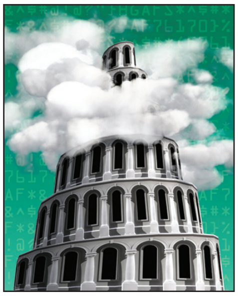
Editor: George Pallis • gpallis@cs.ucy.ac.cy



Decentralized Edge Clouds

Abhishek Chandra, Jon Weissman, and Benjamin Heintz • *University of Minnesota*

Nebula is a highly decentralized cloud that uses volunteer edge resources. Here, the authors provide insight into some of its key properties and design issues. They also describe a distributed MapReduce application scenario to illustrate the benefits and trade-offs of using distributed and decentralized clouds for distributed data-intensive computing applications.

Cloud computing's appeal is driven by three distinct factors: on-demand pay-as-you-go resource access that lowers the total cost of infrastructure ownership; the elasticity of resources to meet varying demand; and the colocation of computation and data to enable applications such as data analytics. These factors typically result in a largely centralized computing infrastructure confined to a few data centers, with applications confined to run in a single data center. This centralized cloud model is appropriate for largely stateless services with limited data transmission, such as the Web, or for analyzing batch data that originates inside the cloud, as when mining database transactions.

In other settings, however, the cloud's centralized nature might be limiting in terms of performance and cost. For example, moving large amounts of distributed data into the cloud could be costly, particularly when such data might be amenable to in situ local processing. Similarly, interactive cloud applications that are latency-sensitive might require close proximity to end users, particularly when such users are mobile and have limited network connectivity. For these cases, we envision moving the cloud closer to where the data, users, and even computation might reside. In some sense, clouds are an attempt to tame a distributed world in which these three entities are already dispersed worldwide. Consequently, making the cloud distributed can lead to better performance and enable a wider diversity of applications and services.

Distributed clouds are well suited to applications with large amounts of distributed data or interactive users. They're complementary to centralized clouds, which might still be the best option for applications requiring tight data and compute coupling. In other cases, applications could exploit each model's strengths by combining distributed and centralized clouds to create a hybrid cloud platform.

In general, we can classify distributed clouds into three architectural models, ranging from tightly coupled to highly dispersed: Multi-data-center clouds consist of multiple tightly coupled data centers belonging to the same cloud provider. Loosely coupled multi-service clouds combine services from multiple diverse cloud providers. Finally, decentralized edge clouds utilize edge resources to provide data and compute resources in a highly dispersed manner.

Here, we briefly examine the first two models, then discuss the third model in detail. In particular, we describe Nebula, a highly decentralized cloud we're building that uses volunteer edge resources.

Multi-Data-Center Clouds

Many cloud infrastructures are built using multiple data centers located in different geographic regions. As an example, Amazon Elastic Compute Cloud (EC2) has multiple Regions, each of which consists of data centers in a geographic area such as the US East/West, Asia, Europe, and so on. Such geographically distributed clouds provide two main benefits. First, users in different locations can use or be directed to resources closest to them, thus providing better latency and load distribution. Second, failures in one part of the network or one cloud location don't affect the rest of the cloud infrastructure, thus

1089-7801/13/$31.00 © 2013 IEEE

providing better fault tolerance and availability. Multiple data centers can be linked either through the commodity Internet or through dedicated network links. Because a single cloud provider controls these data centers, they're usually centrally managed and provisioned. Communication across geographic regions typically suffers from wide-area latencies, although bandwidth capabilities might be sufficient for moving data and computation across data center boundaries in the latter case. Individual Microsoft applications such as Windows Live Mesh (now SkyDrive), for example, are decomposed across multiple data centers.[1]

