An Overview of Cloud Computing

During the last several decades, dramatic advances in computing power, storage, and networking technology have allowed the human race to generate, process, and share increasing amounts of information in dramatically new ways. As new applications of computing technology are developed and introduced, these applications are often used in ways that their designers never envisioned. New applications, in turn, lead to new demands for even more powerful computing infrastructure.

To meet these computing-infrastructure demands, system designers are constantly looking for new system architectures and algorithms to process larger collections of data more quickly than is feasible with today's systems. It is now possible to assemble very large, powerful systems consisting of many small, inexpensive commodity components because computers have become smaller and less expensive, disk drive capacity continues to increase, and networks have gotten faster. Such systems tend to be much less costly than a single, faster machine with comparable capabilities.

Building systems from large numbers of commodity components leads to some significant challenges, however. Because many more computers can be put into a computer room today than was possible even a few years ago, electrical-power consumption, air-conditioning capacity, and equipment weight have all become important considerations for system designs. Software challenges also arise in this environment because writing software that can take full advantage of the aggregate computing power of many machines is far more difficult than writing software for a single, faster machine.

Recently, a number of commercial and academic organizations have built large systems from commodity computers, disks, and networks, and have created software to make this hardware easier to program and manage. These organizations have taken a variety of novel approaches to address the challenges outlined above. In some cases, these organizations have used their hardware and software to provide storage, computational, and data management services to their own internal users, or to provide these services to external customers for a fee. We refer to the hardware and software environment that implements this service-based environment as a *cloud-computing* environment. Because the term "cloud computing" is relatively new, there is not universal agreement on this definition. Some people use the terms *grid computing*, *utility computing*, or *application service providers* to describe the same storage, computation, and data-management ideas that constitute cloud computing.

Regardless of the exact definition used, numerous companies and research organizations are applying cloud-computing concepts to their business or research problems including Google, Amazon, Yahoo, and numerous universities. This article provides an overview of some of the most popular cloud-computing services and architectures in use today. We also describe potential applications for cloud computing and conclude by discussing areas for further research.

***Image credit**: NOAA Photo Library, NOAA Central Library; OAR/ERL/National Severe Storms Laboratory (NSSL)*

Image credit:
NASA/JPL-Caltech/Institut d'Astrophysique Spatia

Nomenclature

Before describing examples of cloud computing technology, we must first define a few related terms more precisely. A computing *cluster* consists of a collection of similar or identical machines that physically sit in the same computer room or building. Each machine in the cluster is a complete computer consisting of one or more CPUs, memory, disk drives, and network interfaces. The machines are networked together via one or more high-speed local-area networks. Another important characteristic of a cluster is that it's owned and operated by a single administrative entity such as a research center or a company. Finally, the software used to program and manage clusters should give users the illusion that they're interacting with a single large computer when in reality the cluster may consist of hundreds or thousands of individual machines. Clusters are typically used for scientific or commercial applications that can be parallelized. Since clusters can be built out of commodity components, they are often less expensive to construct and operate than supercomputers.

Although the term *grid* is sometimes used interchangeably with cluster, a computational grid takes a somewhat different approach to high-performance computing. A grid typically consists of a collection of heterogeneous machines that are geographically distributed. As with a cluster, each machine is a complete computer, and the machines are connected via high-speed networks. Because a grid is geographically distributed, some of the machines are connected via wide-area networks that may have less bandwidth and/or higher latency than machines sitting in the same computer room. Another important distinction between a grid and a cluster is that the machines that constitute a grid may not all be owned by the same administrative entity. Consequently, grids typically provide services to authenticate and authorize users to access resources on a remote set of machines on the same grid. Because researchers in the physical sciences often use grids to collect, process, and disseminate data, grid software provides services to perform bulk transfers of large files between sites. Since a computation may involve moving data between sites and performing different computations on the data, grids usually provide mechanisms for managing long-running jobs across all of the machines in the grid.

Grid computing and cluster computing are not mutually exclusive. Some high-performance computing systems combine some of the attributes of both. For example, the Globus Toolkit [1], a set of software tools that is currently the *de facto* standard for building grid-computing systems, provides mechanisms to manage clusters at different sites that are part of the same grid. As you'll see later in this article, many cloud-computing systems also share many of the same attributes as clusters and grids.

The Google Approach to Cloud Computing

Google is well known for its expanding list of services including their very popular search engine, email service, mapping services, and productivity applications. Underlying these applications is Google's internally developed cloud-based computing infrastructure. Google has published a series of papers in the computer-science research literature that demonstrate how they put together a small collection of good ideas to build a wide variety of high performance, scalable applications. In this section we describe what Google has built and how they use it.

