Scalable Transactions in the Cloud: Partitioning Revisited*

Francisco Maia[1], José Enrique Armendáriz-Iñigo[2], M. I. Ruiz-Fuertes[3], and Rui Oliveira[1]

[1] Computer Science and Technology Center, University of Minho, Braga, Portugal, {fmaia,rco}@di.uminho.pt

[2] Departamento de Ingeniería Matemática e Informática, Universidad Pública de Navarra, Pamplona, Spain, enrique.armendariz@unavarra.es

[3] Instituto Tecnológico de Informática
Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain, miruifue@iti.upv.es

Abstract. Cloud computing is becoming one of the most used paradigms to deploy highly available and scalable systems. These systems usually demand the management of huge amounts of data, which cannot be solved with traditional nor replicated database systems as we know them. Recent solutions store data in special key-value structures, in an approach that commonly lacks the consistency provided by transactional guarantees, as it is traded for high scalability and availability. In order to ensure consistent access to the information, the use of transactions is required. However, it is well-known that traditional replication protocols do not scale well for a cloud environment. Here we take a look at current proposals to deploy transactional systems in the cloud and we propose a new system aiming at being a step forward in achieving this goal. We proceed to focus on data partitioning and describe the key role it plays in achieving high scalability.

Keywords: Distributed Systems, Cloud computing, Transactional support

1 Introduction

Nowadays, the most common service architecture to build enterprise web applications is based on a multi-tier approach. We can identify three different layers. The web server layer, the application server layer and the data layer. Typically, when the application load increases we can scale the web and application layers horizontally. This means we can add web servers and application servers and the system will be able to cope with more client requests. This is not the case at the data layer. To scale the data layer and still provide consistency and persistency

* This work has been partially supported by the Spanish Government under grant TIN2009-14460-C03-02 and by the Spanish MEC under grant BES-2007-17362 and by project ReD – Resilient Database Clusters (PDTC/EIA-EIA/109044/2008).

guarantees, a replication mechanism must be implemented. However, traditional database replication techniques do not scale well after a few tens of replicas [1].

Following the CAP (Consistency, Availability and Partition Tolerance) theorem [2][3] it is clear that we cannot provide a data management system which is at the same time always available, partition tolerant and providing consistent data. On traditional relational database management systems (RDBMS), availability is often compromised in order to guarantee data consistency. These systems will not accept client requests if they cannot serve consistent data as a consequence of failures. Guaranteeing consistency is also the main reason why these systems do not scale well, as the replication mechanisms are costly and heavily depend on synchronization.

Recently, cloud computing has emerged as a promising computing paradigm aimed at developing highly scalable distributed systems. Cloud computing comes from the idea that computing should be seen as an utility with a model of use similar to that of the electric network. Concepts such as *Infrastructure as a Service*, *Platform as a Service* or *Software as a Service* then became very common. Each resource should be seen as a service and all the implementation details hidden from the client. To provide this level of abstraction to the client, properties such as elasticity and high availability became crucial to the implementation of these services. Elasticity denotes the capacity of a system to adjust its resource consumption according to the load, while availability typically represents revenue to the service provider.

Cloud computing impelled a number of companies (Amazon, Microsoft, Google or Hadoop) to provide a new set of data management services with elasticity, availability and cost effectiveness as their core features. The elasticity property of these systems required them to scale really well and thus these data management systems became a building block for applications with incredible fast growth, such as social network applications [4]. Examples of these systems are Google Bigtable [5], Amazon's Dynamo [6] or PNUTS from Yahoo! [7] just to mention a few. To provide high scalability and availability, these systems often compromise consistency guarantees, in contrast to the traditional approach described above. These data storage services often provide a key-value data model and lack transactional guarantees even for a single row access. Given the data model and the fact they do not support SQL, these systems are often grouped under the name of *NoSQL* systems.

