GlobLease: A Globally Consistent and Elastic Storage System using Leases

Ying Liu, Xiaxi Li and Vladimir Vlassov
Department of Software and Computer Systems
KTH Royal Institute of Technology, Stockholm, Sweden
{*yinliu, xiaxi, vladv*}*@kth.se*

***Abstract*—Nowadays, more and more IT companies are expanding their businesses and services to a global scale, serving users in several countries. Globally distributed storage systems are employed to reduce data access latency for clients all over the world. We present GlobLease, an elastic, globally-distributed and consistent key-value store. It is organised as multiple distributed hash tables (DHTs) storing replicated data and namespace. Across DHTs, data lookups and accesses are processed with respect to the locality of DHT deployments. We explore the use of leases in GlobLease to maintain data consistency across DHTs. The leases enable GlobLease to provide fast and consistent read access in a global scale with reduced global communications. The write accesses are optimized by migrating the master copy to the locations, where most of the writes take place. The elasticity of GlobLease is provided in a fine-grained manner in order to precisely and efficiently handle spiky and skewed read workloads. In our evaluation, GlobLease has demonstrated its optimized global performance, in comparison with Cassandra, with read and write latency less than 10 ms in most of the cases. Furthermore, our evaluation shows that GlobLease is able to bring down the request latency under an instant 4.5 times workload increase with skewed key distribution (a zipfian distribution with an exponent factor of 4) in less than 20 seconds.**

***Keywords*-Distributed Storage, Elasticity, Geo-replication**

I. Introduction

With the increasing popularity of Cloud computing, as an essential component of it, distributed storage systems have been extensively used as backend storages by most of the cutting-edge IT companies, including Microsoft, Google, Amazon, Facebook, LinkedIn, etc. The rising popularity of distributed storage systems is mainly because of their potentials to achieve a set of desired properties, including high performance, data availability, system scalability and elasticity. However, achieving these properties is not trivial. The performance of a distributed storage system depends on many factors including load balancing, replica distribution, replica synchronization and caching. To achieve high data availability without compromising data consistency and system performance, a set of algorithms needs to be carefully designed, in order to efficiently synchronize data replicas. The scalability of a distributed storage system is achieved through the proper design of the system architecture and the coherent management of all the factors mentioned above. Some of the state of the art systems achieving most of the above desire properties are presented in [1], [2], [3], [4].

System performance can be largely leveraged when using replication. Replication provides a system to handle workload simultaneously using multiple replicas, thus achieving higher system throughput. Furthermore, intuitively, the availability of data is increased by maintaining multiple copies in the system. However, replication also brings a side-effect, which is the maintenance of replica consistency. Consistency maintenance among replicas imposes an extra communication overhead in the storage system that can cause the degradation of the system performance and scalability. This side-effect is even more obvious when the system is geo-replicated, where the communications among replicas might experience relatively long latency. Furthermore, since the storage service is a stateful service, the elasticity of a distributed storage system is extremely hard to achieve. Specifically, the elasticity of a storage system cannot be achieved only by adding or removing storage servers. The state (data) need to be replicated or reallocated, which introduces a significant data movement overhead.

We consider the following scenario. Assume a large scale service with clients distributed in a global scale, where the majority of them are readers. It is often the case that the popular contents attract significant percentage of readers. Typically, the workload increase caused by the popular contents is usually spiky and not long-lasting. Typical applications include Wikis, WEB 2.0 and social network applications. A well-known incident was the death of Michael Jackson, when his profile page attracted a vast amount of readers in a short interval, causing a sudden spiky workload. In order to efficiently and effectively handle such skewed and spiky workload, we propose GlobLease, a consistent and elastic storage system that can be deployed in a global scale. It achieves low latency read accesses in a global scale and efficient write accesses in one area with sequential consistency guarantees. Fine-grained elasticity with reduced data movement overhead is integrated in GlobLease to handle the popular contents (spiky and skewed workload).

The contributions of this work are as follows.