Google Design Philosophy

Although Google's clouds are very high-performance computer systems, the company took a dramatically different approach to building them than what is commonly done today in the high-performance and enterprise-computing communities [2]. Rather than building a system from a moderate number of very high-performance computers, Google builds their cloud hardware as a cluster containing a much larger number of commodity computers. They assume that hardware will fail regularly and design their software to deal with that fact. Since Google is not using state-of-the-art hardware, they're also not using the most expensive hardware. Consequently, they can optimize their costs, power consumption, and space needs by making appropriate tradeoffs.

Another key aspect of Google's design philosophy is to optimize their system software for the specific applications they plan to build on it. In contrast, the designers of most system software (e.g. operating systems, compilers, and database management systems) try to provide a combination of good performance and scalability to a wide user base. Since it is not known how different applications will use system resources, the designer uses his or her best judgment and experience to build systems that provide good overall performance for the types of applications they expect to be run most often. Because Google is building both the system and application software, they know what their applications require and can focus their system-software design efforts on meeting exactly those requirements.

Google File System

Google's design philosophy is readily evident in the architecture of the Google File System (GFS) [3]. GFS serves as the foundation of Google's cloud software stack and is intended to resemble a Unix-like file system to its users. Unlike Unix or Windows file systems, GFS is optimized for storing very large files (> 1 GB) because Google's applications typically manipulate files of this size. One way that Google implements this optimization is by changing the smallest unit of allocated storage in the file system from the 8 KB block size typical of most Unix file systems to 64 MB. Using a 64 MB block size (Google calls these blocks *chunks*) results in much higher performance for large files at the expense of very inefficient storage utilization for files that are substantially smaller than 64 MB.

Another important design choice Google makes in GFS is to optimize the performance of the types of I/O access patterns they expect to see most frequently. A typical file system is designed to support both sequential and random reads and writes reasonably efficiently. Because Google's applications typically write a file sequentially once and then only read it, GFS is optimized to support append operations. While any portion of a file may be written in any order, this type of random write operation will be much slower than operations to append new data to the end of a file.

Since GFS is optimized for storing very large files, Google designed it so that the chunks that constitute a single GFS file do not need to reside on one disk as they would in a conventional Unix file system. Instead, GFS allocates chunks across all of the machines in the cloud. Doing chunk allocation in this manner also provides the architectural underpinnings for GFS fault tolerance. Each chunk can be replicated onto one or more machines in the cloud so that no files are lost if a single host or disk drive fails. Google states that they normally use a replication factor of three (each chunk stored on three different machines), and that they do not use the fault-tolerance techniques used in enterprise-class servers such as redundant arrays of inexpensive disks (RAID).

MapReduce

Built on top of GFS, Google's MapReduce framework is the heart of the computational model for their approach to cloud computing [4, 5]. The basic idea behind Google's computational model is that a software developer writes a program containing two simple functions—*map* and *reduce*—to process a collection of data. Google's underlying runtime system then divides the program into many small tasks, which are then run on hundreds or thousands of hosts in the cloud. The runtime system also ensures that the correct subset of data is delivered to each task.

The developer-written *map* function takes as its input a sequence of <key_{in}, $value_{in}$> tuples, performs some computation on these tuples, and produces as its output a sequence of <key_{out}, $value_{out}$> tuples. There does not necessarily need to be a one-to-one correspondence between input and output key/value tuples. Also, key_{in} does not necessarily equal key_{out} for a given key/value tuple.

The developer-written *reduce* function takes as its input a key and a set of values corresponding to that key. Thus, for all <key', $value_i$> tuples produced by the *map* function that have the same key key', the *reduce* function will be invoked once with key' and the set of all values $value_i$. It's important to note that if the tuple <key', $value_i$> is generated multiple times by the *map* function, $value_i$ will appear the same number of times in the set of values provided to the *reduce* function, i.e., duplicate values are *not* removed. Once invoked, the *reduce* function will perform some computation on the set of values and produce some output value that the runtime infrastructure will associate with the key that was supplied as input to *reduce*.

Example

To illustrate how MapReduce might be used to solve a real problem, consider the following hypothetical application. Suppose a software developer is asked to build a tool to search for words or phrases in a collection of thousands or millions of text documents. One common data structure that is useful for building this application is an *inverted index*. For every word *w* that appears in any of the documents, there will be a record in the inverted index that lists every document where *w* appears at least once. Once the inverted index is constructed, the search tool can rapidly identify where words appear in the document collection by searching the index rather than the entire collection.