In between the traditional RDBMS and the NoSQL systems, a third kind of data management systems is being proposed. Namely, there have been several attempts to provide transactional support in the cloud [8–12]. These systems aim at providing high scalability and availability and yet still provide data consistency guarantees by supporting some types of transactions. The key idea behind these systems is to simplify as much as possible all kind of interaction among replicas to decide the outcome of a transaction. These systems also rely heavily on data partitioning to provide the desired scalability properties.

Along this paper we look at data partitioning as a key to scalability. In order to better illustrate our idea we take a look at one of the existing systems that provides simple transactional support in the cloud. We propose an architecture of our own aimed at solving some issues the previous system may exhibit. In Section 3, we highlight the role partitioning plays in scalability. We discuss some related work in Section 4. In Section 5, we take a look to the next step forward in all the ideas covered in the paper and their implementation in a real system. Finally, conclusions end the paper.

2 Transactional Support in the Cloud

Alongside transactional RDBMS and the more recent NoSQL databases a novel approach is being made regarding data management. In the pursuit of the strong consistency guarantees made by RDBMS and the high scalability and availability of the NoSQL databases, some proposals have been made representing a compromise between the two worlds. *ElasTraS* [10] is an elastic, scalable and self managing transactional database for the cloud. The ElasTraS system provides some basic transactional support while still providing high scalability.

2.1 *ElasTraS*

ElasTras allows the management of both large single tenant databases and the small independent databases commonly observed in a multi-tenant DBMS. In any case, data is partitioned (small databases may fit into a single partition) and transactions are only allowed to access data on a single partition. ElasTraS offers full transactional RDBMS functionality to small single-partition databases and still a rich functionality for the multiple-partition case, relaying on a judicious schema partitioning.

The architecture of ElasTraS is depicted in Figure 1. The system has four main components. We succinctly describe each one of these components and the relations between them in order to understand how ElasTraS works.

The *Owning Transaction Manager (OTM)* component has a set of partitions for which is responsible. A certain OTM is responsible for (owns) a partition when that partition is assigned to it by the system. Each partition is owned by a single OTM at a time. The OTM executes transactions guaranteeing *ACID* properties at the partition level. It is important to remember that, even if each OTM can own more than one partition, each partition is independent from each other and there is no support for inter partition transactions. ElasTraS was designed in a way that clearly separates system data from application data. This separation makes it possible to remove system data and system management from the application data path thus favoring scalability. This feature led to two key components of the system: the *TM Master* and the *Metadata Manager*.

The TM Master monitors the system and is responsible for failure recovery and for making the necessary adjustments in partition assignment to tune the system according to its load. In a sense, it is the component that governs system elasticity.

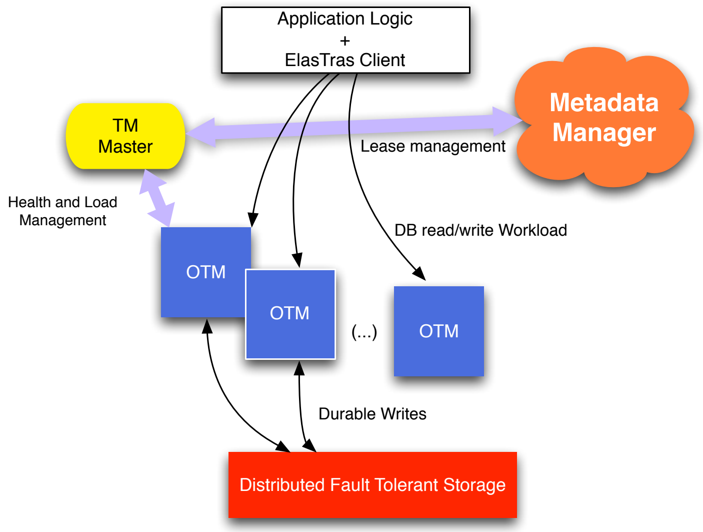

Fig. 1. ElasTraS architecture

The Metadata Manager stores the system state. This includes the mapping of partitions to OTMs (the partition ownership). By partition ownership we mean information about the nodes responsible for a certain partition of the data. The Metadata Manager is the component that guarantees that a certain partition is owned by a single OTM. This is done by recurring to a lease mechanism. Detailed information on these mechanisms can be found in [10] and is beyond the focus of this paper.

