An Autonomic Approach for Replication of Internet-based Services [*]

D. Serrano, M. Patiño-Martinez, R. Jimenez-Peris
Universidad Politenica de Madrid (UPM), Spain
{dserrano,mpatino,rjimenez}@fi.upm.es

B. Kemme
McGill University, Canada
kemme@cs.mcgill.ca

Abstract

As more and more applications are deployed as Internet-based services, they have to be available anytime anywhere in a seamless manner. This requires the underlying infrastructure to provide scalability, fault-tolerance and fast response times. While replicating the services and the data they access across sites that are located in different geographic regions is a promising means to achieve these requirements, data consistency is challenging if data continuously changes and queries are dynamic by nature, as is typical for e-commerce applications. Thus, current WAN replication solutions either trade performance for data consistency or are not able to scale in wide-area settings. In this paper, we present a novel approach to provide performance and consistency for Internet services. One of the main contributions is an autonomic replica placement module that places data copies only on servers close to clients that actually need them. The goal is to find the right trade-off between fast local access and the overhead of keeping data copies consistent. As data access patterns might change over time, reconfiguration is done periodically and online, i.e., allowing sites to receive new data copies or drop data copies while at the same time transaction processing continues in the system.

1. Introduction

As Internet-based services become the new standard for customer-to-business and business-to-business interaction, they have to be available anytime anywhere in a seamless manner. This means that services should be accessible despite site failures and disconnections from the Internet. Furthermore, the service infrastructure needs to be scalable to support an ever increasing client base. Finally, in the realm of Internet-based services, one of the most important performance metrics is the response time observed by clients. Clients should have the impression that the service is located locally. Any distribution within the service infrastructure should not impact the responsiveness towards client requests. A standard technique to achieve these properties in data-intensive applications is data replication where data copies are located on servers close to the clients that access them. However, if the data change continuously, if many clients access the same data concurrently, and if queries are dynamic by nature, then data consistency becomes a problem. Many e-commerce applications must provide transactional properties for critical data, and keeping data copies consistent can quickly turn into a nightmare.

A lot of research has been done on data replication, but mostly addresses LAN environments where network latency is not an issue. Some approaches for edge computing can mask the Internet latency by bringing data close to clients. However, they often only deal with static content while dynamic content is only kept at a single data center leading to high latencies and a potential bottleneck. Those approaches that replicate dynamic content, either fail to provide strong consistency and fault tolerance as they propagate data changes lazily to remote copies, i.e., only after transaction commit [22], or fail to provide low response times because they use too stringent correctness criteria such as 1-copy-serializability (1CS) [2], or fail to scale because they are based on full data replication [12].

In this paper, we present a new replication approach to provide seamless availability, scalability and performance to Internet services. Our approach relies on 1-copy-snapshot isolation (1CSI) [13] as correctness criterion. It is nearly as strong as 1CS but has much higher concurrency potential as read operations and their potential conflicts with write operations are much easier to handle. For the basic replica control, we follow the ideas of [12]. Read-only requests can be executed lo-

[*]This work was supported in part by Microsoft Research through its PhD Scholarship Programme, the Spanish National Science Foundation (MEC) under grant TIN2007-67353-C02, Madrid Regional Research Council under the Autonomic project and the EU Commission under the NEXOF-RA project.

cally. Update transactions only require one WAN message exchange, which is the same as many strategies that rely on lazy update propagation.

A main limitation of [12] and many other replication solutions, however, is the use of full replication that limits scalability [21] as all updates have to be performed at all sites. What is more, in the context of WAN, full replication also has the shortcoming of having to propagate every update to every WAN location resulting in heavy-weight traffic. To address this issue, our system supports partial replication. That is, each site only maintains copies of some of the data items incurring a lower update load. Nevertheless transactions can access an arbitrary subset of data items.

A particular challenge lies in providing partial replication in a WAN setting. In here, it is particularly important to decide on where to place copies. In general, we can assume that each geographical location has a local access pattern. For instance, in an e-book store Italian books will be mostly bought by clients in Italy although some access comes from outside Italy. Thus, a server located in Italy should have all data related to Italian books. For other servers a trade-off has to be made between the advantage of having a copy locally and the costs of keeping the copy up-to-date. This problem has received little attention so far and is one of our main contributions. Given some access pattern, we decide on where to place copies of data items. Care is taken that a minimum number of replicas per data item remain in the system at any time.

We also take into account that access patterns might change over time. E.g., requests for books related to a popular TV show will soar in a specific country only when the show goes on air in this country. Access patterns might even change on a daily basis. Replica placement should be able to adjust to these changing access patterns. Our solution automatically self-optimizes, determining periodically the number of replicas and their placement to minimize response time and replication overhead. An important related concern is that traditionally any reconfiguration has been performed offline, i.e., when no transactions were executing. This is incompatible with the availability requirement. Thus, our solution allows receiving or dropping data copies online, while processing transactions. Availability not only requires masking failures of individual sites, it also needs to recover and reintegrate failed sites into the system. We exploit our reconfiguration mechanism for online data transfer to also recover failed sites online. Thus, our system is also self-healing.

