International Journal of Research in Engineering & Technology (IJRET)
ISSN 2321-8843
Vol. 1, Issue 2, July 2013, 105-110

MINIMIZING LATENCY IN CLUSTER WITH CLOUD COMPUTING

HAMSA K[1], ANUSHA RAMESH[2] & JAYASIMHA SR[3]

[1]Assistant Professor, Department of MCA, Acharya Institute of Technology, Bangalore, Karnataka, India

[2,3]Assistant Professor, Department of MCA, RV College of Engineering, Bangalore, Karnataka, India

ABSTRACT

The DDN Web Object Scaler (WOS) is a revolutionary object-based, cloud storage system that addresses the needs of content scale-out and global distribution. At its core is the WOS object clustering system, intelligent software that allows a massively scalable content delivery platform to be created out of small building blocks, enabling the system to start small, and easily grow to a multi-Petabyte scale Objects stored in the WOS cluster are managed by policies which determine where the data should physically reside. WOS policies dictate content distribution within the cluster. Using policy-based content distribution with WOS, organizations can easily create disaster recover sites, place content close to where it will be accessed to improve performance and latency, or share content across the globe.

One of the most unique features of WOS is that it maintains data location intelligence across the WOS cluster and minimizes object access latency, one of the biggest issues in cloud computing today. With this built-in intelligence, WOS ensures that data is always served back to clients from the "closest" location (i.e. with the lowest possible latency) and that bandwidth costs between zones are kept in check. The specifics of how WOS determines the closest instance of data is the topic of the remainder of this document.

KEYWORDS: PUT, GET, DELETE, DNS, IP

INTRODUCTION

Individual units i.e., Nodes that make up a WOS cloud can be deployed anywhere on the globe as long as they have Internet Protocol (IP) connectivity to each other. WOS enables all the nodes to work together, presenting an aggregated pool of capacity. WOS nodes are self-contained appliances configured with disk storage, CPU and memory resources, gigabit Ethernet network interfaces and the WOS clustering software, which manages the cluster and the data placement inside the cluster.

Nodes are administratively assigned to zones, which typically represent distinct geographic locations. The system administrator makes policy-based decisions about where object instances stored in WOS physically reside by defining the replica count on a per-zone basis this remains discussed in WOS Architecture.

WOS ARCHITECTURE OVERVIEW

As mentioned above, WOS maintains location and latency intelligence of all instances of all objects in the WOS cluster. WOS does this by maintaining, on each WOS node, an in-memory latency & object location map which defines the lowest latency path between that node and any object for a given request (PUT, GET, DELETE, etc.). In addition, when using the WOSLib client-side library (see below), the latency map is generated from that client's perspective and maintained on the client, which enables the client to directly access the closest (lowest latency) node directly from the client.

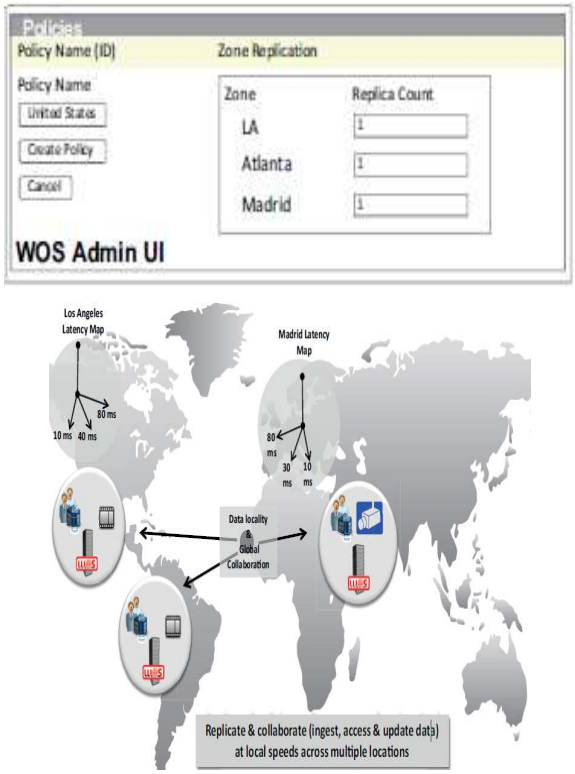

Figure 1: WOS Interfaces

Access to the WOS cluster is initiated by connecting to any node in the cluster via one of the interfaces defined below. WOS nodes are referenced via a unique IP address (assigned during WOS deployment), and each node maintains a lightweight web server that is accessible from the network via the API. WOS provides two application interface (API) methods into the WOS Cluster.