- We explore the use of multiple DHTs for geo-replication, which implements geo-aware routing.
- We propose a lease-based consistency protocol, that shows high performance for global read and regional write accesses by reducing the global communication.
- We provide the fine-grained elasticity of GlobLease using affiliated nodes, that enables the efficient handling

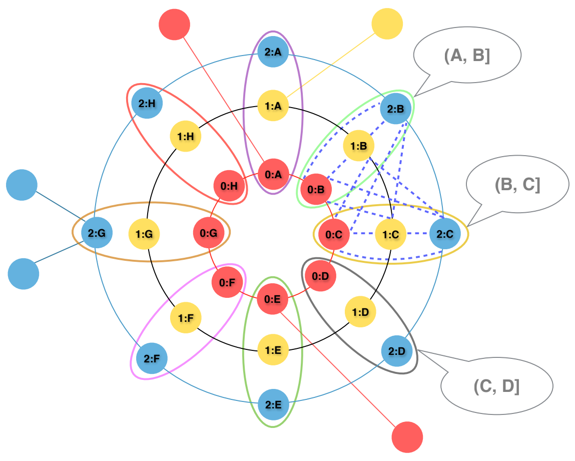

Figure 1. GlobLease system structure having three replicated DHTs

of spiky and skewed workload.

- We evaluate the geo-performance and fine-grained elasticity of GlobLease in comparison with Cassandra in Amazon EC2.

II. System Architecture

We assume that readers have some knowledge of DHTs and are familiar with the concepts of availability, consistency and scalability in a distributed storage system. The background knowledge can be obtained in [5], [6], [7].

GlobLease is constructed with a configurable number of replicated DHTs shown in Fig. 1. Each DHT maintains a complete replication of the whole data. This design provides flexibility in replica distribution and management at a global scale. Specifically, GlobLease forms up replication groups across the DHT rings, which scales out the limitation of successor list replication [8]. Multiple replicated DHTs can be deployed in different geographical regions in order to improve data access latency. Building GlobLease with DHT-based overlay provides it with a set of desirable properties, including self-organization, linear scalability, and efficient lookups. The self-organizing property of DHTs allows GlobLease to efficiently and automatically handle node join, leave and failure events using pre-defined algorithms in each node to stabilize the overlay [5], [9]. The peer-to-peer (P2P) paradigm of DHTs enables GlobLease to achieve linear scalability by adding/removing nodes in the ring. One-hop routing can be implemented for efficient lookups [2].

A. Nodes

Each DHT Ring is given a unique ring ID shown as numbers in Fig. 1. Nodes illustrated in the figure are virtual nodes, which can be placed on physical servers with different configurations. Each node participating in the DHTs is called a standard node, which is assigned a node ID shown as letters in Fig. 1. Each node is responsible for a specific key range starting from its predecessor's ID to its own ID. The ranges can be further divided online by adding new nodes. Nodes that replicate the same keys in different DHTs form *the replication group*. For simple illustration, the nodes form the replication group shown within the ellipse in Fig. 1 are responsible for the same key range. However, because of possible failures, the nodes in each DHT ring may have different range configurations. Nodes that stretch outside from the DHT rings in Fig. 1 are called *affiliated nodes*. They are used for fine-grained management of replicas, which are explained in Section IV-A. GlobLease stores key-value pairs. The mappings and lookups of keys are handled by consistent hashing of DHTs. The values associated with the keys are stored in the memory of each node.

B. Links

1) Basic Links: Links connecting a node's predecessor and successor within the same DHT are called *local neighbour links* shown as solid lines in Fig. 1. Links that connect a node's predecessors and successors across DHTs are called *cross-ring neighbour links* shown as dashed lines. Links within a replication group are called *group links* shown as dashed lines. Normally, routings of requests are conducted with priority choosing local neighbour links. A desired deployment of GlobLease assumes that different rings are placed in different locations. In such case, communications using local neighbour links are much faster than using cross-ring neighbour links. Cross-ring neighbour is selected for routing when there is failure in the next hop local neighbour.

The basic links are established when a standard node or a group of standard nodes join GlobLease. The bootstrapping is similar to other DHTs [5], [9] except that GlobLease needs to update cross-ring neighbour links and group links.