Constructing the inverted index is straightforward using MapReduce. To do so, the developer would construct a *map* function that takes as its input a single <*document name, document contents*> tuple, parses the document, and outputs a list of <*word, document name*> tuples. For one invocation of *map*, key_{in} might be "`speech.txt`", and $value_{in}$ might be "`We choose to go to the moon in this decade and do the other things—not because they are easy, but because they are hard!`" If the map function were invoked with the key/value tuple shown above, map would parse the document by locating each word in the document using whitespace and punctuation, removing the punctuation, and normalizing the capitalization. For each word found, map would output a tuple <key_{out}, $value_{out}$> where key_{out} is a word and $value_{out}$ is the name of the document key_{in}. In this example, 25 <key_{out}, $value_{out}$> tuples would be output as follows

```
<we, speech.txt>
<choose, speech.txt>
<to, speech.txt>
<go, speech.txt>
<to, speech.txt>
…
<hard, speech.txt>
```

The *map* function would be invoked for each document in the collection. If the cloud has thousands of nodes, *map* could be processing thousands of documents in parallel. Accessing any document from any machine in the cloud via GFS makes *map* task scheduling easier, since a file doesn't need to be pre-positioned at the machine processing it.

The *reduce* function in this example is very easy to implement. The MapReduce infrastructure will aggregate all <*key,value*> tuples with the same key that were generated by all map functions, and send the aggregate to a single *reduce* function as <*key, list of values*>. This *reduce* invocation will iterate over all values and output the key and all values associated with it. In our example, suppose we have a second document whose name is "`song.txt`" and its contents are "`I'll see you on the dark side of the moon.`" The map functions processing `speech.txt` and `song.txt` will output the tuples `<moon,speech.txt>` and `<moon, song.txt>` respectively. *Reduce* will be invoked with the key/value tuple `<moon, list(speech.txt, song.txt)>` and will output something that looks like

```
moon: speech.txt song.txt
```

If the word moon appeared in other documents, those document names would also appear on this line.

Hundreds or thousands of *reduce* functions will process the output of different *map* functions the same way, once again taking advantage of the parallelism in the cloud. Because of the way the MapReduce infrastructure allocates work, the processing of a single word *w* will be done by only one *reduce* task. This design decision dramatically simplifies scheduling *reduce* tasks and improves fault tolerance since a failed *reduce* task can be restarted without affecting other *reduce* tasks and without doing unnecessary work.

After all reduce functions have finished processing tuples, there will be a collection of files containing one or more lines of *word: document list* mappings as shown above. By concatenating these files, we have constructed the inverted index.

Discussion

The MapReduce model is interesting from a number of perspectives. Decades of high-performance-computing experience has demonstrated the difficulty of writing software that takes full advantage of the processing power provided by a parallel or distributed system. Nevertheless, the MapReduce runtime system is able to automatically partition a computation and run the individual parts on many different machines, achieving substantial performance improvements. Part of the reason that this is possible goes back to Google's design philosophy. MapReduce is not a general-purpose parallel programming paradigm, but rather a special-purpose paradigm that is designed for problems that can be partitioned in such a way that there are very few dependencies between the output of one part of the computation and the input of another. Not all problems can be partitioned in this manner, but since many of Google's problems can, MapReduce is a very effective approach for them. By optimizing MapReduce performance on their cloud hardware, Google can amortize the costs and reap the benefits across many applications.

Another key benefit of the special-purpose nature of MapReduce is that it can be used to enhance the fault-tolerance of applications. Recall that Google builds its clouds as clusters of commodity hardware and designs its software to cope with hardware failures. Because a MapReduce computation can be partitioned into many independent parts, the MapReduce runtime system can restart one part of the computation if its underlying hardware fails. This restarting operation can be accomplished without affecting the rest of the computation, and without requiring additional expertise or programming effort on the part of the application developer. Google's approach to fault tolerance is in stark contrast to most approaches today that require substantial programmer effort and/or expensive hardware redundancy.

Performance

How well does MapReduce work in practice? Google's published results show impressive results for processing large data. In a paper published in 2004 [4], Google describes two experiments they ran on an 1800 machine cloud. Each machine had two 2 GHz Intel Xeon processors, 4 GB of memory, two 160 GB disks connected via an IDE interface, and gigabit Ethernet. Although not explicitly stated in the paper, we assume the processors were single-core processors.

Google's first experiment was to create and run a program they called grep, which was designed to search for a three-character pattern in 10^{10} 100-byte records—roughly one terabyte of data. Google states the pattern only appeared in 92,337 of the records. They claim their MapReduce implementation of `grep` was able to find the pattern in all of the data in approximately 150 seconds.

Their second experiment was modeled after the TeraSort benchmark [6] and was designed to sort a terabyte of data (10^{10} 100-byte records). The MapReduce sort implementation was able to sort this data in 891 seconds, which was faster than the fastest reported TeraSort benchmark at that time (1,057 seconds). This result is also interesting from a software-engineering perspective. Google claims that it took less than 50 lines of user code to implement this sort program. Because most of the details of task scheduling and file management are hidden in the MapReduce and GFS runtime system, it is possible to write such a simple program.