- **HTTP/REST:** Applications utilizing the REST interface gain access into the WOS cluster by sending HTTP requests (GET, PUT, DELETE, etc.) to port 80 of any node in the cluster, which then performs the operation. Requests are automatically directed to the node(s) that are best able to service the request without use of client redirects. Typically a multi-host DNS maintains the IP addresses of the nodes in the local zone and utilizing DNS round-robin to load-balance requests to local WOS nodes. The node receiving the client request proxies that request to whichever node(s) in the cluster it needs to access in order to perform the operation. This is transparent to the client. The node receiving the request uses its own latency map to prefer the most local source of data.

- **WOSLib (C++/Java/Python/PHP):** A high-performance component known as the WOS Library, or WOSLib acts as a client to the WOS cloud and runs on client machines that will access the WOS cluster directly. WOSLib provides an API that allows applications to GET, PUT, and DELETE objects. Additionally, WOSLib maintains information about the cluster topology and makes intelligent routing decisions about where in the cluster to store data, thus serving to load balance the cluster. WOSLib can be run on many machines at once, with each becoming a client to the cluster. In this way, parallel access to the cluster is enabled, allowing entire server farms to share a high-performance global content pool. All of the WOSLib requests are directed to the specific node that contains the object, which optimizes network bandwidth by removing the need to proxy requests.

While it may appear that there are significant differences between the REST/HTTP and WOSLib interface capabilities, in reality there is a subset of WOSLib running in every WOS node which maintains the OID and node latency map for each WOS node in the cluster. The REST/HTTP interface utilizes the node-level WOSLib (and the node latency

map) via the lightweight web server hosted on each WOS node. The REST/HTTP interface communicates with the node-level web server/WOSLib in much the same way that client applications interact with the client-level WOSLib.

The primary advantage of WOSLib on the client is that data latency is measured all the way from the application to the nodes in the WOS cluster rather than just between WOS nodes in the cluster as the case with the REST/HTTP interface. When using REST, applications target object requests to nodes in the cluster without the advantage of understanding object placement like WOSLib does. The consequence is that the use of REST results in more proxied requests and a corresponding increase in network traffic. Of course, there are benefits to using the REST/HTTP interface as well such as ease of porting the application to the API and offloading WOSLib processing from the application server.

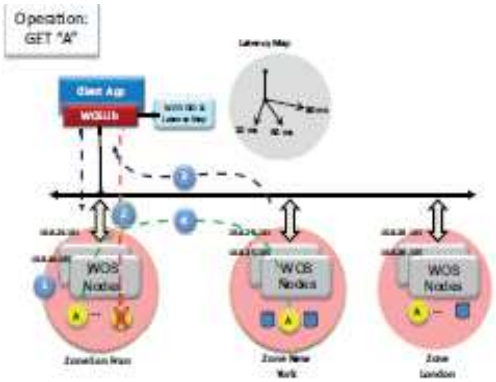

Figure 2: Data Corruption

Get Operation – Corrupted with Repair Process Flow

- WOSLib selects least latency path to the Object requested & sends GET request to San Fran node
- Node in Zone "San Fran" detects silent object corruption by comparing computed hash value with hash value stored in the object
- Client WOSLib identifies next nearest copy from its OID/Latency map & retrieves it to the client app
- In the background, the good copy in the New York zone is used to replace corrupted object in San Fran zone

REST Interface Latency

Below is an example of a HTTP/REST GET operation. Note that since there is no WOSLib at the client/application level, the application has no information about the WOS cluster except for the IP addresses of local WOS nodes that have been set up in DNS. Once DNS provides a WOS node IP address (using round robin for load balancing), the application sends the HTTP GET request to the randomly selected node.

By luck of the draw, If the object is actually located on this node, the node will immediately retrieve the object & return it to the client. However, if the WOS node does NOT contain the object, the node will use its local OID location and latency table to identify the location of the closest object, and then act as a proxy to return it to the client.

Note that in the HTTP/REST case, the latency is correctly maintained between the nodes of the cluster, but the latency between the client application and the node originally contacted is unknown. For HTTP/REST configurations, it is important that only the closest nodes are listed in DNS for that site to minimize latency.

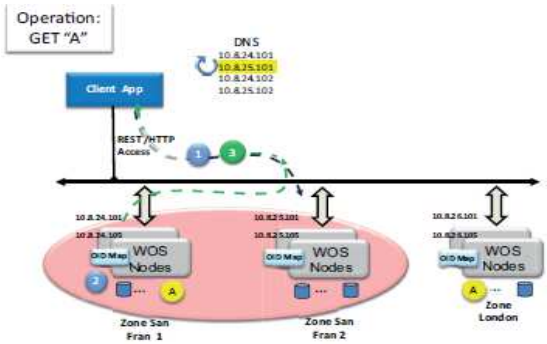

Figure 3: Get Request to WOS

Get Operation with HTTP/REST Interface Process Flow

- The Client Application connects to WOS node selected by DNS (round robin list) and issues an HTTP GET request
- Selected node locates closest instance of data using its local OID & node latency map. In this case the closest object is on another node in a different zone, but still local to the San Fran network
- The selected node acts as proxy to retrieve the closest instance of the object and route it back to the client application

The WOS NAS Gateway and WOS Cloud solutions are designed to maximize user benefit from the WOS cluster. Using the diagram below as an example and assuming that objects are replicated to all three sites, users at each site can collaborate (read, write, update) at local network speeds via NAS protocols because WOSLib will always access the closest object to the user. Object updates are immediately available at the other sites because the WOS NAS gateway databases continuously synchronize between sites, and once the updated object replicates, it will be accessible at local LAN speeds for users local to that site. Nearly identical multi-site collaboration and update capabilities are supported by WOS Cloud solution as well.