The evaluation shows that our approach outperforms static partial replication schemes [21] where data location is decided at start-time, and full replication schemes [12] where all data is replicated at all sites.

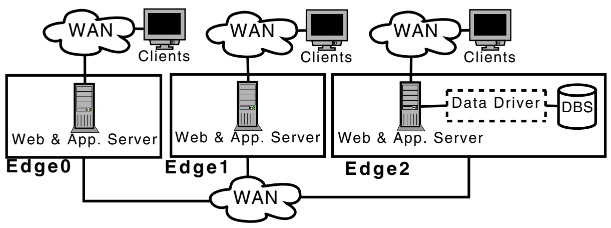


Figure 1. Centralized Architecture

2. Related Work

Traditional edge computing used in content delivery networks (CDN), deploys edge servers around the Internet and replicates static read-only content (see Fig. 1). Each edge server hosts the web and application server tiers and returns static content directly to the clients without the need for WAN communication. Requests to dynamic data are redirected to the central server where the database is hosted often paying the price of one WAN round per update operation. As access to dynamic content starts to prevail in current applications, the central server becomes heavily loaded. In the context of dynamic content, [10] proposes application-aware replication. However, it only considers weak consistency at the object level and not strong consistency at the transaction level. In [22], there is an origin edge server with a full replica of the database, others have partial replicas. The database is split into partitions and each partition has a master. All update transactions for a given data partition are forwarded to its master. If an edge server receives a request for a data partition it does not hold, it forwards it to the origin server. Although this hybrid approach (one full replica plus several partial replicas) scales better than full replication it scales substantially less than pure partial replication (no full replicas) [21]. As replication is performed lazily and different servers are masters for different partitions, the level of consistency provided is fairly low. The approach also does not provide any fault tolerance or autonomic behavior.

[12] presents a full replication approach for edge computing in which full consistency is guaranteed. It uses 1CSI as correctness criterion as we do and provides measures for fault tolerance. The basics of our replica control algorithm are based on [12]. However, we considerably reduce the replication overhead by supporting partial replication. Furthermore, we also support online recovery of nodes and provide a continuous, autonomic and adaptive reconfiguration of the system.

Most other work on database replication focuses on LANs (e.g., [15, 18, 1, 19, 16, 14, 4, 13]). Partial replication has been studied for LANs quite extensively (e.g. [25, 11, 14, 5, 21, 8, 3]). Most consider 1CS as correctness criterion which requires to consider the read operations accessed by transactions. That is complex and can quickly become a scalability bottleneck. Dynamic allocation of replicas is barely considered.

Tashkent+[8] is a database replication middleware designed for LAN environments dealing with partial replication and is based on 1CSI. It uses a centralized load balancer that spreads transactions among cluster sites. The database is partitioned and replicated in such a way that for each transaction type there will be a sufficient number of nodes that have the entire data set used by this transaction type in memory. Therefore, transactions can execute very fast. As workload might change the assignment of nodes to transaction types might be adjusted dynamically. The purpose of partial replication in our context is quite different as our main goal is to locate copies close to where they are needed and not to assign copies to distribute load.

[3] develops a middleware infrastructure extending a standalone in-memory database to a cluster. It provides strong consistency and produces serializable histories. The database is partitioned into data servers, transactions are analyzed and forwarded to the proper data server. If the data server does not have all the data for an operation, the transaction becomes global and is multicast to all the data servers requiring a voting phase for transaction termination. The work also studies load balancing by starting up or shutting down data serves and provides recovery capabilities.

[20] extends the full replication protocol of [18] which is based on 1CS. One of the proposed algorithms uses a weak ordering reliable broadcast [17] and certifies one transaction at a time. The other is based on total order and certifies several transactions at once. Certification requires a quorum-based voting phase. Some of these ideas can also be found in [25, 24]. Our approach works very different. Additionally, we provide a dynamic allocation of replicas.

In [23] the authors present a hybrid between partial and full replication. All data is replicated at all servers. However, transactions are only submitted to a subset of the servers depending on the demand. When the demand changes, less or more of the servers can be provisioned to accept requests. The replicas of the other servers are updated lazily and the machines can be used for other workload.

Dynamo is a distributed replicated storage system used in Amazon that provides eventual consistency [7]. In contrast, our approach is database oriented, and provides strong consistency.

