of the Symposium on Operating Systems Principles. ACM SIGOPS. October 2003.. vol. . no. . 2003. http://research.google.com/archive/gfs-sosp2003.pdf

[12] Chang F., Dean J., Ghemawat S., Hsieh W., Wallach D., Burrows M., Chandra T., Fikes A., Gruber R.. Bigtable: A Distributed Storage System for Structured Data. Proceedings of OSDI 2006, Seattle, WA, 2004. . vol. . no. . 2004. http://research.google.com/archive/bigtable-osdi06.pdf

[13] Burrows M.. The Chubby lock service for loosely-coupled distributed systems.. Proceedings of the 7th OSDI, Novemeber 2006.. vol. . no. . 2006. http://labs.-google.com/papers/chubby-osdi06.pdf

[14] Chandra T., Griesemer R., Redstone J.. Plaxos made live - An engineering perspective.. Proceedings of PODC 2007.. vol. . no. . . http://labs.google.com/papers/paxos_made_live.pdf

[15] DeanJ., Ghemawat S.. MapReduce: Simplified Data Processing on Large Clusters. Proceedings of OSDI 2004, San Francisco, CA, 2004.. vol. . no. . 2004. http://research.google.com/archive/mapreduce-osdi04.pdf

[16] kirkpatrick M., Amazon releases early info on S3 storage use., 2006, http://www.techcrunch.com/2006/07/12/amazon-releases-early-info-on-s3-storage-use/

[17] Claburn T., In Web 2.0 Keynote, Jeff Bezos Touts Amazon's On-Demand Services, 2007, http://www.informationweek.com/news/internet/showArticle.jhtml?articleID=199100069

[18] Vogels W., Amazon's Dynamo, 2007, http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html

[19] DeCandia G., Hastorun D., Jampani M., Kakulapati G., Lakshman A., Pilchin A., Sivasubramanian S., Vosshall P., Vogels W.. Dynamo: Amazon's Highly Available Key-Value Store. Proceedings of the 21st ACM Symposium on Operating Systems Principles, Stevenson, WA, October 2007. vol. . no. . 2007. http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf

[20] Palankar M., Iamnitchi A., Ripeanu M., Garfinkel S.. Amazon S3 for Science Grids: a Viable Solution?. Data-Aware Distributed Computing Workshop (DADC), Boston, MA, June 2008.. vol. . no. . 2008. http://www.csee.usf.edu/~anda/papers/dadc108-palankar.pdf

[21] CleverSafe, Inc., CleverSafe Architecture, 2009, http://www.cleversafe.org/documentation/Cleversafe-Arch.pdf

[22] Rabin M.. Efficient dispersal of information for security, load balancing, and fault tolerance. Journal of the ACM. vol. 36. no. . 1989. http://citeseerx.ist.p-su.edu/viewdoc/summary?doi=10.1.1.116.8657

[23] Kubiatowicz J., Bindel D., Chen Y., Czerwinski S. et. al.. OceanStore: An Architecture for Global-Scale Persistent Storage. Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), November 2000. vol. . no. . 2000. http://oceanstore.cs.berkeley.edu/publications/papers/pdf/asplos00.pdf

[24] Petersen K., Spreitzer M, Terry D., Xerox PARC's Bayou Project, 1997, http://www2.parc.com/csl/projects/bayou/

[25] Rhea S., Eaton P., Geels D., Weatherspoon H., Zhao B., Kubiatowicz J.. Pond: the OceanStore Prototype. Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03), March 2003. vol. . no. . 2003. http://oceanstore.cs.berkeley.edu/publications/papers/pdf/fast2003-pond.pdf

[26] Kubiatowicz J., An OceanStore Retrospective, 2007, http://twisc.org:8080/0109-07a/default.htm

[27] Rhea S., Geels D., Roscoe T., Kubiatowicz J.. Handling Churn in a DHT. Proceedings of the USENIX Annual Technical Conference, June 2004. vol. . no. . 2004. http://oceanstore.cs.berkeley.edu/publications/papers/pdf/bamboo-usenix.pdf

[28] Rhea S., Godfrey B., Karp B., Kubiatowicz J., Ratnasamy S., Shenker S., Stoica I, Yu H.. OpenDHT: A Public DHT Service. SIGCOMM, August 2005. vol. . no. . 2005. dor09-arch-impl-review.odt

[29] Weatherspoon H., Eaton P., Chun B., Kubiatowicz J.. Antiquity: Exploiting a Secure Log for Wide-Area Distributed Storage. Proceedings of the 2nd ACM European Conference on Computer Systems (EuroSys '07), March 2007. vol. . no. . 2007. http://oceanstore.cs.berkeley.edu/publications/papers/pdf/antiquity06.pdf

[30] Mituzas D., Wikipedia: Site internals, configuration, code examples and management issues (the workbook)., 2007, http://dammit.lt/uc/workbook2007.pdf

[31] Bergsma M., Wikimedia Architecture, 2007, http://www.nedworks.org/~mark/presentations/san/Wikimedia%20architecture.pdf

[32] Sherman A., Lisiecki P., Berkheimer A., Wein J.. ACMS: Akamai Configuration Management System. Proceedings of NSDI 2005. Boston, MA, May 2005. vol. . no. . 2005. http://www.akamai.com/dl/technical_publications/ACMSTheAkamaiConfigurationManagementSystem.pdf

[33] Dilley J., Maggs B., Parikh J., Prokop H., Sitaraman R., Weihl B.. Globally Distributed Content Delivery. IEEE Internet Computing, September/October 2002. vol. . no. . 2002. http://www.akamai.com/dl/technical_publications/GloballyDistributedContentDelivery.pdf

[34] Davis A., Parikh J., Weihl W.. EdgeComputing: Extending Enterprise Applications to the Edge of the Internet. Proceedings of the World Wide Web Conference (WWW), New York, New York, 2004. vol. . no. . 2004. http://www.akamai.com/dl/technical_publications/EdgeComputingExtendingEnterpriseApplicationstotheEdgeoftheInternet.pdf

[35] Wang L., Park K., Pang R., Pai V., Peterson L.. Reliability and Security in the CoDeeN Content Distribution Network. Proceedings of the USENIX 2004 Annual Technical Conference Boston, Ma, June 2004. vol. . no. . 2004. http://nsg.cs.princeton.edu/publication/codeen_usenix_04.pdf

[36] Badam A., Park K., Pai V., Peterson L.. HashCache: Cache Storage for the Next Billion. The 6th USENIX Symposium on Networked Systems Design and Implementation, April 2009.. vol. . no. . 2009. http://www.cs.princeton.edu/~abadam/papers/hashcache.pdf

[37] Peterson L., Bavier A., Fiuczynski E, Miur S.. Experiences building PlanetLab. Proceedings of the 7th USENIX Symposium on Operating System Design and Implementation (OSDI '06) . vol. . no. . 2006. http://nsg.cs.princeton.edu/publication/experiences_osdi_06.pdf

[38] Nelson M.. Building an Open Cloud. Science. vol. 324. no. . 2009. http://www.sciencemag.org/cgi/reprint/324/5935/1656.pdf