Clouder: A Flexible Large Scale Decentralized Object Store

Architecture Overview

Ricardo Vilaça
Computer Science and Technology Center
University of Minho
rmvilaca@di.uminho.pt

Rui Oliveira
Computer Science and Technology Center
University of Minho
rco@di.uminho.pt

ABSTRACT

The current exponential growth of data calls for massive-scale capabilities of storage and processing. Such large volumes of data tend to disallow their centralized storage and processing making extensive and flexible data partitioning unavoidable. This is being acknowledged by several major Internet players embracing the Cloud computing model and offering first generation remote storage services with simple processing capabilities.

In this position paper we present preliminary ideas for the architecture of a flexible, efficient and dependable fully decentralized object store able to manage very large sets of variable size objects and to coordinate in place processing. Our target are local area large computing facilities composed of tens of thousands of nodes under the same administrative domain. The system should be capable of leveraging massive replication of data to balance read scalability and fault tolerance.

1. INTRODUCTION

Massive-scale distributed computing is a challenge at our doorstep. The volume of data quadruples every 18 months, while the available performance per processor doubles in the same time period [19]. Such large volumes of data tend to disallow their centralized storage and processing making extensive and flexible data partitioning unavoidable.

Relational Database Managements Systems (RDBMS) have been the key technology for the management of structured data. However, current systems are based on highly centralized, rigid architectures that fail to cope with the increasing demand for scalability and dependability. Deployed high performance RDBMS invariably rely on mainframe architectures, or clustering based on a centralized shared storage infrastructure. These, although easy to setup and deploy, often require large investments upfront and present severe scalability limitations.

Eschewing these large centralized architectures is key to provide elastic infrastructures capable of scaling-out and flexible enough to adjust to different application requirements. Some highly decentralized storage systems (e.g.. [5, 3, 4]) have recently been developed by major Internet based companies to support their respective Web operations and are being exploited as storage services *in the Cloud*. Having all started from similar requirements, these systems ended up providing a similar service: A simple tuple store interface, that allows applications to insert, query, and remove individual elements. More complex relational and processing facilities, let alone transactional guarantees common in traditional database management systems, are left to performed by the client applications outside the system. By doing so, these services focus on a specific narrow tradeoff between consistency, availability, performance, scale, and cost, that fits tightly their motivating very large application scenarios but is much less attractive to common business needs, in which there isn't a large in-house research development team for application customization and maintenance.

Our aim with Clouder is to steer the current tradeoff towards the needs of common business users, thus providing additional consistency guarantees and higher level data processing primitives smoothing the migration path for existing applications. At the same time, the boom of the Web 2.0, led to the emergence of new web applications or services, such as social network applications, that need to store and process large amounts of data. These applications usually have lower consistency requirements, which favor performance and can be invaluable to exploit simple asynchronous aggregation operations, but most of them cannot afford to deal with conflicts that arise from concurrent updates.

Clouder should be able to leverage large computing facilities composed of tens of thousands of nodes under the same administrative, or at least, ownership domain. Unlike current deployments [5] however, these nodes are not expected to be functionally or permanently dedicated to the object store but can even be commodity machines mainly dedicated to common enterprise or academic tasks.

In the remainder of the paper we present Clouder's motivation, its main characteristics and solutions as well as our current major challenges. A succinct comparison with related work concludes de paper.

WDDDM '09, March 31, 2009, Nuremberg, Germany

2. CLOUDER

2.1 Context and Motivation

To efficiently address distributed processing over massive-scale data stores data can no longer be managed in abstract but its structure, contents, and usage patterns need to be

disclosed to the data management system. Additionally, this applications require richer client APIs then existing data stores, which leads to object store that combine data store operations with in place processing.

With Clouder, our overall goal is to design and prototype a flexible, efficient and dependable peer-to-peer system able to manage very large sets of variable size objects and to coordinate in place processing. The system is expected to seamlessly fragment and persistently store application objects and to resort to combined replication techniques in order to balance scalability and fault tolerance.

Additionally, as a general purpose object store, it should manage very large sets of variable size objects. The achievement of these goals is to be demonstrated by a prototype providing consistent, conflict-free, data storage, and in-place processing capabilities, while allowing to leverage large computing infrastructures with distributed storage capabilities.

2.2 Architecture

To manage very large distributed sets of data two main approaches prevail: structured network overlays [16, 20, 18], in which all nodes are logically organized in a well-know structure; and unstructured network overlays [14, 9, 2], in which nodes are not (a priori) structured but are probabilistically managed.

The structured approach, heavily depends on the attributes and adjustment of the logical overlay to the underlying physical network. A correct overlay contributes to an efficient communication leveraging, nodes and links with higher capacity, locality, etc. Furthermore, as the overlay provides approximate views of the whole system it is possible to attain reasonable guarantees of message delivery and order. On the other hand, the usability of a structured overlay is limited by the dynamics of the system's membership. Changes on the set of participants lead to recalculating the overlay which, in the presence of high churn rates, can be impractical.