The Metadata Manager component is similar to the Chubby [13] system used at Google. This component is not in the data path and, as the information it stores is not expected to grow nor change very often, it does not represent a scalability bottleneck. However, this is a critical component of the system and downtime of the Metadata Manager represents downtime of the system as a whole. To avoid this from happening the Metadata Manager is, like Chubby, replicated using an implementation of the Paxos algorithm[14].

The last component is a *Distributed Fault-tolerant Storage* which persistently stores application data and a commit log which will allow recovery in the presence of OTM failures.

As stated before, ElasTraS can be used as a single large database where data is partitioned across the OTMs and as a multi-tenant database where each partition is an independent small database. Data partition assumes a key role in this design and will influence the system's performance. Further on we will devote a section of this paper on this topic.

2.2 What Can Be Improved in ElasTraS

ElasTraS represents a pragmatic approach to the design of a transactional data store for the cloud. However, there are some aspects of the system's design we believe can be improved. The first observation we can make is that ElasTraS will scale really well if we have several small databases and we can fit each one into a single partition or if we have a large database that we can split into several independent partitions. If this is not the case the system's bottleneck will be the OTM. The system would be able to cope with as many requests as the single OTM. To improve the system's performance in this case we would have to scale up the OTM which is not the best solution and hinders elasticity.

Moreover, recovery of the OTMs in ElasTraS implies downtime of the partitions the OTM owns. In the case of OTM failure the TM Master detects the failure and launches a new OTM. This new OTM will have exclusive ownership over the same set of partitions as its predecessor and will have access to the commit log. With log information it will be able to recover the state and start to service that set of partitions. During this OTM recovery process the set partitions it owns will be unaccessible. This might not be desirable specially when a database is entirely assigned to a single partition as it will mean downtime of the entire database, no matter how fast the recovery process is.

2.3 Possible Solution and Our Proposed System

We propose a new system evolved from ElasTraS that aims at solving the problems stated in the previous section. The main change we introduce is at the OTM level and in the way transactions are executed. The TM Master and Metadata Manager components remain with similar functions. With availability in mind, instead of assigning a partition to a single OTM, we assign it to a *OTM group*. This OTM group is a set of OTMs under a replication mechanism. Specifically, we apply a Database State Machine (DBSM) approach [15][16]. The failure of a single OTM does not represent unavailable partitions on an OTM group, thus improving the overall availability of the system at the cost of adding the new replication layer. The system elasticity is also guaranteed as we can create new OTM groups when load increases or reassign partitions when the load decreases. Each OTM group can also be assigned more than one partition and at the limit we can have an OTM group per partition.

As we mentioned earlier, a strong consistent replication mechanism does not scale well. However, in our design we only consider a small number of OTMs per OTM group which will not make the replication mechanism the system's bottleneck. Moreover, the DBSM approach uses a single atomic broadcast message to perform the certification protocol over the replicas avoiding blocking protocols such as the modified 2PC in ElasTraS.

Another change is on the Distributed Fault-tolerant Storage component. In ElasTraS the DFS is common to all the OTMs. All the persistent writes to the DFS are expensive and if we have a high number of OTMs this component can become a performance bottleneck. In our design there is no need for the DFS to be common to all the OTMs. In fact, the persistency guarantees given by the DFS can now be ensured by the replication mechanism itself. Furthermore, when an OTM crashes, another one is launched and the current state is passed by the existing replicas. This also represents an enhancement in system performance as we avoid expensive DFS writes.