2) Routing Links: With basic links, GlobLease is able to conduct basic lookups and routings by approaching the requested key hop by hop. In order to achieve efficient lookups, we introduce the routing links, which are used to reduce the message routing hops to reach the responsible node of the requested data. In contrast to basic links, routing links are established gradually with the processing of requests. For example, when node A receives a data request for the first time, which needs to be forwarded to node B, the request is routed to node B hop by hop using basic links. When the request reaches node B, node A will get an echo message regarding the routing information of node B including its responsible key range and ip address. Finally, the routing information is kept in node A's routing table maintained in its memory. As a consequence, a direct routing link is established from node A to node B, which can be used for the routings of future requests. In this way, all nodes in the overlay will eventually be connected with one-hop routing. In failure scenarios, if some links are not reachable, they will be removed from the routing table.

III. Lease-based Consistency Protocol

In order to guarantee data consistency in replication groups across DHTs, a lease-based consistency protocol is designed. Our lease-based consistency protocol implements sequential consistency model and is optimized for handling global read-dominant and regional write-dominant workload.

A. *Lease*

A lease is an authorization token for serving read accesses within a time interval. A lease is issued on a key basis. There are two essential properties in the lease implementation. First is authorization, which means each replica of the data that has a valid lease is able to serve the read access for the clients. Second is the time bound, which allows the lease itself to expire when the valid time period has passed. The time bound of lease is essential in handling possible failures on non-masters. Specifically, if an operation requires the update or invalidate of leases on non-masters, which cannot be completed due to failures, the operation waits and can proceed when the leases are expired naturally.

B. *Lease-based Consistency*

We assign a master on a key basis in each replication group to coordinate the lease-based consistency protocol among replicas. The lease-based protocol handles read and write requests as follows. Read requests can be served by either the master or any non-masters with a valid lease of the requested key. Write requests have to be routed to and only handled by the master of the key. To complete a write request, a master needs to guarantee that leases associated with the written key are either invalid or properly updated together with the data in all the replicas. The validities of leases are checked based on lease records, which are created on masters whenever a lease is issued to a non-master. The above process ensures the serialization of write requests in masters and no stale data will be provided by non-masters, which complies the sequential consistency guarantee.

C. *Lease Maintenance*

The maintenance of the lease protocol consists of two operations. One is lease renewals from non-masters to masters. The other one is lease updates issued by masters to non-masters. Both lease renewals and updates need cross-ring communications, which are associated with high latency in a global deployment of GlobLease. Thus, we try to minimize both operations in the protocol design.

A lease renewal is triggered when a non-master receives a read request while not having a valid lease of the requested key. The master creates a lease record and sends the renewed lease with updated data to the non-master upon receiving a lease renewal request. The new lease enables the non-master to serve future reads of the key in the leasing period.

Lease update of a key is issued by the master to its replication group when there is a write to the key. We currently provide two approaches in GlobLease to proceed with lease updates. The first approach is *active update*. In this approach, a master updates leases along with the data of a specific key in its replication group whenever it receives a write on that key. The write is returned when the majority of the nodes in the replication group are updated. This majority should include all the non-masters that still hold valid leases of the key. Write to the majority in a replication group guarantees the high availability of the data. The other approach is *passive update*. It allows a master to reply to a write request faster when a local write is completed. The updated data and leases are propagated to the non-masters asynchronously. The local write is applicable only when there are no valid leases of the written key in the replication group. In case of existing valid leases in the replication group, the master works as the active update. In this way, passive update also keeps sequential consistency guarantee.

Active update provides the system with higher data availability, however, it results in worse write performance because of cross-ring communication. Passive update provides the system with better write performance when the workload is write dominant. However, the data availability is compromised in this case. Both passive and active updates are implemented in separate APIs in GlobLease and can be used by different applications with different requirements.

D. *Leasing Period*

The length of a lease is configurable in our system design. At the moment, the reconfiguration of the length of the lease is implemented with node granularity. Further, we plan to extend it to key granularity. The flexibility of lease length allows GlobLease to efficiently handle workload with different access patterns. Specifically, read dominant workload works better with longer leases (less overhead of lease renewals) and write dominant workloads cooperate better with shorter leases (less overhead of lease updates if the passive update mode is chosen).