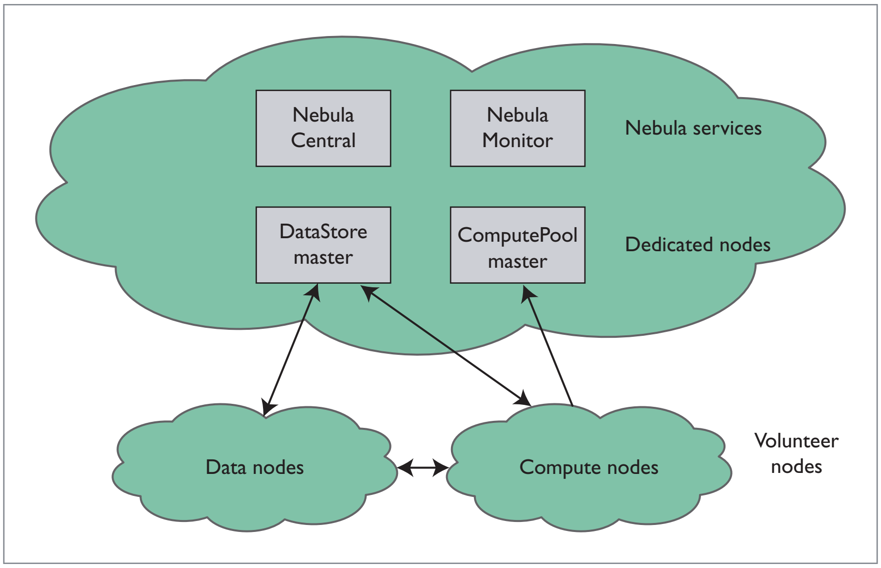


Figure 1. Nebula system architecture. A Nebula application's data and compute are managed by its DataStore and ComputerPool masters, respectively. All Nebula applications are multiplexed across the shared voluntary resource pool.

Loosely Coupled Multi-Service Clouds

Clouds represent a diverse landscape containing resource providers, application hosting platforms, and service providers across the *infrastructure-*, *platform-*, and *software-as-a-service* (IaaS, PaaS, and SaaS) spectrum. For example, the satellite earth imagery cloud (Google Earth) provides geographical data, while Amazon EC2 provides raw computational resources. Newly emerging, distributed data-intensive applications would likely require integrating and coordinating multiple distinct clouds, as with data mining across datasets in multiple clouds or scientific workflows that use distinct data and computation across widely distributed sources. The current cloud interaction paradigm is client-server (as with Web services or http), which forces all output data to flow back to the client even if it's intermediate in the end-to-end multicloud application. The problem is even more pronounced if the end-user application is resource constrained — for instance, if the network path to the end-user application is poor, as with a low-bandwidth wireless network. Such loosely coupled clouds can be more efficiently integrated using proxy networks[2] consisting of numerous logically connected edge nodes that can assume a rich set of roles such as routing, caching, and intermediate processing.

Decentralized Edge Clouds

Nebula is a highly decentralized and dispersed cloud infrastructure that uses edge resources for both computation and data storage.[3,4] Unlike the previously described cloud architectures, Nebula falls into a different part of the distributed cloud landscape. It's designed primarily for applications with highly dispersed data that are both large and widely distributed, so that data upload into a traditional cloud constitutes a nontrivial portion of the execution time. For example, consider a blog analysis application that processes blogs with multimedia content hosted throughout the Internet. Nebula uses edge resources for both computation and data storage to address the data's wide dispersion. The edge resources could be either volunteer or dedicated resources across content distribution networks (CDNs) or even ISPs, provided these were equipped to offer computational services. If the infrastructure employs volunteer edge resources, Nebula will also be cheaper in terms of the monetary costs to transport, store, and process data — benefitting, for instance, a small-scale application designer or user.

Nebula presents several challenges because of its highly decentralized cloud model in addition to its use of volunteer edge resources. First, such resources are heterogeneous, highly dispersed, and loosely coupled. Second, the system is susceptible to failures due to node churn and network connectivity problems. Third, the system must be easy to use and manage both for users who execute their applications on the volunteer platform, and for volunteer nodes that donate their resources. Furthermore, volunteer nodes must be completely safe and isolated from malicious code that might execute as part of a Nebula-based application.