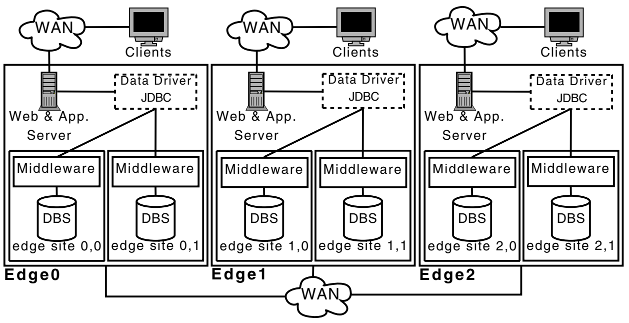


Figure 2. Proposed Architecture

3. System Overview

Overall Architecture Our solution consists of a set of edge servers that are interconnected via the Internet (see Fig.2). Clients connect to the nearest edge server by resorting to DNS-based redirection [9] and then send all requests to this edge server. Each edge server has the standard web and application server tiers that generate the web-pages, cache static websites, and contain the application programs that eventually access the dynamic data stored in the database tier. The possible replication of Web and application tiers is not further considered here as we focus on the database tier. The database tier of each edge server can have several sites, called the *edge sites*. Each of these sites has a partial replica of the database. To the web and application tiers, however, these multiple sites appear as one single database as they access the database tier via a standard JDBC driver. Each edge site runs a standard database system (DBS) providing snapshot isolation (e.g., PostgreSQL), and storing the database replicas. Additionally, each site runs a replication middleware. It intercepts the JDBC requests from the web and application tiers to control the execution at the database tier. The replication logic is embedded in the JDBC driver and the replication middleware.

Replication Consistency Consistency is ensured by an autonomic partial replication protocol based on [12]. Both read-only and update transactions are first executed at any site of any edge server. This enables to execute requests at the server they are submitted to and balance the workload across the sites within one edge server. If data is not replicated locally, the operation is automatically forwarded to a site with the necessary data. When a read-only transaction finishes successfully the transaction commits locally only affecting the sites at which it was executed. For an update transaction, a certification phase is needed to enforce data consistency. The certification phase determines whether the transaction fulfills the snapshot isolation criterion, which requires that no two concurrent transactions may update the same data items. Such conflicts are rare and we detect them at the record level.

Certification in our system is delegated to one of the edge servers, named certifier. Thus, only one edge server pays the cost of certification. Each edge server has a proxy which is the only one to communicate with the certifier. All sites in an edge server send their requests to their local proxy which forwards them to one of the sites at the certifier. All sites at the certifier have to certify transactions according to a common global order to guarantee the determinism of the certification process. For this purpose, total order multicast is used within the certifier edge server as offered by group communication systems (GCS) [6]. Once a transaction is certified, the outcome is communicated to the remaining sites in a hierarchical manner.

Autonomic Replication The autonomic controller takes care of self-optimizing the configuration of the system by monitoring data access patterns and performance and deciding when a reconfiguration is needed. The controller decides how many replicas of each data item are needed and where they should be located. In order to keep the autonomic approach manageable data are grouped in data partitions, and the granularity of replication is a data partition. The autonomic approach is orthogonal to the way data is partitioned. Data can be clustered using some soft computing algorithm (e.g., data clustering or data mining) or can be partitioned according to some information extracted from the application or even provided by the application designer. How to decide on the data partitions is not the scope of this paper.

We consider as reconfiguration the creation or deletion of a data partition. The basic idea is as follows. A local partition, i.e., a partition in an edge, will be deleted if it is accessed rarely by local clients. A partition that is not replicated locally but accessed frequently will be replicated as a local partition to enable local access. Data partitions that are only read or mostly read are replicated everywhere to enable local reads. This autonomic behavior is non-intrusive as data partitions remain accessible by clients even if a reconfiguration of the partition is in process.

4. Autonomic Replication

4.1. Data Replication Protocol

A client sends all its requests to the edge server it is connected to. The application server tier submits operations one at a time via the JDBC driver to the replication middleware of one of the local edge sites.

Using snapshot isolation, the database evolves through a series of snapshots, each one incorporating the updates of a newly committed transaction. These snapshots are sequentially numbered by the replication middleware. When the middleware receives the first operation of a transaction, it assigns to the transaction the current snapshot identifier. The transaction will therefore read from that snapshot. Then, each operation is parsed to identify which data partitions it accesses. If all accessed partitions are local, then the operation is executed at the local database and the results are sent back through the data driver. If some partitions are not local, the operation must be redirected to another site. We discuss redirection in the next sections and assume for now that all operations of the transactions are able to execute locally.

When the local middleware receives a commit request for a read-only transaction, it simply commits the transaction locally. If the transaction updated some data, the middleware gets the set of changes in form of a writeset. Then, the middleware sends the writeset to the local proxy. The local proxy sends all the writesets to a site of the certifier edge server which multicast them in total order within its LAN together its own writesets. Thus, all sites at the certifier edge server receive all writesets in the same total order. Upon receiving a writeset in total order each site runs the certification process for the transaction. The certification process checks that there is no conflict between the writeset of the transaction being certified and the writesets of concurrent transactions that were already certified. A positive outcome is sent together with the writeset to the proxies of all edges containing data accessed by the transaction and then, each local proxy sends it to the other nodes in the edge having affected data partitions. Abort notifications are only sent to the edge where the transaction originated. Certified writesets are sent in FIFO order along the entire communication path to guarantee that all sites apply the writesets in the right order.

4.2. Partition Identification

Partial replication requires being able to determine which data partitions a database operation accesses.