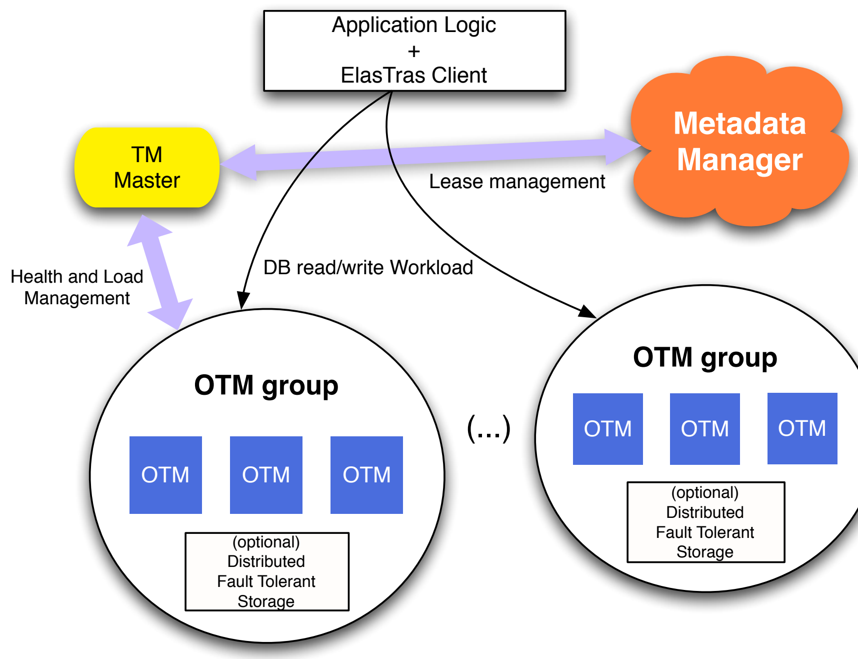

Fig. 2. Our system architecture

ElasTraS was designed to fit two different scenarios. The changes we are introducing will improve the ElasTraS system in both scenarios when considering a read intensive workload. In the first one, where we have several small databases sharing the same data management system resources, ElasTraS assigns each of these small databases to an OTM. The size of the database is therefore limited according to the ability of the single OTM to handle requests on that database. In our design we have each database assigned to a replicated system (OTM group) which leads to performance comparable to that of a replicated RDBMS, which is still the common and effective solution to data management. The second scenario, where a large database is divided into independent partitions, we have a similar situation as the performance per partition is improved. Our system design is depicted in Figure 2.

Although ElasTraS is suitable to the case where we have several small databases or a partitioned large database, there is no solution when we have a database large enough to exceed the single node capacity and which cannot be partitioned. When this is the case, in our system, we can extend the group abstraction and create groups of groups. This allows inter group transactions which means inter partition transactions. The downside is that this has a performance cost. Namely, running a certification protocol over OTM groups can be expensive. However, this empowers the system to accommodate another set of databases. These databases are characterized by having a workload which allows partitioning but still has a small percentage of global transactions, i.e., those that access more than one partition. The architecture of the system incorporating the notion of groups of OTM groups is depicted in Figure 3. In this design, a certain OTM group plays the role of the coordinator and aggregates results from all the different OTM groups as well as guarantees data consistency. This feature is not trivial to implement and definitely requires the OTM groups to support a special transactional call to ensure consistency. Failures of the coordinator must be considered although, in this case, as the coordinator is an OTM group, it is, by definition, already replicated.