In a later paper on MapReduce [5], Google quantifies how many MapReduce jobs are run and how much data is processed in their production systems. They claim that in September 2007 their production MapReduce clouds processed over 400 *petabytes* of data in 2,217,000 jobs (one petabyte equals 1,000 terabytes, or 10^{15} bytes). These numbers are very impressive and demonstrate how useful MapReduce has become for Google's production computing environment.

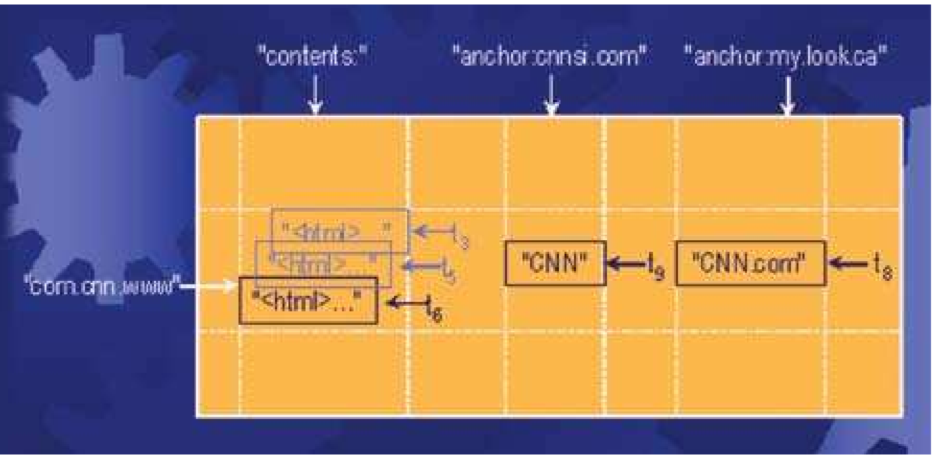
Bigtable

The last major component of Google's approach to cloud computing is their Bigtable data storage system [7]. In many respects, Bigtable superficially resembles a relational database management system (RDBMS). Both store data in tabular form with labeled rows and columns, and they allow data to be searched using the row name (and possibly the column name) as keys. Both also allow new data to be added, data in existing rows to be updated, and data to be deleted. Despite these similarities, there are also some important differences that we shall describe.

Consistent with their design philosophy, Google designed Bigtable so that it performs well in areas most relevant to Google's applications at the expense of generality and performance in less relevant areas. One area where Google paid particular attention was to design Bigtable so that it could store very large quantities of data. Bigtable horizontally partitions an individual table by grouping sets of rows into *tablets*, which are managed by different machines in a Google cloud. Since Bigtable stores table rows in lexicographic order, the developer can assign row keys in a manner that increases data locality. Google claims that Bigtable can store petabytes of data across thousands of servers [7]. In this same paper, Google reported that their Google Maps imagery data consumed 70 TB of space in Bigtable. They also reported that their Crawl Project (not described in the paper) data consumed 800 TB of space in a Bigtable table. The Google Maps and Crawl Project data are from 2006, so we assume that larger tables are being stored in Bigtable today.

Bigtable expands on the concepts of rows and columns from relational databases in a couple of ways. In Bigtable an individual data record, called a *cell*, is referenced by a row name, a column name, and a *timestamp*. The timestamp allows multiple versions of the same data to be stored in the same cell. Bigtable also extends the idea of a column by introducing the concept of *column families*. A column family is a set of related columns. Every column belongs to one column family, but a column family can have as many columns as desired. Google designed Bigtable so that creating a new column family is a relatively heavyweight operation while adding or deleting columns to/from a column family is a very lightweight operation. In contrast, changing the schema of a relational database is typically a time-consuming operation that requires shutting down the database during the change.

Figure 1: BIGTABLE example



Source*: Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE.* ***Bigtable****: A distributed storage system for structured data. ACM Trans. on Computer Systems. 2008 June; 26(2).*

Since Bigtable makes it easy to add columns to a column family, and columns can be sparsely populated, data can be stored in ways that would be inefficient or slow in a conventional relational database. Google describes an example Bigtable table (see Figure 1) that stores the contents of specific web pages as well as information about how other web pages link to them [7]. Each row in this table represents a specific web page. The row name is the URL of the page written in reverse order (`com.cnn.www` in the example) to improve locality of reference for related URLs stored in the table. Using the timestamp feature of Bigtable, multiple snapshots of the web page are stored at different points in time in the `contents`: column family. This table also has a column family called `anchor`:, which contains a column for every web page that points to the URL specified in the row name. The column name is the URL of the web page where the link was found. Every cell in the `anchor`: columns contains the "anchor" text whose underlying link is the URL of the row name. In the example, the `anchor`: column family has a column `cnnsi.com` because there is a link on the `cnnsi.com` web page to the `www.cnn.com` web page. The text "`CNN`" is stored in the corresponding cell, because that is the text on the `cnnsi.com` website that, when clicked in a web browser by a user, causes the browser to go to the page `www.cnn.com`. Thus, every time a new web page is discovered that points to `www.cnn.com`, a new column is added to the `anchor`: family. The name of this column is the new website's URL, and the cell referenced by this column and the `www.cnn.com` row contains the appropriate anchor text. Although it would be possible to build a relational database to store the data in this table, the cost of frequently adding new columns and the inefficiency of leaving most of the cells in them empty would make doing so impractical.