This task is performed in our system by analyzing SQL statements. The middleware at each site keeps the schema of the full database. The information analyzed in the SQL statements includes the tables being accessed, the primary keys, and the values of the attributes that group the partitions (e.g., the language in which a book is written). If the analysis cannot determine the partitions with full certainty, all partitions that potentially could be accessed are considered to be on the safe side. This analysis can be performed at application deployment time or at run-time. The replication method is orthogonal to the way analysis is performed and can work with either the pre- or online analysis technique.

After the partitions being accessed are identified, the middleware decides where to execute the operation. For that, the replication middleware at each site keeps a directory of the data partitions, the list of sites keeping a copy of them and the edge server at which the sites are located. That is, the directory keeps a map of all replicas of all data partitions. The directory also keeps the status of each site, up or down. The directory is updated whenever a partition is added or removed, and sites fail or recover. We can expect the number of updates to this directory to be small. Based on this directory, the middleware first checks whether the partitions needed by the operation are replicated locally. If so the operation is executed. Otherwise, the operation has to be redirected. The middleware checks whether some site in the local edge server has all the partitions, and if not, it checks whether some other edge server has a site with them. If a site is found, the middleware forwards the operation to the middleware of this site where it is executed and the result returned. Thus, the middleware of the original site collects all the information of the transaction such as its writesets.

If no site with all necessary partitions exists, the operation is redirected to the site with the highest number of partitions accessed by the operation. This site then sends sub-requests to other sites to retrieve the missing data, and joins the data. We expect such distributed operation execution to occur seldom. It might occur, for instance, for some infrequent administrative transactions accessing a large number of partitions. In such case, we believe it is better to have fairly expensive distributed operations than having sites with all necessary data partitions. Guaranteeing for each operation to have a site with all necessary partitions requires a priori and perfect knowledge of the partitions accessed by each transaction. Furthermore, in the worst case, it might require sites with nearly a full copy of the database. Such site could quickly become a bottleneck since it has to apply all updates on all the data.

4.3. Guaranteeing 1CSI

1CSI requires that transactions see a globally consistent snapshot of the system [13]. This is tricky in a partially replicated system in which transaction operations might be redirected since all sites at which the transaction executes should provide the same snapshot. We apply the technique of dummy transactions proposed in [21]. After a transaction commits at a site, reflecting a new snapshot, a set of dummy transaction is started. In this way, a redirected operation can be associated with a dummy transaction that started with the same snapshot as in the original site. Starting dummy transactions has very little overhead at the database, and unused dummy transactions are garbage collected when it is known that no transaction will need them. For this purpose, writeset messages piggyback the start timestamp of the oldest transactions being run at each site. The certifier sites can determine the oldest transaction run in the system and inform the other sites piggybacking the information on certified writesets.

4.4. Self-Optimization: Overview

This section describes the self-optimization algorithm used to decide on the number of replicas of each data partition and their placement. The self-optimization algorithm is completely orthogonal to the replication protocol. At system start-up, each edge server starts transaction execution with a full copy of the database. Throughout the execution, statistics about the accessed data are collected. The self-optimization algorithm is triggered periodically.

In the following we denote with $P = \{p_1..p_k\}$ the set of data partitions and with $E = \{e_1..e_m\}$ the set of edge server sites. The self-optimization algorithm is organized into five phases:

1. **Evaluation phase.** In the evaluation phase, the algorithm evaluates for each partition p_i and edge e_j whether p_i should have a replica at e_j. This is done with the help of a matrix $J : (P \times E)$ where each element $J(p_i, e_j)$ is the evaluation of the data partition p_i at edge e_j. The evaluation could be, e.g., the number of operations since the last reconfiguration that accessed partition p_i and originated at edge e_j. A decision function f takes as input J and returns a transformed matrix $F : (P \times E)$ where each entry $F(p_i, e_j)$ has the value $true$ if the partition p_i should have a replica at edge e_j, and $false$ otherwise. An example of a decision function returns $true$ if the number of local accesses to a partition is bigger than a given minimum threshold, otherwise it returns $false$. The decision func-

tion can also take into account the read/update nature of accesses to data partition.

2. **Allocation phase.** The allocation phase now determines where partitions should be dropped, where partitions remain where they are, and where new partition replicas have to be added. The allocation takes the matrix F and the current replica placement as input but, also considers a replication degree r_i which indicates the minimum number of copies that p_i should have for fault-tolerance purposes. The result is a matrix $S : P \times E \rightarrow \{propagate, drop, no - action\}$. $S(p_i, e_j) = propagate$ means that the partition p_i should be propagated to edge site e_j. $(p_i, e_j) = drop$ means that the partition p_i should be dropped at edge site e_j. $(p_i, e_j) = no - action$ means no modification of p_i at edge e_j is needed.

3. **Propagation phase.** Once it is decided where to allocate each data partition replica, the algorithm triggers the propagation of the partitions identified in the previous phase. The partition propagation is performed in a non-intrusive manner without stopping transaction processing whilst guaranteeing data consistency. The propagation guarantees that the new site gets a correct snapshot of the partition and all the subsequent snapshots. It also guarantees that transactions reading the new partition replica run on the right snapshot of the data.