Another essential issue of leasing is the synchronization of the leasing period on a master and its replication group. Every update from the master should correctly check the validity of all the leases on the non-masters according to the lease records and update them if necessary. This indicates that the record of the leasing period on the master should be the same with or last longer than the corresponding lease on the non-masters. Since it is extremely hard to synchronize the timings in a distributed system [6], we ensure that the record of the leasing periods on the master starts later than the leasing periods on the non-masters. The leases on the non-masters start when the messages of issuing the leases arrive. On the other hand, the records of the leases on the master start when the acknowledgement messages of the successful starting of the leases on the non-masters are received. With the assumption that the latency of message delivery in the network is much more significant than the clock drifts in each participating nodes. The above algorithm guarantees that the records of the leases on the master last longer than the leases on the non-masters and assures the correctness of the consistency guarantee.

E. *Master Migration and Failure Recovery*

Master migration is implemented based on a two-phase commit protocol. Master failure is handled by using the replication group as a Paxos group [10] to elect a new master. In order to keep the sequential consistency guarantee in our protocol, we need to ensure that either no master or only one correct master of a key exists in GlobLease.

The two phase commit master migration algorithm works as follows. In the prepare phase, the old master acts as the coordination node, which broadcasts new master proposal message in the replication group. The process will only move forward when an agreement is received from all the nodes from the replication group. In the commit phase, the old master broadcasts the commit message to all the nodes and changes its own state to recognize the new master. Notice that message loss or node failures may happen in this commit phase. If nodes in the replication group, which are not the new master, fail to commit to this message, the recognition of correct mastership is further fixed through an echo message gradually triggered by write requests. Specifically, if the mastership is not correctly changed, the write requests will be forwarded to the old master from the replication group. Since write messages should only be forwarded to the master, when the old master receives a write message from a node in its replication group, it assumes that this node does not know the correct master in the system. An echo message with the correct master information is sent to this node. And the write request is forwarded to the new master. If the new master fails to acknowledge in the commit phase, a roll-back operation will be issued from the old master to its replication group.

Master failure recovery is implemented based on the assumption of fail stop model [11]. There are periodical heartbeat messages from the non-master nodes in the replication group to check the liveness of the current master. If a master node cannot receive the majority of the heartbeat message within a timeout interval, it will give up its mastership to guarantee our previous assumption that there is no more than one master in the system. In the meantime, any non-master node can propose a master election process in the replication group if it cannot receive the response of the heartbeat messages from the master within sufficient continuous period. The master election process follows the two-phase Paxos algorithm. A non-master node in the replication group proposes its own ring ID as well as node ID as values. Only non-master nodes that have passed the heartbeat timeout interval may propose values and vote for the others. The node with the smallest ring ID gets more than the majority of the promises wins the election. Any non-master node that fails to recognize the new master will be guided through the write echo message described above.

F. Handling Read and Write Requests

With the lease consistency protocol, GlobLease is able to handle read and write requests with respect to the requirement of sequential consistency model. Read requests can be handled by the master of the key as well as the non-masters with valid leases. In contrast, write requests will eventually be routed to the responsible masters. The first time write and future updates of a key are handled differently by master nodes. Specifically, the first time, a write always uses the active update approach, because it creates a record of the written key on non-master nodes, which ensures the correct lookup of the data when clients contact the non-master nodes for read accesses. In contrast, future updates of a key can be handled either using the active or passive approach. After updating the data and lease on non-master nodes, lease records, which store the information of the leasing periods, the associated keys, and the associated non-master nodes, are maintained in master nodes. The lease records are referred when a write request is handled on the master node to decide whether the updates of the leases are required to the non-master nodes if the passive write mode is chosen. Algorithm 1 and Algorithm 2 present the pseudo codes for processing read and write requests.

Algorithm 1: Pseudo Code for Read Request

```
n.receiveReadRequest(msg)
if n.isResponsibleFor(msg.to) then
    if n.isMaster(msg.key) then
        value = n.getValue(msg.key);
        n.returnValue(msg.src, value);
    end
    if n.isExpired(lease) then
        n.forwardRequestToMaster(msg);
        n.renewLeaseRequest(msg.key);
    else
        value = n.getValue(msg.key);
        n.returnValue(msg.src, value);
    end
else
    nextHop = n.getNextHopOfReadRequest(msg.to);
    n.forwardRequestToNextNode(nextHop);
end
```

IV. Scalability and Elasticity

The architecture of GlobLease enables its scalability in two forms. First, the scalable structure of DHTs allows GlobLease to achieve elasticity by adding or removing nodes to the ring overlay. With this property, GlobLease can easily expand to a larger scale in order to handle generally larger workload or scale down to save cost. However, this form of scalability is associated with large overhead, including reconfiguration of multiple ring overlays, key range divisions and the data rebalancing associated, and the churn of the routing table stored in each node's memory. Furthermore, this approach is feasible only when the workload is growing in a uniform manner. Thus, when confronting intensively changing workloads or with skewed distribution, this form of elasticity might not be enough. We have extended the system with fine-grained elasticity by using affiliated nodes.