To implement the functionality in Bigtable that is most important to Google's applications, Google also omitted some functionality that would normally be found in an RDBMS. For example, Bigtable does not implement a join operation. Another database concept that Bigtable lacks is the transaction. While an individual row in a table can be updated atomically, operations across rows cannot be performed atomically or rolled back. For Google's applications, these omissions are not a serious impediment. Instead, they simplify the design of other parts of Bigtable. For example, deleting a column or a column family in a table becomes more complex if there is a running transaction that is manipulating rows with data in affected columns or column families.

Hadoop

After Google published the series of scientific papers describing their approach to cloud computing and their successful experiences using it [2, 5, 7], the approach generated a great deal of interest and enthusiasm outside of Google. Using insights gained from these papers, the open source software development community has created an implementation of Google's cloud approach called Hadoop [8]. Hadoop is part of the Apache open source project and contains an implementation of MapReduce and a GFS-like distributed file system called Hadoop Distributed File System (HDFS). HBase, a related Apache project, is an open source implementation of Bigtable [9].

As with GFS, HDFS stores large files across large clusters in sequences of blocks. Replication is also included within HDFS. Hadoop MapReduce and HDFS, as well as HBase, employ master/slave architectures very similar to the approach Google took in designing its corresponding systems. Unlike Google's systems, which are written in C++, Hadoop and HBase are written in Java.

Hadoop began as part of the Nutch open source search engine in 2002-2004 [10]. Nutch was built by a small group of developers working part-time. After the 2003-2004 publication of Google's papers, work began to add some of the Google concepts to Nutch, in order to address some of Nutch's scalability limitations. By 2006, Hadoop was split out of Nutch and became a separate effort. Since its launch, the community using and supporting Hadoop has grown substantially. Hadoop is currently in release 0.19.1, and there are now many Hadoop user groups and applications. Yahoo is currently a major supporter of Hadoop. To give a sense of how far Hadoop has matured in a short time, a 900-node Hadoop cluster at Yahoo set a new TeraSort benchmark in July 2008 [11]. This cloud was able to sort a terabyte of data in 209 seconds, beating the old record of 297 seconds. According to Yahoo, this is the first time a Java implementation or an open source TeraSort implementation has set a record for this benchmark.

HBase is newer and less mature than Hadoop. As with Hadoop, a Google scientific paper motivated its development. Initial work on HBase began in 2006 and was first released with Hadoop 0.15 in October 2007.

The Google approach to cloud computing is also gaining interest in academia. Google has joined forces with IBM to initiate university research to address large-scale computing problems across the Internet [12]. The current initiative is with Carnegie Mellon University, the University of Maryland, the Massachusetts Institute of Technology, Stanford University, and the University of California, Berkeley. Google and IBM are providing computing hardware to these universities to run the Hadoop and HBase software.

Professor Jimmy Lin, a faculty member in the College of Information Studies at the University of Maryland, College Park, brought together students from the Computational Linguistics and Information Processing Laboratory and University of Maryland Institute for Advanced Computer Studies to examine large-scale natural-language processing problems using Hadoop. This special seminar course was first held in the Spring 2008 with five projects. PhD students led small teams to explore open research problems and used MapReduce to assist with large scale parallelization issues associated with statistical machine translation, language modeling, identity resolution in email, biomedical network analysis, and image processing. Included in the course were various speakers who led discussions on cloud computing during the semester.

Amazon Approach to Cloud Computing

Amazon is best known for selling books online, but they are also actively investing in services that allow developers to take advantage of their computing technology. Amazon Web Services provide developers use of open APIs to access Amazon's vast infrastructure in a manner vaguely reminiscent of timeshared computing. By using these APIs, developers can create interfaces and access the computing infrastructure provided by Amazon on a fee-based schedule, with the ability to grow as needed. Software developers, start-up companies, and established companies in need of reliable computing power are members of a large and growing crowd using Amazon services.

One of these services is the beta launch of Amazon Elastic Compute Cloud or EC2 [13]. The Amazon Elastic Compute Cloud provides virtualization for developers to load Amazon-managed machines with their preferred software environments and execute custom applications. This is accomplished by first creating an Amazon Machine Instance (AMI) with the operating system, custom configuration settings, libraries, and all needed applications. Once created, the AMI is loaded into the Amazon Simple Storage Service (AS3) and receives a unique identifier. The unique identifier can then be used to run as many instances of the AMI as needed using Amazon's APIs. Additionally, Amazon provides a set of pre-built AMIs that can be used by developers.