4. **Validation phase.** In the validation phase the system checks whether the adaptation has had the foreseen performance improvement. The algorithm analyzes whether system performance has improved by monitoring several performance parameters, e.g., response time. Based on this, the algorithm decides whether the adaptation decisions were right or wrong.

5. **Adaptation phase.** In this phase, those adaptation decisions identified as wrong are reverted and thresholds adapted to prevent taking the wrong decision in the next execution of the protocol.

4.5. Self-Optimization: Details

The autonomic controller within the middleware implements the self-optimizer algorithms. Each edge site e_j collects local statistics. For each partition p_i, it keeps track of $J(p_i, e_j)$, which in our case is the number of accesses to partition p_i made by transactions submitted to e_j. Furthermore, the average response times for operations executing on a certain partition are measured. It can be expected that the response times are larger if the partition is not locally replicated.

The self-optimization algorithm is executed every time t. The *evaluation phase* is done in a distributed fashion. Site e_j maintains for each partition p_i a threshold value $T_j(p_i)$. Threshold values for different partitions can be different, and the threshold values of different sites for the same partition can also be different. Time period t and thressholds can be defined experimentally or autonomically. Site e_j now checks if $J(p_i, e_j)$ is below $T_j(p_i)$. If this is the case, e_j determines $F(p_i, e_j)$ to be $false$, i.e., the partition should not be replicated, otherwise *true*. Then it sends both the vectors $F(P, e_j)$ and $J(P, e_j)$ to its local proxy. The local proxy collects all vectors and sends them to a central site in the certifier edge. This site collects all information and builds the global matrix F.

From there, the central site at the certifier edge performs the *allocation phase* and generates matrix S. First, by looking at its partition directory, it determines for each partition p_i the current number of replicas $current_i$. This considers only replicas at sites that are currently up and running. Then, based on matrix F and the current configuration, it sets add_i to be the number of replicas that should be added and $remove_i$ to be the number of replicas that should be removed. For each partition p_i, the candidate replicas to be removed are ranked according to the $J(p_i, e_j)$ values. If the number of partition replicas that will remain is bigger than r_i, $current_i - remove_i \geq r_i$, then the central site tells all sites that want to remove the partition replica to do so. If $current_i - remove_i < r_i$ but $current_i - remove_i + add_i \geq r_i$, i.e., the final number of replicas is bigger than r_i but removing all requested replicas immediately would lead to a shortage, then the dropping of partition replicas is delayed until after the creation of the new replicas is finalized. Finally, if the final number of copies of data partitions is below r_i, i.e., $current_i - remove_i + add_i < r_i$, only $r_i - current_i - add_i$ copies of the partitions will be removed after the creation of new copies (if any) is completed. In the last case, the partitions that will be removed will be those with the smallest number of local accesses. The matrix S is now populated and multicast to all sites via their proxies.

Given S each site updates its partition directory to keep track of all replicas of a partition. Furthermore, each site knows who has to get new partition replicas. This starts the *propagation phase*. For a site e_j that should add a replica of partition p_i, it will receive the partition from the closest site (in terms of latency). Once the propagation of partitions is finished (see next section), local execution on these partitions can start.

During the *validation phase* each site checks how response times behave for each p_i that was created. If the response time does not improve, then it means that replicating the data locally has not paid off. During the *adaptation phase*, the edge site decreases $T_j(p_i)$ so that p_i is dropped in the following execution of the protocol and not created again. This avoids that partitions are continuously created and dropped when their access proportions are around the threshold.

4.6. Online Reconfiguration

The reconfiguration of partitions can be activated due to several events: the execution of the self-optimization algorithm, recoveries of failed sites, or reconnection of edge servers disconnected from the Internet. In order to provide partition replicas to sites, all sites keep a log of all applied writesets. All writesets are tagged with the transaction commit timestamp (the counter of committed transactions in the system).

We can distinguish two reconfiguration cases. A site that wants to have a new replica of p_i might already have an outdated copy of the partition, but this copy is more or less recent. This occurs, e.g., when a site recovers shortly after it failed, or an edge server reconnects after a disconnection from the Internet. Alternatively, the site might not have any copy of partition p_i or a very outdated copy. If the site does not have a relatively recent copy, the first action that it is performed is transferring a checkpoint of the data partition (obtained by means of a SELECT statement). The checkpoint also contains the commit timestamp of the latest committed transaction before obtaining the checkpoint. This means, the transferred checkpoint contains all changes up to this last committed transaction. Note that the site may continue executing transactions on its partition replica while reading the checkpoint and transferring it. The updates of these transactions will not be included in the checkpoint as the query reads only committed values as of start time due to snapshot isolation. If the new site has a relatively recent copy, it identifies the commit timestamp of the last transaction registered in the local log. Again, this means that its partition replica contains all changes up to this last committed transaction. In both cases, the sender site now scans the writesets in its log and sends those updates on the partition missed by the new site. The new site applies the updates to the partition replica and also logs the writeset in its own log.

