About the Feasibility of Transactional Support in Cloud Computing

Francisco Maia, Rui Oliveira
Departamento de Informática
Universidade do Minho
Braga, Portugal
{*fmaia, rco*}*@di.uminho.pt*

José Enrique Armendáriz-Iñigo
Departamento de Ingeniería Matemática e Informática
Universidad Pública de Navarra
Pamplona, Spain
enrique.armendariz@unavarra.es

***Abstract*—In this paper we review what it has been stated so far about transactional support on the cloud computing environment. Then, we propose to extend them with some ideas already stated in replicated databases, like the certification process, to solve certain problems about coordination in the commit phase of transactions in the cloud.**

***Keywords*-transactions; distributed data storage; scalability; cloud computing;**

I. Introduction

Cloud computing has emerged as one of the most promising computing paradigms in developing highly scalable distributed systems [1], [2], [3]. Its popularity is due to the fact that it provides the notion of elasticity and illusion of infinite resources on the cloud. Resources can be dynamically added or removed as the system load varies. It permits to transfer the risk to cloud service providers (like Amazon, Yahoo!) instead of the smaller company developing an application. There is no up front cost and features a “pay-as-you-go” model, this allows new applications to be easily and rapidly tested without the expensive costs involved in developing the proper infrastructure.

We have that most of distributed applications, like those web-based, are divided into three layers: application server, web server and database server. The application and web servers can be easily scaled up by adding new servers; however, both of them are built on top of the database server. This database constitutes the bottleneck of the system, even though it can be replicated for higher scalability and availability in a master-slave configuration [4] where read-only transactions can be run on secondaries or a multimaster [5] by taking advantage of communication primitives featured by a Group Communication System [6]. In any case, the problem of scalability is partially alleviated since in the case of the former the bottleneck is the master for update intensive scenarios whereas in the latter the total order broadcast primitive used to synchronize all replicas does not scale well after tens of replicas are considered. Keep in mind that replicated databases are still providing to final users richer semantics through SQL semantics than those provided by cloud based applications which are based on key-value pairs storage [7], [8] and no transactional support.

The key scalability issue here for database replication and, most importantly, transactional support is to reduce the number of messages to be transmitted [2] and, respectively, the number of participants to decide the outcome of transactions [1]. The first one tries to circumvent the problems arisen with distributed state maintenance which is deadlock prone and hard to manage while the second one is focused on reducing the overhead and response time of a transaction without penalizing availability. Of course, there has been some works that provide consistent data storage without transaction support [7], systems that lies amid [2], [3] and a new system, called ElasTras, that supports transactions in the cloud [1]. These systems do not provide a language query like SQL to query transactions but they can scale horizontally by static or dynamic partitioning pretty much more than traditional replicated databases. They have different commitment protocols that range from Paxos [9] which is costly or use an evolved form of the 2-phase-commit protocol. In this paper we want to discuss the feasibility of existing solutions developed in multimaster database replication [5] to the cloud environment with its own requisites [1].

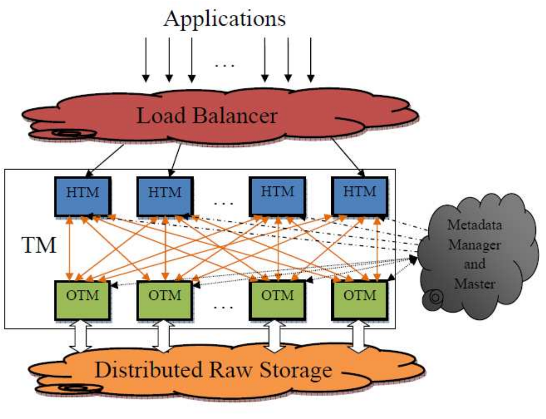

Figure 1. Abstraction of the architecture of the system

II. System Model

The system model we assume is very similar to ElasTras (see Figure 1). The model assumed is the key-value data model as used in BigTable [8]. At the top of the system is the load balancer where requests are handled and forwarded to a Higher Level Transaction Manager (HTM in Figure 1). An HTM can decide to execute the transaction locally, if it is read-only by caching, or redirect it to the appropriate Owning Transaction Manager (OTM) which owns exclusive access rights to the *partitioned* data accessed by the transaction that it has aggressively cached. data can be *statically* or *dynamically* partitioned. The set of OTMs and HTMs are responsible for transactional guarantees and constitute the core of the system denoted as Transaction Manager (TM). This set can grow up or shrink as there is an increase or a decrease in demand. The actual data is stored in the Distributed Storage Layer that takes care of replication and fault-tolerance. The Metadata Manager and Master (MMM) stores the system state and checks failures of OTMs and reassigns partitions for load balancing. MMM needs synchronous replication of its contents and uses Paxos [9] for that purpose.