AMIs can be sized to the requirements of individual applications. AMIs fall into categories ranging from a small instance to an extra-large instance. A small instance has less memory, virtual cores, storage, and I/O performance than a large one. Similar to a timesharing system, Amazon bills users by the instance-hour. As the size of memory, number of cores, or other features increases, the instance-hour fee increases. Amazon offers standard instances as well as high-CPU instances.

Amazon is also now claiming location transparency for a globally distributed cloud. They are building out their computational footprint to be more geographically distributed. Additionally, they are improving fault tolerance by creating Availability Zones that will allow users to create instances of their applications in distributed regions.

Microsoft Approach to Cloud Computing

Microsoft announced its Azure Services Platform in October 2008 [14, 15]. Similar to the Amazon approach, Microsoft is developing a cloud-based, hosted-services platform. In addition to providing compute and storage resources for consumers to develop and host applications, Microsoft is also offering cloud applications that are already developed and ready for consumption.

The Azure Service Platform is built on the Windows Azure cloud operating system, which provides a development, hosting, and management environment for cloud applications. Numerous services are available on top of the Azure operating system including Live Services, SQL Services and .NET Services.

During the Community Technology Preview, Azure is offered for free to allow users and consumers to test and evaluate it. Potential users can also download an Azure SDK and Azure tools for Microsoft Visual Studio to simulate the Azure framework during the preview period. Once Azure is launched for commercial use it will be priced using a consumption-based model. Consumption will be measured in compute time, bandwidth, and storage and transactions (put and gets).

Microsoft is using a combination of Microsoft .NET framework and the Microsoft Visual Studio development tools to provide a base for developers to easily launch new solutions in the cloud. It is noted that both applications running in the cloud and outside of the cloud can use the Azure cloud platform. For the initial offering, only applications built with .NET can be hosted, but Microsoft claims that this constraint will be relaxed for Azure in 2009.

Azure's storage framework is based on storing of binary large objects (blobs), communications queues to provide access to the data via Azure applications, and a query language that can provide table-like structures. An Azure account holder can have one or more containers where each container can hold one or more blobs. Each blob has a maximum size of 50 GB, and can be subdivided into smaller blocks. To work with the blobs of data, entity and property hierarchies are provided through tables. These tables are not SQL-like relational tables and are not accessed using SQL. Instead, access to these tables is provided via the Microsoft Language Integrated Query (LINQ) syntax query language. Queues are also available to provide communication between instances as will be discussed later. Representational State Translation (REST) is the convention used to both expose and identify data stored in the Azure cloud. All Azure data storage is replicated three times to enhance fault tolerance.

The .NET Services provide access control at the application level, a service bus for exposing application services and allowing services to communicate with each other, and a workflow management system for creating complex services from existing simpler services (service orchestration).

SQL Services will be used to provide a set of services for working with both unstructured and relational data. The first set of SQL Services will only provide database services in the Azure cloud, but will expand in the future. While this service appears similar to a relational database storage model, Microsoft claims that it is slightly different in that it is a hierarchical data model without a pre-defined schema. Therefore, access to this data is not provided via a structure query language but instead through the RESTful interface or C# syntax defined by Microsoft's LINQ.

The Live Services use the cloud to store data while running on a variety of live operating desktop and mobile systems. This allows the Live Operating System to synchronize data across numerous related mesh systems. As an example, a user can create a mesh with his/her desktop, laptop, and mobile phone and keep them seamlessly synchronized at all times.

Windows Azure divides application instances into virtual machines (VMs) similar to the Amazon AMIs described earlier. Unlike the Amazon AMIs, a Windows Azure developer cannot supply his/her own VM image. Developers create Web role and Worker role instances, where a Web role accepts incoming HTTP requests and Worker roles are triggered by Web roles via a queue. Any work performed by a Worker role instance can then be stored in the Azure storage or sent outside of the cloud network. Web role instances are stateless. To expand the performance of an application, multiple Worker role instances can be run across dedicated processor cores. If a Worker role or Web role fails, the Azure fabric restarts it.

Other Cloud Computing Approaches and Applications

Amazon, Google, and Microsoft are not alone investing in computing as a service. Other organizations to test the waters include Dell, IBM, Oracle, and some universities.

IBM is providing a variety of cloud-based services by using existing functionality and capabilities of the IBM Tivoli portfolio [16]. Tivoli is a collection of products and software services that can be used as building blocks to support IBM Service Management software. IBM's cloud-based services, which target independent software vendors (ISVs), offer design of cloud infrastructures, use of worldwide cloud computing centers, and integration of cloud services.