When the sender reaches the end of the log then it sends the timestamp of the last writeset sent to a certifier site and the new site. The certifier site then takes care of sending the writesets produced in the meantime and determining the end of the reconfiguration, registering the new copy of the data partition in the data partition directory by multicasting a configuration message indicating this fact in the local edge and to all proxies which forward it. From then on, all sites know that existence of this new partition replica and can redirect operations to it, if necessary.

4.7. Failures

Failures are handled differently depending on whether a certifier site, a proxy site or a regular site fails. All sites within an edge server communicate via a group communication system. Thus, within an edge, each site can detect failures of other sites via the GCS.

Whenever a site fails, its client connections (i.e., connections of the application server tier to this site) are redirected to another site in the same edge server. When a regular site fails, the proxy forwards the failure notification to the other edge servers (via the certifier edge). All sites mark the site as failed in their partition directory. This information is taken into account in the next reconfiguration cycle to possibly create new replicas of the data partition (typically at the edge server where the failed site belonged to). All transaction active at the failed site and not yet in the process of being certified are aborted. Sites that redirected operations to this site will detect its failure via the TCP/IP link. If no updates were performed in the failed site on behalf of a transaction the next operations can be redirected to another site. Otherwise the transaction must abort.

The failure of a proxy involves more processing. One of the remaining sites at the edge server takes over as proxy. As the proxy is responsible for forwarding writesets to the certifier and receiving outcomes from the certifier, the status of each outstanding transaction has to be reconstructed (see [12] for details). It is ensured that each writeset is certified exactly once and applied if successful. Furthermore, also partition reconfigurations and site reconfigurations are disseminated through the system via proxies. The reconfiguration process ensures that these messages are eventually received by all despite the proxy switch.

For the certifier edge, the first important observation is that all available sites perform certification of all transactions providing fault-tolerance for the certification process. The failure of an individual certifier might result in the loss of some transactions that were submitted for certification to this site but not yet multicast to the rest. The proxy will detect the failure of the link with its certifier site and reconnect to another site. It is ensured that any outstanding transaction is certified exactly once and applied if successful.

If a failure occurs during the reconfiguration process we can distinguish two cases. If the site sending the checkpoint fails, a new one is searched to restart the reconfiguration process. If the new site had already received a checkpoint or writesets from the old one, they do not need to be resent. If the new site fails, then the reconfiguration is simply dismissed. The next optimization cycle will take the pertinent measures.

For Internet disconnections of edge servers, we adopt a primary-component approach. The certifier edge server acts as primary component. Therefore, it can always progress. If a non-certifier edge server cannot communicate with the certifier edge, it cannot process update transactions. Read-only transactions can run on the current state of the edge server possibly reading stale data, if this is acceptable for the application. Otherwise, also read-only transactions are blocked until connection with the certifier edge is reestablished.

5. Evaluation

Our system consists of three edge servers. Each server has a site running the web and application server tier (Tomcat) and two sites for the replicated database with our replication middleware interposed that provides its own standard JDBC driver. We use PostgreSQL as database. The edge servers are connected through an emulated WAN with a latency of 150ms. Sites within the edge server are LAN connected.

It has been shown in the past that WAN replication is superior to approaches with one central database [12]. For this reason, we choose as our first baseline comparison the WAN replication approach from [12] (labeled as *Full* in the figures). This approach is based on full replication and is not autonomic. It provides the most performant approach we are aware of and a similar degree of fault-tolerance. Our replica control algorithm is based on that of [12]. Thus, a comparison with it also allows us to better understand the impact of partial replication and adaptability. As a second baseline we choose our proposed approach without the autonomic component (labeled as *NoAdapt*). This baseline will help us to demonstrate the need for the autonomic behavior to really benefit from partial replication in Internet-based systems. The third configuration is our proposed approach providing adaptive replication (labeled as *Adapt*).

5.1. Experimental Setup

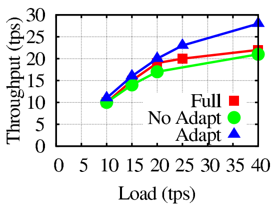
We have evaluated our system with the TPC-W benchmark[26] which simulates an online bookstore and was designed to evaluate transactional Web-based systems, i.e., exactly those systems for which we have designed our adaptive replication tool. The performance metrics reported by TPC-W is the number of processed web interactions per second. Each web interaction can involve one or more transactions. TPC-W offers three workload profiles by varying the ratio of browsing and ordering transactions. In our evaluations we chose the ordering mix with 50% browsing and 50% shopping as it has the highest update rate, and thus, provides a good stress test for our system.

The database is split into 92 data partitions. Six of these partitions are accessed by 80% of the transactions. The other partitions are accessed by the remaining 20% of the transactions. This distribution represents the typical access distribution of hot spots (80% of the transactions access 5-10% of the data) that are the most challenging workloads to deal with. At each edge server, 80% of the local transactions access two of the six highly accessed partitions and 20% access the remaining partitions. Each edge server accesses two different highly accessed partitions representing the locality found in Internet applications.