On the contrary, an unstructured network does not rely on any a priori [2] global structure but on basic gossip communication. Each node uses local information about its neighborhood and relays information to a random subset of its neighbors. With the correct fan-out these protocols ensure a high probability of message delivery. Due to their unstructured properties gossip communication is naturally resilient to churn and faults and simple to maintain.

For Clouder, we adopt both approaches for the management of two collaborating layers, a soft- and a persistent-state subsystem, of distinct structural and functional characteristics (Figure 1).

At the top, a soft-state layer is responsible for the 1) client interface, 2) data partitioning, 3) caching, 4) concurrency control, and 5) high level processing. Clouder provides a generic object-oriented model. It is assumed that objects have a unique key. The current client API allows applications to perform simple object manipulations operations (*put(key,object)*, *object get(key)*, and *delete(key)*), to iterate over standard collections and execute pre-defined general processing operations *object* operation(args*)*. Depending on the application requirements object keys can be ordered to allow range access. In both cases, key space is horizontally partitioned into groups of objects that form units of data distribution. To facilitate its adoption and the migration of general purpose applications to Clouder, a subset of the Java Persistence API is being implemented and offered

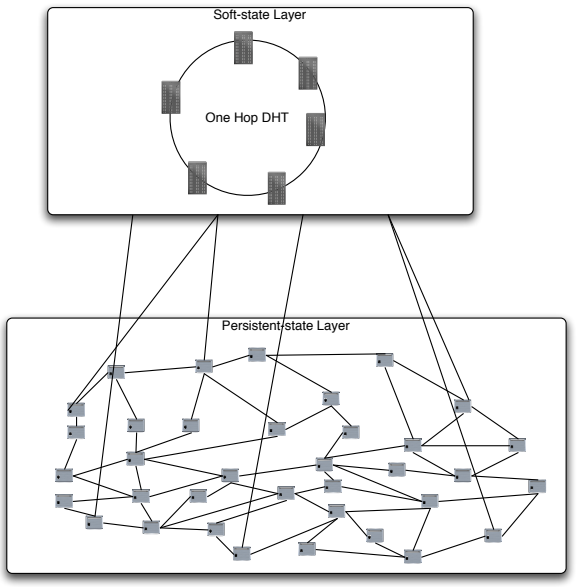

Figure 1: Clouder Architecture

as a Web Service. Others, such as LINQ to Entities can also be envisaged.

This soft-state layer is assumed to be have a reasonably stable membership. These nodes require global knowledge as they need to delegate and coordinate among themselves the partitioning of data. An "one hop" DHT like Beehive [15] or OneHop [8] seems the best fit to manage such subsystem [17]. Each node of the DHT manages the mapping of its object space partition into a set of nodes in the underlying persistent-state layer. The resilience of each node metadata is currently ensured through the use of synchronous primary-backup replication within the soft-state layer and asynchronously saved in the persistent-state layer. In case of the failure of a node fail-over to one the backups is automatic. Should all metadata of a partition be lost at the soft-layer (due to multiple failures) recovery through the persistent layer is possible albeit subject to miss updates inherent to asynchronous writes. Each node keeps memory cache of a set of objects on its behalf. By having backups also caching the primary's soft-copies of data, the replication schema, primarily target at fault tolerance, can also help on load-balancing read operations.

With automatic and timely failover ensured, having all operations addressed to a given object handled by a single master node (even if it can then delegate control) obviates much of the per-object concurrency control complexity. Load-Balancing can easily be adjusted through repartitioning of the object space. With this approach, the serialization of operations becomes straightforward while, if allowed, relaxed consistency criteria can be accommodated helping the trade-off towards performance. at the expense of overloading the master node.

Processing at the soft-layer is meant to avoid off-system aggregation which would, in general, defeat the purpose of the whole computation model.

Stable storage is provided by the persistent-state layer. This subsystem is meant to be supported by a very large

local network with weak assumptions on the nodes and network reliability. These characteristics do not suit a structured approach due to the overhead of the reconfigurations imposed by frequent membership changes. An unstructured network approach is adopted as epidemic communication is specially insensitive to churn and provides fine tuning possibilities invaluable to self-adaptation and self-tuning [12]. Objects stored in the persistent-layer are massively replicated through gossiping. An object is assumed to be *safely* stored once it is stored in m nodes (which become the entry points for the object at the soft-layer). For the sake of fault-tolerance further replicas of the object are created.

2.3 Current Open Issues

While we feel quite confident with the current design of the Clouder system to address its initial goals, we also identify many open issues that justify deep research.

Currently, our client API is general but nevertheless limited. It considers the exchange of opaque objects and their manipulation through their (non-disclosed) filter methods for standard iterations. This is a most salient aspect of the system and the subject of ongoing research. To efficiently address distributed processing data can no longer be managed in abstract but its structure and contents need to be disclosed to the data management system. Standard and emergent applications require richer client APIs based on common entity-attribute-value model [13] as acknowledge by the design of recent proposals of large data storage services [1, 7].