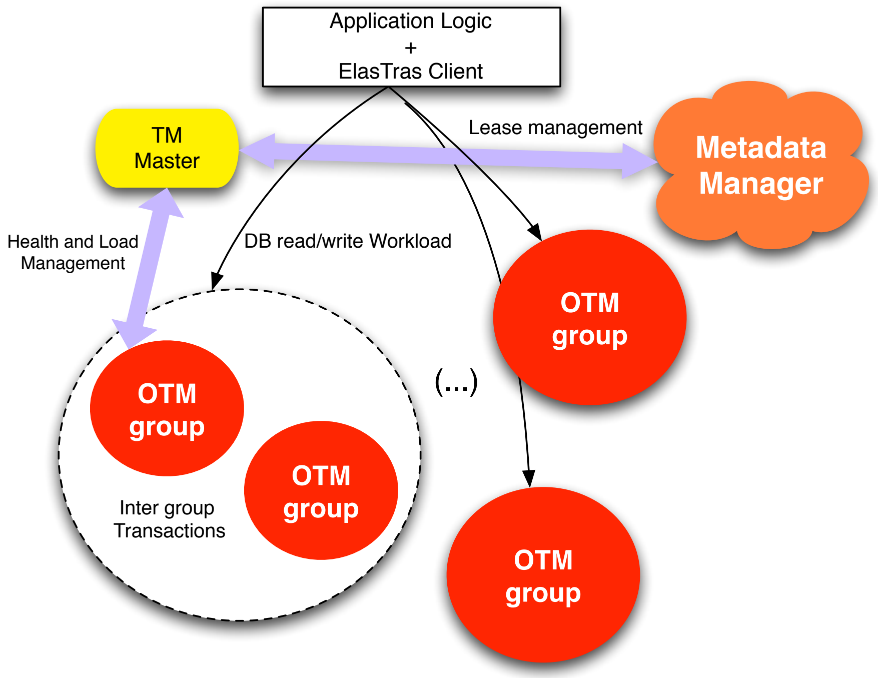

Fig. 3. Our system architecture with the notion of inter partition transactions.

3 Partitioning, the Key to Scalability

Data partitioning is a common method used for improving the performance and the scalability properties of data management systems. The database is divided into smaller pieces called partitions. Partitions are then managed by different system nodes or replicas. This design allows single-partition transactions (those accessing data from a single partition) to run independently from transactions running in other partitions, thus increasing the system throughput. In the cloud computing environment this has been the main approach to provide services with high scalability and availability.

Recently, many applications proved to exhibit a highly partitionable data set. A characteristic example of this are social network applications. These applications allow a high degree of data partitioning (geographic partitioning, partition by user or interests, etc.). Moreover, when social network applications need to access data from different partitions, the specific nature of these applications allows the system to relax consistency guarantees avoiding the impairment of scalability.

Observing the systems described earlier, one can note that data partitioning influences the type of transactions supported by the system. In fact, in these systems, transactional support is guaranteed at the partition level. A certain transaction can only access as much data as the one assigned to a single partition. The way data is partitioned defines the system's ability to scale.

Starting with a single database in a single node how would a dynamic partitioning system be implemented? Such system should have as an input the live value of the load of the system and the value of the single node's capacity to handle load. When the system's load reaches a certain percentage of the total capacity of the system (let's say the typical 70%), the partitioning system will have to launch a new node and assign data partitions to it. The system has to make a decision on how data should be partitioned. The simple case would be ignoring the workload and simply divide data into two partitions. The problem with this approach is that it requires those inter partition transactions that do not scale. Thus this is not a proper approach. The solution is to look at the workload and partition the data according to which data transactions are accessing. The problem is that if we have a transaction accessing all the data in the database the system will not be able to scale.

Data partitioning is crucial to scalability and some proposals have already been made, such as *Schema Level Partitioning* [10], but new partitioning schemas should be studied. Moreover, each data set has its own optimum partitioning schema and we can easily predict it will be hard to find a solution applicable to every case.

Data partition should be also part of data schema design in modern applications with scale requirements. Design techniques should be investigated in order to facilitate the design of database schemas which are inherently scalable.