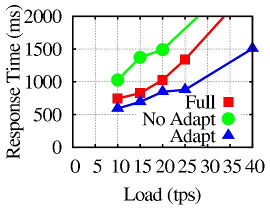
There are 100 clients (100 RBE's) evenly distributed among the edges. Edge server 0 is the certifier edge. Each experiment has three phases, a warm up phase of 2 minutes, a measurement phase of 4 minutes and a cold down phase of half minute. The autonomic controller collects statistics every half minute. In order to show the benefits of the self-optimization aspect, in the NoAdapt and Adapt configurations edge 1 does not have copies of the two locally highly accessed data partitions. The non-highly accessed data partitions are evenly split among replicas. This means that the Adapt configuration will automatically create copies of the highly accessed partitions by recovering them from a remote edge server. The Adapt configuration was also tried starting with a full replication configuration yielding the same results as the configuration reported here. That is, the self-adaptation yielded an equivalent final configuration independently of the initial one.

5.2. System Performance

Our first test evaluates the system performance by showing throughput and response time results at the certifier edge. The results at the non-certifier edges were similar and omitted here. Fig. 3(a) shows the throughput obtained for increasing loads. We can see that the full and non adaptive configurations exhibit the poorest performance reaching a maximum throughput of 20 transactions per second (tps). The adaptive replication reaches almost 30 tps, a nearly 50% higher



(a) Throughput

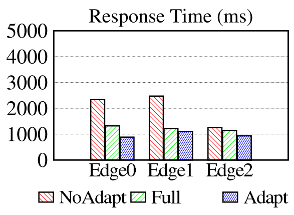


(b) Response Time

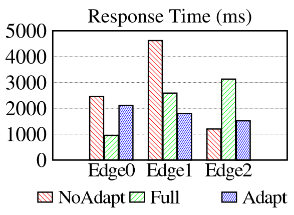
Figure 3. Performance

throughput. This demonstrates the superior performance of our approach. The enhanced performance of the adaptive replication over the full replication is due to the differences in replication overhead. Full replication approach has a high replication overhead since all sites need to apply all writesets produced in the whole system. In contrast, in the adaptive replication approach each site only processes a small fraction of all the writesets. Thus, a lower fraction of the site capacity is lost due to replication overhead and more of the capacity can be used to execute additional transactions. Thus, it has a higher throughput peak than full replication. The experiment also shows that self-optimization has better performance problems. The reason is that it adapts the configuration of partial replicas to exploit locality in edge 1, whilst the non-adaptive configuration suffers a bad performance due to the high number of redirections needed at E_1. That is, if actual locality does not match the partial replica configuration a lot of redirections are needed. This results in a high redirection overhead reducing the peak throughput.

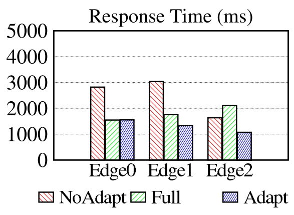
Fig. 3(b) shows the average response time over all transaction types. The non-adaptive configuration shows the worst response time, providing response times of 1 second for the lowest injected loads and increasing very fast with increasing load going beyond 1.5 seconds for loads above 20 tps. The full replication configuration behaves slightly better, providing response times below 1.5 seconds until it saturates at 25 tps. Adaptive replication provides response times below 1.5 seconds for loads up to 40 tps, a significant improvement over the other two approaches. If we look at the response time for small loads, without saturating the system, we can observe that the response times are similar for adaptive replication and full replication. This means that the adaptive replication is able to provide local access for most transactions as does the full replication approach. For higher loads, the full replication approach shows higher response times than adaptive replication as the system gets earlier saturated due to the higher replication overhead.



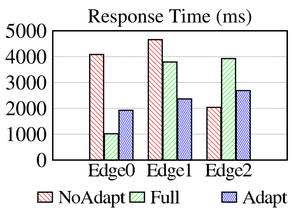
(a) execute search, peak load



(b) buy confirm, peak load



(c) execute search, saturation



(d) buy confirm, saturation

Figure 4. Web interaction response time

The reason of the higher response times of non-adaptive replication compared to adaptive replication is the fact that edge E_1 needs to redirect most of the transactions to other edge servers. These redirections introduce a severe communication delay increasing substantially the response time. Additionally, redirection consumes processing capacity, overloading the system faster which increases the response time further.

5.3. Customer Experience

In this section we profile the response time per type of web interaction. There are 14 different transaction types specified in TPC-W. In here, we have selected two types with very different profiles. The first one ("execute search") is a heavy read only transaction. The second one ("buy confirm") is an update transaction that updates several tables. We show the results for a load with a peak throughput (20 tps) and a load leading to saturation (25 tps).

For the peak load, the first observation is that the non-adaptive replication has very high response times at edge 1 for all transaction types. The reason is that there are two partitions highly accessed by local clients that are not locally replicated. These requests are redirected to edge 0 that has copies of the partitions.