Researchers at the University of Michigan (UM) have developed a novel anti-virus application using cloud computing ideas [17]. By aggregating a collection of open source and commercial anti-virus software as a cloud-based service and letting the individual anti-virus packages "vote" on whether an infection has occurred on a host, they demonstrated that their CloudAV service was more effective at detecting viruses than any single anti-virus software package. With a small software client running on each end user host, the UM researchers also claimed that a centralized virus detection system would be easier to manage in an enterprise than maintaining signature files and software releases on hundreds or thousands of end hosts.

Concerns and Challenges

Perhaps the biggest danger that arises when a technology gains sufficient interest from enough people is that it will begin to be viewed as a panacea. Gartner refers to such a situation as the *peak of inflated expectations* in their Hype Cycle [18]. While we believe that cloud computing can indeed be applied to many kinds of problems successfully, we also think that it's necessary to consider carefully whether the problem needing to be solved could best be addressed by an existing technology.

When we described Google's Bigtable data storage system, we compared it to RDBMSs. There are many problems that are best solved using a relational database, and systems like Bigtable do not add value. For example, a fundamental requirement of a banking database is that information about how much money is in each customer's bank account must be accurate at all times, even while money is being transferred between accounts or after a system has crashed. Such an application cries out for a transactional model that is part of an RDBMS, but not Bigtable. Being able to store petabytes of data is less important here than being able to execute transactions correctly.

The Amazon approach to cloud computing is ideal for small organizations or organizations with unpredictable computing usage requirements. For large organizations or organizations that process particularly sensitive data, this approach may not

Image credit:
***X-ray**: NASA/CXC/U. Colorado/Linsky et al.;*
Optical*: NASA/ESA/STScI/ ASU/J.Hester & P.Scowen.*

Image credit: NOAA Photo Library, NOAA Central Library; OAR/ERL/National Severe Storms Laboratory (NSSL)

make sense. With the Amazon approach, a user is effectively renting computing resources. Renting computing resources may not be the most cost-effective use of funds for a large corporation. As an organization grows in size and importance, the *value* of its data also increases dramatically. An automotive manufacturer would probably not want to store the highly proprietary designs for next year's car models on another company's servers. Similarly, a government agency and the citizens it serves would probably not want sensitive data such as citizens' tax returns to be stored on a computer system that is not owned and controlled by the government.

Cloud computing approaches use parallelism to improve the computational performance of applications. The Google MapReduce framework is particularly good at this so long as the problem fits the framework. Other approaches to high performance computing have similar constraints. It's very important for developers to understand the underlying algorithms in their software and then match the algorithms to the right framework. If the software is single-threaded, it will not run faster on a cloud, or even on a single computer with multiple processing cores, unless the software is modified to take advantage of the additional processing power. Along these lines, some problems cannot be easily broken up into pieces that can run independently on many machines. Only with a good understanding of their application and various computing frameworks can developers make sensible design decisions and framework selections.

Future Research Areas

Although much progress has already been made in cloud computing, we believe there are a number of research areas that still need to be explored. Issues of security, reliability, and performance should be addressed to meet the specific requirements of different organizations, infrastructures, and functions.

Security

As different users store more of their own data in a cloud, being able to ensure that one user's private data is not accessible to other users who are not authorized to see it becomes more important. While virtualization technology offers one approach for improving security, a more fine-grained approach would be useful for many applications.

Reliability

As more users come to depend on the services offered by a cloud, reliability becomes increasingly important, especially for long-running or mission-critical applications. A cloud should be able to continue to run in the presence of hardware and software faults. Google has developed an approach that works well using commodity hardware and their own software. Other applications might require more stringent reliability that would be better served by a combination of more robust hardware and/or software-based fault-tolerance techniques.

Vulnerability to Attacks

If a cloud is providing compute and storage services over the Internet such as the Amazon approach, security and reliability capabilities must be extended to deal with malicious attempts to access other users' files and/or to deny service to legitimate users. Being able to prevent, detect, and recover from such attacks will become increasingly important as more people and organizations use cloud computing for critical applications.

Cluster Distribution

Most of today's approaches to cloud computing are built on clusters running in a single data center. Some organizations have multiple clusters in multiple data centers, but these clusters typically operate as isolated systems. A cloud software architecture that could make multiple geographically distributed clusters appear to users as a single large cloud would provide opportunities to share data and perform even more complex computations than possible today. Such a cloud, which would share many of the same characteristics as a grid, could be much easier to program, use, and manage than today's grids.

Network Optimization

Whether clouds consist of thousands of nodes in a computer room or hundreds of thousands of nodes across a continent, optimizing the underlying network to maximize cloud performance is critical. With the right kinds of routing algorithms and Layer 2 protocol optimizations, it may become possible for a network to adapt to the specific needs of the cloud application(s) running on it. If application-level concepts such as locality of reference could be coupled with network-level concepts such as multicast or routing algorithms, clouds may be able to run applications substantially faster than they do today. By understanding how running cloud applications affects the underlying network, networks could be engineered to minimize or eliminate congestion and reduce latency that would degrade the performance of cloud-applications and non-cloud applications sharing the same network.