A. Affiliated Nodes

Affiliated nodes are used to leverage the elasticity of the system. Specifically, the application of affiliated nodes allows configurable replication degrees for each key. This is achieved by attaching affiliated nodes to any standard

Algorithm 2: Pseudo Code for Write Request

```
n.receiveWriteRequest(msg, MODE)
%Check whether it is a key update with passive update mode;
if n.contains(msg.key) & MODE == PASSIVE then
    leaseRec = n.getLeaseRecord(msg.key);
    if leaseRec == ALLEXPIRE then
        n.writeValue(msg.key, msg.value);
        lazyUpdate(replicationGroup, msg);
        return SUCCESS ;
    end
else
    lease = n.generatorLease();
    for $server \in replicationGroup$ do
        checkResult = n.issueLease(server, msg.key,
        msg.value, lease) ;
    end
    while retries do
        ACKServer = getACKs(checkResult) ;
        noACKServer = replicationGroup-ACKServer ;
        leaseExpired = getLeaseExp(leaseRec) ;
        if $noACKServer \in leaseExpired$ &
        $sizeOf(noACKServer) <$
        $sizeOf(replicationGroup)/2$ then
            lazyUpdate(noACKServer, msg);
            n.writeValue(msg.key, msg.value);
            for $server \in ACKServer$ do
                n.putLeaseRecord(server, msg.key, lease) ;
            end
            return SUCCESS;
        else
            for $server \in noACKServer$ do
                checkResult += n.issueLease(server,
                msg.key, msg.value, lease);
            end
            retries -= retries;
        end
    end
    return FAIL;
end
```

nodes, which are called host standard nodes in this case. Then, a configurable subset of the keys served in the host standard node can be replicated at attached affiliated nodes. The affiliated nodes attached to the same host standard node can have different configurations on the set of the replicated keys. The host standard node is responsible to issue and maintain leases of the keys replicated at each affiliated node. The routing links to the affiliated nodes are established in other standard nodes' routing tables respect to a specific key after the first access forwarded by the host standard node. If multiple affiliated nodes hold the same key, the host standard node forwards requests in a round-robin fashion.

Affiliated nodes are designed as lightweight processes that can join/leave system overlay by only interacting with a standard node. Thus, addition and removal of affiliate nodes introduce very little overhead. Rapid deployment of affiliated nodes allows GlobLease to swiftly handle workload spikes. Furthermore, the dynamic configuration of the replication degrees on a key basis allows skewed (popular) keys to be highly replicated on the affiliated nodes on demand. This key-based extra replication not only allows GlobLease to handle skewed workloads, but also further leverage the fast deployment of affiliated nodes, which requires less data movement overhead by precisely replicating the highly demanded keys. In this way, GlobLease is also able to handle spiky and skewed workload in a swift fashion. There is no theoretical limit on the number of the affiliated nodes in the system, the only concern is the overhead to maintain data consistency on them, which is explained in the next section.

Consistency Issues: In order to guarantee data consistency in affiliated nodes, a secondary lease is established between an affiliated node and a host standard node. The secondary lease works in a similar way as the lease protocol introduced in Section III. An affiliated node holding a valid lease of a specific key is able to serve the read requests of that key. The host standard node is regarded as the master to the affiliated node and maintains the secondary lease. The principle of issuing a secondary leases on an affiliated node is that it should be a sub-period of a valid lease of a specific key holding on the host standard node. The invalidation of a key's lease on a host standard node involves the invalidation of all the valid secondary leases of this key issued by this host standard node to its affiliated nodes.

V. Evaluation

We evaluate the performance of GlobLease under different intensities of read/write workloads in comparison with Cassandra [1]. Furtermore, the evaluation of GlobLease goes through its performance with different read/write ratios in workloads and different configurations of lease lengths. The fine-grained elasticity of GlobLease is also evaluated through handling spiky and skewed workloads.