Nebula implements numerous services and optimizations to address these challenges, including location-aware data and computation placement, replication, and recovery. Figure 1 shows the Nebula system architecture. In addition to the volunteer nodes that donate their computation and storage resources, Nebula consists of a set of global and application-specific services that are hosted on dedicated, stable nodes. These resources and services together constitute Nebula's four major components:

- *Nebula Central* is the front end for the Nebula ecosystem. It provides a simple, easy-to-use, Web-based portal that allows volunteers to join the system, application writers to inject applications, and tools to manage and monitor application execution.
- The *DataStore* is a scalable data storage service that enables efficient and location-aware data processing. Each DataStore consists of volunteer data nodes that store the actual data, and a DataStore master that maintains the storage system metadata and makes data placement decisions.
- The *ComputePool* provides computation resources through a set of volunteer compute nodes. Code execution on a compute node occurs inside a Google Chrome Web browser-based Native Client sandbox, thus providing a secure way to donate computational resources. Compute nodes within a ComputePool are scheduled by a ComputePool master that coordinates their execution. The compute nodes use the DataStore to access and retrieve data, and are assigned tasks based on application-specific computation requirements and data location.
- The *Nebula Monitor* conducts performance monitoring of volunteer nodes and network characteristics. This monitoring information consists of node computation speeds, memory and storage capacities, and network bandwidth, as well as health information such as node and link failures. This information is dynamically updated; the DataStore and ComputePool masters use it for data placement, scheduling, and fault tolerance.

These components work together to enable the execution of data-intensive applications on the Nebula platform. To this end, Nebula considers network bandwidth along with resources' computation capabilities in the volunteer platform. Consequently, resource management decisions optimize computation time as well as data movement costs. In particular, compute resources can be selected based on their locality and proximity to their input data, whereas data might be staged closer to efficient computational resources. In addition, Nebula implements replication and task re-execution to provide fault tolerance and robustness in the presence of failures.

Distributed Data-Intensive Cloud Computing

Let's now examine the opportunity and challenge of using distributed clouds to conduct data-intensive computation on geographically distributed data. We illustrate these issues using MapReduce[5] as an example application framework. MapReduce is a popular programming model used for numerous data-intensive applications. Here, we consider the problem of executing MapReduce across a distributed cloud environment, in scenarios where the input data is also geographically distributed.

The MapReduce programming model consists of two computational stages: *map* and *reduce*. Traditionally, the input data is prepartitioned and colocated on mapper nodes (those that execute the map tasks), whereas the intermediate data must be shuffled across the network between mappers and reducers. Bringing MapReduce to geographically distributed settings presents myriad challenges, perhaps the most important of which is wide-area communication. For MapReduce, unless input data and compute resources are already colocated, the system must push inputs over wide-area links, which can be prohibitively costly in terms of both performance and money. The shuffle between mappers and reducers also takes place over wide-area links. Consequently, communication time can easily come to dominate the overall execution.

For a given MapReduce application, the MapReduce scheduler can minimize communication costs by choosing the right placement of map and reduce computation. At one extreme, all of the distributed inputs can be pushed into a single centralized location for computation, while at the other, computation can be colocated with the input data, leading to purely distributed computation. The trade-off between these approaches regards the relative cost of pushing input data versus shuffling the intermediate data, as well as the use and load balancing of compute resources. It's interesting to explore the benefits and limitations of distributed cloud models in terms of where they lie between these two extremes.

Multi-data center clouds are most amenable to more centralized approaches because they focus a large number of resources across only a few sites. In the extreme case, it might be possible to carry out the entire MapReduce application at a single data center, allowing the shuffle operation to use fast local-area links exclusively. Multi-data-center clouds comprise only a few data centers, which are often connected using predictable dedicated links. As a result, this architecture lends itself well to optimizing application execution based on resource and network characteristics. For example, we've developed a model to optimize MapReduce execution times in these settings, leading to a 30 to 40 percent reduction in execution time over Hadoop, the popular open source MapReduce implementation. A major downside of the multi-data-center approach, however, is that unless the data centers are placed very close to the data sources, input data must be pushed over wide-area network links before computation can begin.