Interoperability

Interoperability among different approaches to cloud computing is an equally important area to be studied. There are many cloud approaches being pursued right now and none of them are suitable for all applications. If every application were run on the most appropriate type of cloud, it would be useful to share data with other applications running on other types of clouds. Addressing this problem may require the development of interoperability standards. While standards may not be critical during the early evolution of cloud computing, they will become increasingly important as the field matures.

Applications

Even if all of these research areas could be addressed satisfactorily, one important challenge remains. No information technology will be useful unless it enables new applications, or dramatically improves the way existing applications are built or run. Although the effectiveness of cloud computing has already been demonstrated for some applications, more work should be done on identifying new classes of novel applications that can only be realized using cloud computing technology. With proper instrumentation of potential applications and the underlying cloud infrastructure, it should be possible to *quantitatively* evaluate how well these application classes perform in a cloud environment. Along these same lines, experimental software engineering research should be conducted to measure how easily new cloud-based applications can be constructed relative to non-cloud applications that perform similar functions. This research should also compare the dependability of similar cloud and non-cloud based applications running in production environments. Application-focused research will help organizations make well-informed business decisions on where to apply cloud technology, and give cloud technology developers guidance on what kinds of improvements to the technology will provide the greatest benefits to application developers and end users.

Conclusion

We have described a number of approaches to cloud computing in this article and pointed out some of their strengths and limitations. We have also provided motivation and suggestions for additional research. The approaches outlined in this article, along with other strategies, have already been applied successfully to a wide range of problems. As more experience is gained with cloud computing, the breadth and depth of cloud implementations and the range of application areas will continue to increase.

Like other approaches to high performance computing, cloud computing is providing the technological underpinnings for new ways to collect, process, and store massive amounts of information. Based on what has been demonstrated thus far, ongoing research efforts, and the continuing advancements of computing and networking technology, we believe that cloud computing is poised to have a major impact on our society's data-centric commercial and scientific endeavors.

References and Further Reading:

[1] Globus Alliance homepage. http://www.globus.org, 2008.

[2] Barroso LA, Dean J, Hölzle U. Web search for a planet: the Google cluster architecture. IEEE Micro. 2003 March-April; 23(2).

[3] Ghemawat S, Gobioff H, Leung S. The Google filesystem. Proceedings ACM Symposium on Operating Systems Principles. 2003 October.

[4] Dean J and Ghemaway S. MapReduce: Simplified data processing on large clusters. Proceedings Operating Systems Design and Implementation. 2004 December.

[5] Dean J and Ghemaway S. MapReduce: Simplified data processing on large clusters. Communications of the ACM. 2008 January; 51(1).

[6] Gray J. Sort Benchmark home page. http://research.microsoft.com/barc/SortBenchmark, 2006.

[7] Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE. Bigtable: A distributed storage system for structured data. ACM Trans. on Computer Systems. 2008 June; 26(2).

[8] Hadoop Project homepage. http://hadoop.apache.org/core, 2008.

[9] HBase Project homepage. http://hadoop.apache.org/hbase, 2008.

[10] Cutting D. Hadoop: a brief history [Internet]. http://research.yahoo.com/files/cutting.pdf, 2008.

[11] Yahoo Developer Blogs. Apache Hadoop wins terabyte sort benchmark [Internet]. http://developer.yahoo.com/blogs/hadoop/2008/07/apache_hadoop wins_terabyte_sort_benchmark.html, 2008.

[12] IBM homepage. IBM and Google announce university initiative to address internet-scale computing challenges [Internet]. http://www-03.ibm.com/press/us/en/pressrelease/22414.wss, 2007.

[13] Amazon Web Services. Amazon Elastic Compute Cloud homepage. http://aws.amazon.com/ec2, November 2008.

[14] Chappel D. Microsoft Azure homepage. Introducing the Azure Services Platform [Internet]. http://www.microsoft.com/azure/whatisazure.mspx, October 2008.

[15] Microsoft Azure homepage. http://www.microsoft.com/azure, 2008.

[16] IBM homepage. IBM launches cloud services initiative [Internet]. http://www-03.ibm.com/press/us/en/pressrelease/25341.wss, 2008.

[17] Oberheide J, Cooke E, Jahanian F. CloudAV: N-Version antivirus in the network cloud. Proceedings of the 17th USENIX Security Symposium; 2008 July.

[18] Gartner homepage. Understanding Hype Cycles [Internet]. http://www.gartner.com/pages/story/story.php.id8795.s.8.jsp, 2008.