A. Experiment Setup

We use Amazon Elastic Compute Cloud (EC2) to evaluate the performance of GlobLease. The choice of Amazon EC2 allows us to deploy GlobLease in a global scale. We evaluate GlobLease with four DHT rings deployed in the U.S. west (California), U.S. East, Ireland, and Japan. Each DHT ring consists of 15 standard nodes and a configurable number of affiliated nodes according to different experiments. We use the same Amazon EC2 instance to deploy standard nodes and affiliated nodes. One standard or affiliated node is deployed on one Amazon EC2 instance. The configuration of the nodes are described in Table I.

As a baseline experiment, Cassandra is deployed using the same EC2 instance type and amount in each region as GlobLease. We configure read and write quorums in Cassandra in favor of its performance. Specifically, for read dominant workload, Cassandra reads from one replica and writes to all replicas. For write dominant workload, Cassandra writes to one replica and reads from all replicas. Note that with this configuration, Cassandra gains more performance since only one replica from one region is needed to process a request. However, Cassandra only

Table I
NODE SETUPS

Specifications	Nodes in GlobLease	YCSB client
Instance Type	m1.medium	m1.xlarge
CPUs	Intel Xeon 2.0 GHz	Intel Xeon 2.0 GHz*4
Memory	3.75 GiB	15 GiB
OS	Ubuntu Server 12.04.2	Ubuntu Server 12.04.2
Location	U.S. West, U.S. East, Ireland, Japan	U.S. West, U.S. East, Ireland, Japan

Table II
WORKLOAD PARAMETERS

Total clients	50
Request per client	Maximum 500 (best effort)
Request rate	100 to 2500 requests per second (2 to 50 requests/sec/client)
Read dominant workload	95% reads and 5% writes
Write dominant workload	5% reads and 95% writes
Read skewed workload	Zipfian distribution with exponent factor set to 4
Length of the lease	60 seconds
Size of the namespace	10000 keys
Size of the value	10 KB

achieves casual consistency in read-dominant experiment and sequential consistency in write dominant experiment, which is less stringent than GlobLease. More replicas (overhead) are needed to achieve sequential consistency with read dominant workload in Cassandra. Even though, GlobLease outperforms Cassandra as shown in our evaluations.

We have modified Yahoo! Cloud Serving Benchmark (YCSB) [12] to generate either uniform random or skewed workloads to GlobLease and Cassandra. YCSB clients are deployed in an environment described in Table I and parameters for generating workloads are presented in Table II.

B. Varying Load

Fig. 2 presents read performance of GlobLease with comparison to Cassandra using the read dominant workload. The workloads are evenly distributed to all the locations according to GlobLease and Cassandra deployments. The two line plots describe the average latency of GlobLease and Cassandra under different intensities of workloads. In GlobLease, the average latency slightly decreases with the increase of workload intensity because of the efficient usage of lease. Specifically, each renewal of the lease involves the interaction between master and non-master nodes, which introduces high cross region communication latency. When the intensity of the read dominant workload increases, within a leasing period, data with valid leases are more frequently accessed, which results in a large portion of requests are served with low latency. This leads to the decrease of the average latency in GlobLease. In Contrast, as the workload increases, the contention for routing and the access to data on each node are increased, which causes the slight increase of average latency in Cassandra.

The boxplot in Fig. 2 shows the read latency distribution of GlobLease (left box) and Cassandra (right box). The outliers, which are high latency requests, are excluded from the boxplot. The high latency requests are discussed in Fig. 4 and Fig. 5. The boxes in the boxplots are increasing slowly since the load on each node is increasing. The performance of GlobLease is slightly better than Cassandra in terms of the latency of local operations (operations that do not require cross region communication) shown in the boxplots and the average latency shown in line plots. There are several techniques that contribute to the high performance of GlobLease, including one-hop routing (lookup), effective load balancing (key range/mastership assignment) and efficient key-value data structure stored in memory.