Finally, it is important to notice that not all applications coexist well with relaxed consistency guarantees. Applications designed for banking, travel agencies, insurance companies, etc., still require data to be strongly consistent. Considering an application with these requirements, it is clear that transactions accessing data on more than one partition (global transactions) will require some kind of synchronization mechanism in order to guarantee data consistency. Synchronization mechanisms weaken the scalability of a system, and thus represent an obstacle to the move of such applications to the cloud. The system we propose in this paper follows the pragmatic approach of assuming a scalability and performance cost when providing inter partition transactions. This enables a new set of applications, those with a high partitioning degree but that require (but not frequently) inter partition transactions, to move to the cloud. Nonetheless, this does not apply to all kinds of applications and there is still work to do in this area.

4 Related Work

Management of distributed transactions [17] has been thoroughly studied in the literature ranging from distributed databases to the particular case of replicated databases [18, 19]. The latter one aims to hide all aspects of data replication from the user. Hence, the replicated database should behave as if it were a stand-alone database system managing all user transactions. This way, when considering serializability, an extended criteria has been proposed, called one-copy serializability (1CS) [18]. It ensures that the interleaved execution of all transactions in the replicated system is equivalent to a serial execution of those transactions on a single database. With the advent of Snapshot Isolation (SI) databases [20] a similar correctness criterion was formalized as 1CSI [19]. It is clear that some kind of coordination is needed to the decide the outcome of a transaction; this is done by a replication protocol. Nevertheless, the scaling factor of these protocols [1] is very limited. On one hand, we have that primary copy solutions [21] do not scale well for update intensive scenarios, apart from the single point of failure, because the maximum throughput is limited by the primary. On the other hand, update everywhere solutions [22, 23] use a total order broadcast facility [24] to generate the one copy history (i.e., update transactions are delivered to all replicas in the same order). It is well-known that total order broadcast does not scale well after a few tens of replicas have been added [25]. Hence, partial database replication protocols seem a good approach to cope with scalability issues [26].

As already mentioned, partitioning means that all replicas do not necessarily store the whole database but part of it. Hence, if transactions can be executed only in one replica, without any kind of interaction with the rest of replicas or keeping it to a minimum, the system may scale up better than full replicated systems. This is specially useful in cloud computing where we want to deploy highly scalable systems while supporting transaction mechanisms. Replicas are owners of one or several partitions, whose assignment can be changed on the fly by a data repository manager, and are entirely responsible for the execution of transactions over that partition. Actually, if the partition can be stored in main memory and the workload consists of transactions whose content can be known in advance (e.g., stored procedures) then transactions can be executed serially in each partition without wasting resources [11, 12]. Nevertheless, if transactions can potentially access several partitions then we have network stalls that have to be reduced to the maximum. One solution is to use *minitransactions* as in Sinfonia [8] or ElasTraS [9], where a modification of the two-phase-commit protocol [18] is proposed in order to make it scalable. In our case, we have taken a more scalable and originally non-blocking solution based on certification protocols for replicated databases [26]. This is the idea behind the notion of an OTM group to increase the performance and fault-tolerance without incurring in the high cost of delivering a total order message outside the cluster or having a number more than reasonable that compromise the scalability inside the cluster.

Another way of overcoming stalls is through the speculative execution of transactions, which is an optimistic approach shown to be useful in [11]. If transactions are decomposed into smaller actions, each one accessing to a different partition, then these actions can be forwarded to the corresponding partitions, where they are serially executed, along with actions coming from other transactions, by a single thread assigned to that partition [12]. The coordination of actions belonging to the same transaction is done with a rendezvous point that decides the outcome of transactions. As we have already mentioned, we prefer to rely on a single total-order delivered message rather than a shared object which can be a single point of failure or storing the set of executed speculative actions to undo them. We believe our approach does not block the execution of transactions in the system and can take all the benefits of these approaches [11, 12].

Finally, the kind of consistency obtained in these cloud environments is a general one-copy multiversion scheduler, since transactions reading from several partitions can have a time (version) wrap. This is due to the fact that there is no guarantee as in 1CSI that transactions get a consistent snapshot of the database. Nevertheless, this is an eventually consistent model [27] as updates will eventually get propagated to all replicas, reaching a consistent state in all of them.