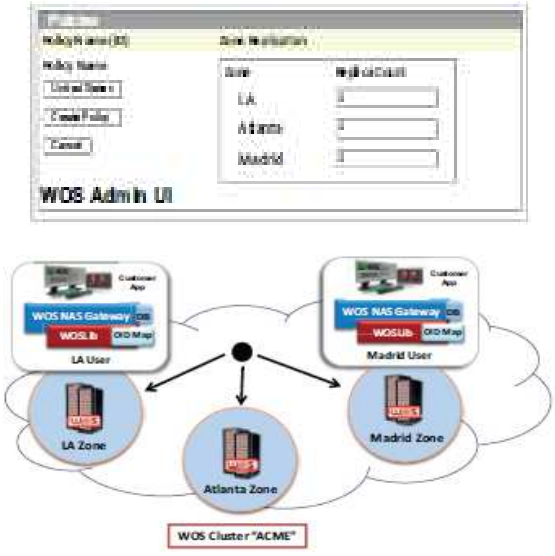

Figure 4: Cloud Environment in Cluster

For clouds and distributed computing environments, latency saps productivity and dramatically increases end user dissatisfaction. WOS is an ideal solution for distributed collaborative environments such as cloud service providers, university and government labs, and large distributed enterprises deploying private or internal clouds because it is designed

from the ground up to intelligently minimize latency and maximize performance by giving customers the ability to place storage resources and data close to their users, and automatically give them access to the closest instance of that data.

CONCLUSIONS

The explosion in the amount of content being generated, combined with the convergence of broadcast, production and consumer electronics is changing the Media and Entertainment industry in dramatic ways. Enormous opportunities are open to create innovative business models, reaching new customers and driving new revenue streams. DDN's innovative, enabling technology and it's experience in resolving the challenges of the world's most content-intensive environments provides its customers with a competitive edge in this rapidly changing, highly competitive market space.

REFERENCES

1. Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of retrievability via hardness amplification," in *Proc. of the 6th Theory of Cryptography Conference (TCC'09)*, San Francisco, CA, USA, March 2009.
2. K. D. Bowers, A. Juels, and A. Oprea, "Hail: A high-availability and integrity layer for cloud storage," in *Proc. of CCS'09*, 2009, pp. 187–198.
3. T. Schwarz and E. L. Miller, "Store, forget, and check: Using algebraic signatures to check remotely administered storage," in *Proc. of ICDCS'06*, 2006, pp. 12–12.
4. M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, andM. Isard, "A cooperative internet backup scheme," in *Proc. of the 2003 USENIX Annual Technical Conference (General Track)*, 2003, pp. 29–41.
5. M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transaction on Computer Systems*, vol. 20, no. 4, pp. 398–461, 2002.
6. L. Carter and M. Wegman, "Universal hash functions," *Journal of Computer and System Sciences*, vol. 18, no. 2, pp. 143–154, 1979.
7. J. Hendricks, G. Ganger, and M. Reiter, "Verifying distributed erasure-coded data," in *Proc. of 26th ACM Symposium on Principles of Distributed Computing*, 2007, pp. 139–146.
8. J. S. Plank and Y. Ding, "Note: Correction to the 1997 tutorial on reed-solomon coding," University of Tennessee, Tech. Rep. CS-03- 504, April 2003.
9. C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for storage security in cloud computing," in *Proc. of IEEE INFOCOM'10*, San Diego, CA, USA, March 2010.
10. C. Wang, K. Ren, W. Lou, and J. Li, "Towards publicly auditable secure cloud data storage services," *IEEE Network Magazine*, vol. 24, no. 4, pp. 19–24, 2010.
11. R. C. Merkle, "Protocols for public key cryptosystems," in *Proc. Of IEEE Symposium on Security and Privacy*, Los Alamitos, CA, USA, 1980.
12. Q. Wang, K. Ren, W. Lou, and Y. Zhang, "Dependable and secure sensor data storage with dynamic integrity assurance," in *Proc. Of IEEE INFOCOM'09*, Rio de Janeiro, Brazil, Appril 2009.
13. J. S. Plank, S. Simmerman, and C. D. Schuman, "Jerasure: A library in C/C++ facilitating erasure coding for storage.