For the evaluation of write dominant workload, we enable master migration in GlobLease. We assume that a unique key is only written in one region and the master of the key is assigned to the corresponding region. This assumption obeys the fact that users do not frequently change their locations. With master migration, more requests can be processed locally if the leases on the requested keys are expired and passive write mode is chosen. For the moment, the master migration is not automated, it is achieved by calling the master migration API from a script by analyzing the incoming workload (offline).

An evaluation using write-dominant workload on GlobLease and Cassandra is presented in Fig. 3. GlobLease achieves better performance in local write latency and overall average latency than Cassandra. The results can be explained in the same way as the previous read experiment.

Fig. 4 shows the performance of GlobLease and Cassandra using two read dominant workload (85% and 95%) in CDF plot. The CDF gives a more complete view of two systems' performance including the cross region communications. Under 85% and 95% read dominant workload, Cassandra experience 15% and 5% cross region communications, which are more than 500 ms latency. These cross region communications are triggered by write operations because Cassandra is configured to read from one replica and write to all replicas, which in favor of its performance under the read dominant workload. In contrast, GlobLease pays around 5% to 15% overhead in maintaining leases (cross region communication) in 85% and 95% read dominant workloads as shown in the figure. From the CDF, around 1/3 of the cross region communication in GlobLease are around 100 ms, another 1/3 are around 200 ms and the rest are, like Cassandra, around 500 ms. This is because renewing/invalidating leases do not require all the replicas to participate. Respect to the consistency algorithm in GlobLease, only master and non-masters with valid lease of the requested key are involved. So master of a requested key in GlobLease might need to interact with 0 to 3 non-masters to process a write request. Latency connecting data centers varies from 50 ms to 250 ms, which result in 100 ms to 500 ms round trip. In GlobLease, write request are processed with global communication latency ranging from 0 ms to 500 ms depending on the number of non-master replicas with valid lease. On the other hand, Cassandra always needs to wait for the longest latency among servers

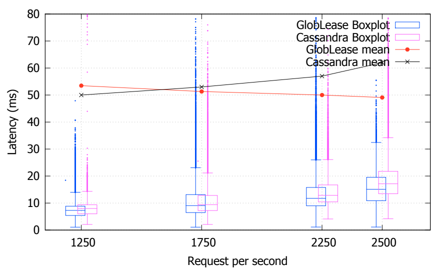

Figure 2. Impact of varying intensity of read dominant workload on the request latency

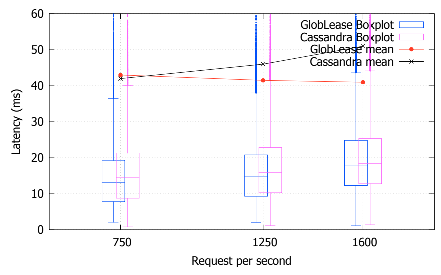

Figure 3. Impact of varying intensity of write dominant workload on the request latency

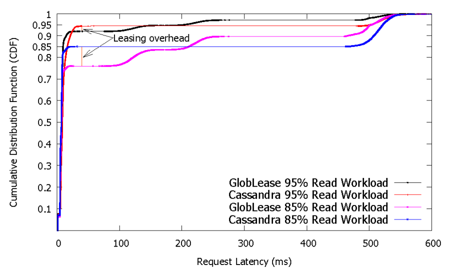

Figure 4. Latency distribution of GlobLease and Cassandra under two read dominant workloads

in different data centers to process a write operation which requires the whole quorum to agree. As a result, GlobLease outperforms Cassandra after 200 ms as shown in Fig. 4. Fig. 5 zooms in on the high latency requests (above 300 ms) in Fig. 4 under three read dominant workloads (75%, 85% and 95%). GlobLease significantly reduces (around 50%) high latency requests comparing to Cassandra. This improvement is crucial to the applications that are latency sensitive or having stringent SLO requirements.

C. Lease Maintenance Overhead

In Fig. 6, we evaluate lease maintenance overhead in GlobLease. The increasing portion of write request imposes more lease maintenance overhead on GlobLease since writes trigger lease invalidation and cause future lease renewals. The y-axis in Fig. 6 shows the extra lease maintenance messages comparing to Cassandra under throughput of 1000 request per second and 60 second lease. The overhead of lease maintenance is bounded by the following formula:

$$\frac{WriteThroughput}{ReadThroughput} + \frac{NumberOfKeys}{LeaseLength * ReadThroughput}$$

The first part of the formula represents the overheads introduced by writes that invalidate leases. The second part of the formula stands for the overheads for reads to renew leases. Even though lease maintenance introduces some overhead, GlobLease can outperform quorum-based storage systems, such as Cassandra, when latency between data centers vary. GlobLease benefits from smaller latency among close data centers as shown in Fig. 4 and Fig. 5.