On the other hand, decentralized edge clouds are conducive to a purely distributed approach because they allow computation to be carried out very close to the input data. This is especially useful for applications that filter or aggregate input data or are map-heavy.

We've developed a MapReduce execution framework over the Nebula cloud that avoids the need for transferring large volumes of data over wide-area links by finding computation resources close to the input data. A MapReduce application is instantiated in Nebula by pushing the input data to the DataStore and providing the application code for map and reduce along with application parameters. The ComputePool master can then start assigning map and reduce tasks to compute nodes. Each map task obtains its input data from the DataStore, performs the map function on it, and partitions the result into as many output files as there are reduce tasks. The output files are uploaded to the DataStore at the end of each map task. A reduce task downloads the map outputs for its partition, carries out the reduce operation, and uploads the output back to the DataStore. Nebula uses a locality-aware MapReduce scheduler that schedules each task on compute nodes close to the data nodes holding the task's input data. As opposed to multi-data-center clouds, dynamic and reactive scheduling techniques[6] are appropriate for such a cloud model due to their highly decentralized nature, as well as their volatility.

Cloud computing services were initially deployed as centralized computing infrastructures. However, the inherently distributed nature of data, users, and computation has resulted in clouds becoming more and more distributed. We envision different cloud models coexisting and serving different types of applications based on their characteristics and requirements. These models can also complement each other through division of labor based on their individual strengths. Programming, deployment, and management of such distributed clouds will be a fertile area of research and development in the near future.

References

1. S. Agarwal et al., "Volley: Automated Data Placement for Geo-Distributed Cloud Services," *Proc. 7th Usenix Conf. Networked Systems Design and Implementation*, Usenix Assoc., 2010, pp. 2–2; http://dl.acm.org/citation.cfm?id=1855711.1855713.
2. S. Ramakrishnan et al., "Accelerating Distributed Workflows with Edge Resources," *Proc. 2nd Int'l Workshop on Workflow Models, Systems, Services and Applications in the Cloud* (CloudFlow 13), to appear, 2013.
3. M. Ryden et al., *Nebula: Data Intensive Computing over Widely Distributed Voluntary Resources*," tech. report TR 13-007, Dept. of Computer Science and Eng., Univ. of Minnesota, Mar. 2013.
4. A. Chandra and J. Weissman, "Nebulas: Using Distributed Voluntary Resources to Build Clouds," *Proc. Workshop Hot Topics in Cloud Computing* (HotCloud 09), Usenix Assoc., 2009; www.usenix.org/event/hotcloud09/tech/full_papers/chandra.pdf.
5. J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Proc. 6th Usenix Symp. Operating Systems Design & Implementation* (OSDI 04), Usenix Assoc., 2004, pp. 137–149.
6. B. Heintz et al., "Cross-Phase Optimization in MapReduce," *Proc. IEEE Int'l Conf. Cloud Eng.* (IC2E 13), IEEE, 2013, pp. 338–347.

Abhishek Chandra is an associate professor in the Department of Computer Science and Engineering at the University of Minnesota. His research interests are in operating systems and distributed systems. Chandra has a PhD in computer science from the University of Massachusetts Amherst. Contact him at chandra@cs.umn.edu.

Jon Weissman is a professor of computer science at the University of Minnesota. His current research interests are in distributed systems, high-performance computing, and resource management. Weissman has a PhD in computer science from the University of Virginia. Contact him at jon@cs.umn.edu

Benjamin Heintz is a PhD candidate in the Department of Computer Science & Engineering at the University of Minnesota. His research interests include distributed systems and data-intensive computing. Heintz has an MS in computer science from the University of Minnesota. Contact him at heintz@cs.umn.edu.

cn *Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.*