Therefore, transactions executed at edge 1 exhibit high response times due to the redirections and edge 0 gets overloaded with the additional load coming from edge 1. In contrast, the adaptive counterpart creates local replicas at edge 1 of the locally highly accessed partitions avoiding this situation. Let us now compare full replication with adaptive replication. The response times for "execute search" with adaptive replication are better than with full replication. The main reason is that the heaviness of this query makes the differences in replication overhead between the two approaches visible. "Buy confirm" is a little different due to its update nature. At edge 1 and 2 the results are in favor of adaptive replication due to the smaller number of writesets processed by each replica compared to full replication. However, at edge 0, adaptive replication is worse than full replication probably due to the increased certification overhead. However, the response time of adaptive replication is still quite good.

When looking at a higher load, the advantages of adaptive replication compared to full replication become even more obvious. Full replication already reaches its saturation point with 25 tps and thus, response times deteriorate. In contrast, adaptive replication is able to scale better due to the reduced replication overhead, and thus keeps the response times low, similar to response times at 20 tps.

6. Conclusions

We have presented an integral approach to replication for Internet based applications. This approach addresses the strict requirement of new paradigms emerging in the last few years such as software as a service in which services should remain available anytime anywhere and provide strong transactional consistency. Our system shows that it can provide a response time similar to the best approach of the current state of the art based on full replication, while providing better scalability thanks to partial replication. The self-optimization component of our system plays a key role to outperform full replication.

References

[1] C. Amza, A. L. Cox, and W. Zwaenepoel. Distributed versioning: Consistent replication for scaling back-end databases of dynamic content web sites. In *Middleware'03*.

[2] P. Bernstein, V. Hadzilacos, et al. *Concurrency Control and Recovery in Database Systems*. 1987.

[3] L. Camargos et al. Sprint: a middleware for high-performance transaction processing. In *EuroSys'07*.

[4] E. Cecchet, J. Marguerite, and W. Zwaenepoel. C-JDBC: Flexible database clustering middleware. In *USENIX'04*.

[5] E. Cecchet, J. Marguerite, and W. Zwaenepoel. Partial replication: Achieving scalability in redundant arrays of inexpensive databases. In *OPODIS'03*.

[6] G. V. Chockler, I. Keidar, and R. Vitenberg. Group Communication Specifications: A Comprehensive Study. *ACM Computer Surveys*, 33(4), Dec. 2001.

[7] G. DeCandia et al. Dynamo: amazon's highly available key-value store. In *SOSP'07*.

[8] S. Elnikety, S. Dropsho, and W. Zwaenepoel. Tashkent+: memory-aware load balancing and update filtering in replicated databases. In *EuroSys'07*.

[9] Z. Fei et al. A novel server selection technique for improving the response time of a replicated service. In *INFOCOM'98*.

[10] L. Gao et al. Application specific data replication for edge services. In *WWW'03*.

[11] J. Holliday et al. Partial database replication using epidemic communication. In *ICDCS'02*.

[12] Y. Lin et al. Enhancing edge computing with database replication. In *SRDS '07*.

[13] Y. Lin et al. Middleware based data replication providing snapshot isolation. In *SIGMOD'05*.

[14] E. Pacitti, C. Coulon, P. Valduriez, and M. T. Özsu. Preventive replication in a database cluster. *Distributed and Parallel Databases*, 18(3), 2005.

[15] E. Pacitti and E. Simon. Update propagation strategies to improve freshness in lazy master replicated databases. *VLDB J.*, 8(3-4):305–318, 2000.

[16] M. Patiño-Martínez et al. MIDDLE-R: Consistent database replication at the middleware level. *ACM Trans. Comput. Syst.*, 23(4), 2005.

[17] F. Pedone et al. Solving agreement problems with weak ordering oracles. In *EDCC'02*.

[18] F. Pedone et al. The database state machine approach. *Distributed and Parallel Databases*, 14(1), 2003.

[19] C. Plattner and G. Alonso. Ganymed: Scalable replication for transactional web applications. In *Middleware'04*.

[20] N. Schiper, R. Schmidt, and F. Pedone. Optimistic algorithms for partial database replication. In *OPODIS'06*.

[21] D. Serrano, M. Patiño-Martínez, R. Jiménez-Peris, and B. Kemme. Boosting database replication scalability through partial replication and 1-copy-snapshot-isolation. In *PRDC'07*.

[22] S. Sivasubramanian, G. Alonso, G. Pierre, and M. van Steen. Globedb: autonomic data replication for web applications. In *WWW '05*.

[23] G. Soundararajan et al. Reactive provisioning of back-end databases in shared dynamic content server clusters. *ACM Transactions on Autonomous and Adaptive Systems*.

[24] A. Sousa et al. Evaluating certification protocols in the partial database state machine. In *ARES'06*.

[25] A. Sousa, R. Oliveira, F. Moura, and F. Pedone. Partial replication in the database state machine. In *NCA'01*.

[26] Transaction Processing Performance Council. *TPCW Benchmark*. http://www.tpc.org/tpcw.