The one hop DHT used in the soft-state layer must guarantee that even in the failure of nodes its structure is maintained so that only one is responsible to handle requests from a given range. Otherwise, total order is not assured and conflicts may occur. The eventual consistency of DHTs has been studied in [11] but further research is required to strengthen the "one leader" guarantees and prevent update conflicts.

We intend to take advantage of the massively parallelization and dissemination properties of the persistent-layer to offer simple asynchronous processing primitives like counting or summing over the set of data [10]. The typical use case would be to have the system continuously compute some relevant statistics about a set of data and offer it to the client upon request within a given degree of accuracy.

The management of distributed objects on top of a peer-to-peer network raises many and interesting challenges. Some are inherent to the large scale of the system which is not forgiving to centralized algorithms or flooding protocols and require judicious control of recurring, system wide, a priori negligible tasks. Others are due to the dynamics of the system's membership and each node current capabilities. The management of the membership on the persistent-state layer and its correct articulation with the above soft-state is not clear to us yet. On one hand the assumption on scale, dynamics and reliability forfeits any strict membership control while, on the other hand, failing to directly access the nodes holding the needed data has a serious impact on the system's performance.

3. RELATED WORK

Major companies like Google, Yahoo and Amazon developed their own decentralized data store to tackle internal data management problems and support their current and future Cloud services. Although having common aspects, like per item consistency, horizontal partitioning, and aiming at being scalable to hundreds of nodes, these systems differ in some requirements, in their architecture and implementation.

Google's BigTable[3], Yahoo's PNUTS [4] and Amazon's Dynamo [5] provide a similar service: a simple tuple store interface, that allows applications to insert, query, and remove individual elements. BigTable and PNUTS additionally support range access in which clients can iterate over a subset of data. Atop of these low level data stores, other services [7, 1] with richer data models and APIs start to appear. These systems were initially, and still are to some extent, justified by internal needs and thus offer particular guarantees and trade-offs. Worth mentioning, are the consistency guarantees provided. At this level, consistency is preserved per tuple or record. Both BigTable and PNUTS are conflict free, providing a serialization of write operations. Read freshness can nevertheless be controlled by the application. Dynamo, on the other hand, does not ensure conflict free executions, allowing the application to see diverging values and delegates conflict resolution.

To achieve high availability these data stores rely on dedicated infrastructures with high levels of service which do not make them appropriate to leverage other less reliable networks such as those found in banks and universities.

4. REFERENCES

[1] Inc Amazon.com. Amazon simpledb. http://aws.amazon.com/simpledb/, 2008.

[2] Nuno Carvalho, Jose Pereira, Rui Oliveira, and Luis Rodrigues. Emergent structure in unstructured epidemic multicast. In *DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 481–490, Washington, DC, USA, 2007. IEEE Computer Society.

[3] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 205–218, Berkeley, CA, USA, 2006. USENIX Association.

[4] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1(2):1277–1288, 2008.

[5] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 205–220, New York, NY, USA, 2007. ACM.

[6] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.

[7] Google. Google app engine datastore. http://code.google.com/appengine/docs/datastore/, 2008.

[8] Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues. Efficient routing for peer-to-peer overlays. In *First Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, March 2004.

[9] Márk Jelasity and Ozalp Babaoglu. T-man: Gossip-based overlay topology management. In *In 3rd Int. Workshop on Engineering Self-Organising Applications (ESOA'05)*, pages 1–15. Springer-Verlag, 2005.

[10] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(3):219–252, 2005.

[11] Jayanth Kumar Kannan, Matthew Chapman Caesar, Ion Stoica, and Scott Shenker. On the consistency of dht-based routing. Technical Report UCB/EECS-2007-22, EECS Department, University of California, Berkeley, Jan 2007.

[12] Miguel Matos, José Pereira, and Rui Oliveira. Self tuning with self confidence. In *In "Fast Abstract", Supplement of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2008.

[13] Prakash Nadkarni and Cindy Brandt. Data extraction and ad hoc query of an entity-attribute-value database. *Journal of the American Medical Informatics Association*, 5(6):511–527, 1998.

[14] José Pereira, Luís Rodrigues, Maria J. Monteiro, Rui Oliveira, and Anne-Marie Kermarrec. Neem: network-friendly epidemic multicast. *Reliable Distributed Systems, 2003. Proceedings. 22nd International Symposium on*, pages 15–24, Oct. 2003.

[15] Venugopalan Ramasubramanian and Emin Gün Sirer. Beehive: O(1)lookup performance for power-law query distributions in peer-to-peer overlays. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 8–8, Berkeley, CA, USA, 2004. USENIX Association.

[16] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM.

[17] John Risson, Aaron Harwood, and Tim Moors. Stable high-capacity one-hop distributed hash tables. In *ISCC '06: Proceedings of the 11th IEEE Symposium on Computers and Communications*, pages 687–694, Washington, DC, USA, 2006. IEEE Computer Society.

[18] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[19] David Skillicorn. The case for datacentric grids. Technical Report ISSN-0836-0227-2001-451, Department of Computing and Information Science, Queen's University, November 2001.

[20] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.