5 Work in Progress

This paper reports on our ongoing implementation of a system that leverages from ElasTraS. The objective is to test it against RDBMS and, if possible, against ElasTraS. Some of the performance measures should help us to compare, for example, the OTM recovery time in ElasTraS with the time our replicated system takes to move transactions from a crashed replica to another one. Another measure would be the time overhead our replication system imposes over transaction execution and try to conclude if the changes we propose behave as expected.

These previous tests should be carried out using different replication mechanisms at the OTM groups. For instance, we are designing and implementing an hybrid replication approach in an OTM group where we can distinguish two subsets of OTMs: primaries and secondaries. The primary OTMs will be in charge of performing certification on behalf of update transactions. Apart from that, they are responsible for asynchronously propagating the updates to a subgroup of secondary OTMs (not all secondary ones), pretty much like in traditional primary copy replication, so that all primary OTMs will cover the whole set of secondary OTMs. The role of secondary OTMs will be to execute read-only transactions; the benefit of this is the elasticity when the system needs to take over new (primary) OTMs for a given OTM group or partition since it already stores most of the up-to-date state of that partition.

We also expect that varying the workload leads to different performances of each of all the different replication mechanisms we have proposed so far in this paper. Alongside the implementation we expect to study suitable workloads and benchmarks in order to be able to fairly compare the different systems [28]. Namely, we expect to use those workloads to research new data partitioning methods and data schema design tools. With these methods and tools designing a scalable system becomes a more straightforward task.

6 Conclusion

Along this paper we described a novel approach to offer transactional support in the cloud environment. Namely, we propose a system aimed at high availability and scalability and still guaranteeing transactional support. The system we proposed is inspired by ElasTraS [10] and presents several enhancements that overcome the problems detected in the original system.

Our system is also elastic and self-managed, characteristics most desirable in a cloud computing scenario. Moreover, the ability to accommodate various independent databases allows the system to be used as a platform service. A client of such a service can deploy their database with a *pay per use* model and without an initial high investment. This aspect has also been regarded as one of the main stimulus to the success of cloud computing [29].

With this paper we also point out the fundamental role partitioning plays in system scalability. In fact, to provide transactional support it is necessary some kind of coordination between partitions or replicas. This coordination can be achieved with many different mechanisms, but it always represents a performance cost and an impairment to scalability. The scalability of a system thus depends on how it will handle data. If we define the size of a transaction based on the number of different tables, relations or partitions it accesses, then the system's scalability will be the maximum when the size of the largest transaction is reduced to the minimum.

We believe that for data management systems it is difficult to provide a system aimed at the famous *one size fits all* model. To achieve maximum system performance and scalability the data set must be studied to find the specific partition schema most appropriate to each case.

Data partitioning, although proved to be crucial to the scalability of a system, must be complemented with novel approaches to providing transactional support in the cloud. There is still plenty of work to be done and we expect our system to be a step forward in this area.