III. Transaction Management

Static partitioning is similar to database partitioning and each one is assigned to an OTM. This approach has the benefit that the application is aware of partitions an transactions can be limited to a partition and provide ACID properties. On the contrary, with dynamic partitioning, ElasTraS has to manage and define (by range or hash-based) the partitions. Moreover, a transaction can potentially access several partitions while the scalability of the system has to be maintained. Hence, to avoid the management of a distributed transaction, it uses a *minitransaction* [2].

As already mentioned, a minitransaction consists in an evolution of the 2PC protocol where an HTM is the coordinator and participants are OTMs. The semantics of a minitransaction are restrictive, it includes a set of items to be read, compared and to be written if some condition is satisfied. This information goes along with the first phase of the commit request, the reason for this comes from scalability issues as no message is exchanged but the commit request itself. A transaction will be committed if all participants have a yes vote in their logs but the coordinator has no log, as opposed to traditional 2PC. This phase will only block if some involved OTM fails instead of the coordinator (HTM). Locks are only held during this phase and to avoid deadlocks, each OTM tries to acquire locks without blocking; if it fails, it will release all locks and send an abort message to the HTM.

IV. Our Proposal

From what it has been seen before, it could be seen that the failure of an OTM stops all the minitransactions involved in that partition. In [2], it is said that this seems realistic since if there is an application trying to access that data, the application has to remain blocked anyway. It can also be avoided by setting a set of “logical participants” thanks to the replication of the OTM. We can achieve this by stating that there can be several OTMs as owners of a given partition that perform a certification test like in [5]. Hence, we can achieve higher availability at the cost of a quantifiable latency cost. Nevertheless, as we are dealing with lower intensive update workloads this will not occur that often. Besides, as we have a certification process over each partition we could avoid the 2PC of minitransactions by extending the certification process to OTMs repsonsible for different partitions. In this case, the certification protocol will simply consist of something like a vote from each partition, because partitions do not need to receive nor apply the updates from each other.

Acknowledgment

This work has been partially supported by the Spanish Government under research grant TIN2009-14460-C03-02.

References

[1] S. Das and D. El Abbadi, “Elastras: An elastic transactional data store in the cloud,” in *USENIX 2009 HotCloud Conference Proceedings*, 2009.

[2] M. K. Aguilera, A. Merchant, M. A. Shah, A. C. Veitch, and C. T. Karamanolis, “Sinfonia: A new paradigm for building scalable distributed systems,” *ACM Trans. Comput. Syst.*, vol. 27, no. 3, 2009.

[3] M. Armbrust, A. Fox, D. A. Patterson, N. Lanham, B. Trushkowsky, J. Trutna, and H. Oh, “Scads: Scale-independent storage for social computing applications,” in *CIDR*. www.crdrdb.org, 2009.

[4] C. Plattner, G. Alonso, and M. T. Özsu, “Extending dbmss with satellite databases,” *The VLDB Journal*, vol. 17, no. 4, pp. 657–682, 2008.

[5] Y. Lin, B. Kemme, M. Patiño-Martínez, and R. Jiménez-Peris, “Middleware based data replication providing snapshot isolation,” in *SIGMOD Conference*, F. Özcan, Ed. ACM, 2005, pp. 419–430.

[6] G. Chockler, I. Keidar, and R. Vitenberg, “Group communication specifications: a comprehensive study,” *ACM Comput. Surv.*, vol. 33, no. 4, pp. 427–469, 2001.

[7] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: amazon’s highly available key-value store,” in *SOSP*, T. C. Bressoud and M. F. Kaashoek, Eds. ACM, 2007, pp. 205–220.

[8] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable: A distributed storage system for structured data,” *ACM Trans. Comput. Syst.*, vol. 26, no. 2, 2008.

[9] L. Lamport, “The part-time parliament,” *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, 1998.