References

1. Gray, J., Helland, P., O'Neil, P.E., Shasha, D.: The dangers of replication and a solution. In Jagadish, H.V., Mumick, I.S., eds.: SIGMOD Conference, ACM Press (1996) 173–182
2. Brewer, E.A.: Towards robust distributed systems (abstract). In: PODC '00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing, New York, NY, USA, ACM (2000) 7
3. Gilbert, S., Lynch, N.A.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News **33**(2) (2002) 51–59
4. von Eicken, T.: Right scale blog: Animotos facebook scale-up. Available at `http://blog.rightscale.com/2008/04/23/animoto-facebook-scale-up/` (2010)
5. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A distributed storage system for structured data. ACM Trans. Comput. Syst. **26**(2) (2008)
6. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazon's highly available key-value store. In Bressoud, T.C., Kaashoek, M.F., eds.: SOSP, ACM (2007) 205–220
7. Cooper, B.F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H.A., Puz, N., Weaver, D., Yerneni, R.: Pnuts: Yahoo!'s hosted data serving platform. PVLDB **1**(2) (2008) 1277–1288
8. Aguilera, M.K., Merchant, A., Shah, M.A., Veitch, A.C., Karamanolis, C.T.: Sinfonia: A new paradigm for building scalable distributed systems. ACM Trans. Comput. Syst. **27**(3) (2009)
9. Das, S., Agrawal, D., Abbadi, A.E.: Elastras: An elastic transactional data store in the cloud. In: HotCloud'09 Workshop at USENIX. (2009)
10. Das, S., Agarwal, S., Agrawal, D., Abbadi, A.E.: Elastras: An elastic, scalable, and self managing transactional database for the cloud. Technical Report UCSB-CS-2010-04, University of California, Santa Barbara (2010)
11. Jones, E.P.C., Abadi, D.J., Madden, S.: Low overhead concurrency control for partitioned main memory databases. In Elmagarmid, A.K., Agrawal, D., eds.: SIGMOD Conference, ACM (2010) 603–614
12. Pandis, I., Johnson, R., Hardavellas, N., Ailamaki, A.: Data-oriented transaction execution. Technical Report CMU-CS-10-101, Carnegie Mellon University (2010)
13. Burrows, M.: The chubby lock service for loosely-coupled distributed systems. In: OSDI, USENIX Association (2006) 335–350
14. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. **16**(2) (1998) 133–169
15. Pedone, F.: The Database State Machine and Group Communication Issues. PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland (1999)
16. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Comput. Surv. **22**(4) (1990) 299–319
17. Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann (1993)
18. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley (1987)
19. Lin, Y., Kemme, B., Jiménez-Peris, R., Patiño-Martínez, M., Armendáriz-Iñigo, J.E.: Snapshot isolation and integrity constraints in replicated databases. ACM Trans. Database Syst. **34**(2) (2009)

20. Berenson, H., Bernstein, P.A., Gray, J., Melton, J., O'Neil, E.J., O'Neil, P.E.: A critique of ANSI SQL isolation levels. In Carey, M.J., Schneider, D.A., eds.: SIGMOD Conference, ACM Press (1995) 1–10
21. Plattner, C., Alonso, G., Özsu, M.T.: Extending DBMSs with satellite databases. VLDB J. **17**(4) (2008) 657–682
22. Lin, Y., Kemme, B., Patiño-Martínez, M., Jiménez-Peris, R.: Middleware based data replication providing snapshot isolation. In Özcan, F., ed.: SIGMOD Conference, ACM (2005) 419–430
23. Elnikety, S., Zwaenepoel, W., Pedone, F.: Database replication using generalized snapshot isolation. In: SRDS, IEEE Computer Society (2005) 73–84
24. Chockler, G., Keidar, I., Vitenberg, R.: Group communication specifications: a comprehensive study. ACM Comput. Surv. **33**(4) (2001) 427–469
25. Jiménez-Peris, R., Patiño-Martínez, M., Alonso, G., Kemme, B.: Are quorums an alternative for data replication? ACM Trans. Database Syst. **28**(3) (2003) 257–294
26. Serrano, D., Patiño-Martínez, M., Jiménez-Peris, R., Kemme, B.: Boosting database replication scalability through partial replication and 1-copy-snapshot-isolation. In: PRDC, IEEE Computer Society (2007) 290–297
27. Vogels, W.: Eventually consistent. Commun. ACM **52**(1) (2009) 40–44
28. Sobel, W., Subramanyam, S., Sucharitakul, A., Nguyen, J., Wong, H., Patil, S., Fox, A., Patterson, D.: Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0. In: 1st Workshop on Cloud Computing (CCA 08). (2008)
29. Gartner: Gartner identifies top ten disruptive technologies for 2008 to 2012. Available at `http://www.gartner.com/it/page.jsp?id=681107` (2010)