D. Varying Read/Write Ratio

In Fig. 7, we vary the read/write ratio of the workload. The workload intensity is fixed to 1000 request per second for both GlobLease and Cassandra. As shown in Fig. 7, GlobLease has larger average latency comparing to Cassandra when the write ratio is low. This is because that GlobLease pays overhead to maintain leases as evaluated in Fig. 6. However, GlobLease outperforms Cassandra when the write ratio grows. This is explained in Fig. 5 where GlobLease reduces the percentage of high latency requests significantly comparing to Cassandra. The improvement on the high latency requests compensate the overhead of lease maintenance leading better average latency in GlobLease.

E. Varying Lease Length

We vary the length of leases to examine its impact on access latency for read-dominant and write-dominant workloads. The workload intensity is set to 1000 requests per sec. Fig. 8 shows that, with the increasing length of the lease, average read latency improves significantly since, in a valid leasing time, more read accesses can be completed locally. In contrast, average write latency increases since more cross-region updates are needed if there are valid leases in non-master nodes. Since the percentage of the mixture of reads and writes in read and write dominant workload are the same (95%), with the increasing length of the lease, they approximate the same steady value. Specifically, this steady value, which is around 60s in our case, is also influenced by the throughput of the system and the number of key entries.

F. Skewed Read Workload

In this experiment, we measure the performance of GlobLease under highly skewed read-dominant workload, which is common in the application domain of social networks, wikis, and news where most of the clients are readers and the popular contents attract most of the clients. We have extended YCSB to generate highly skewed read workload following the Zipfian distribution with the exponent factor of 4. Fig. 9 shows that, when GlobLease has sufficient number of affiliated nodes (6 in this case), it can handle skewed workload by coping the highly skewed keys in the affiliated nodes. The point in the top-left corner of the plot shows the performance of the system without affiliated nodes, which is the case of a system without fine-grain replica management. This scenario cannot expand to higher load because of the limit of high latency and the number of clients.

G. Elasticity with Spiky and Skewed Workload

Fig. 10 shows GlobLease's fine-grained elasticity under highly spiky and skewed workload, which follows a Zipfian distribution with the exponent factor of 4. The workload is spread evenly in three geographical locations, where GlobLease is deployed. The intensity of the workload changes from 400 req/s to 1800 req/s immediately at 50s point in the x-axis. Based on the key ranks of the Zipfian distribution, the most popular 10% of keys are arranged to be replicated in the affiliated nodes in three geographical locations. Based on our observation, it takes only tens

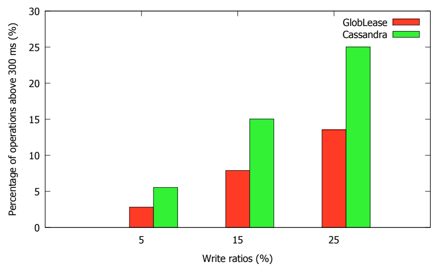

Figure 5. High latency requests

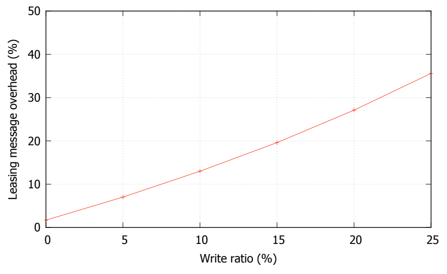

Figure 6. Impact of varying read:write ratio on the leasing overhead

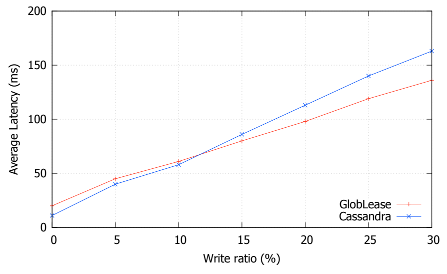

Figure 7. Impact of varying read:write ratio on the average latency

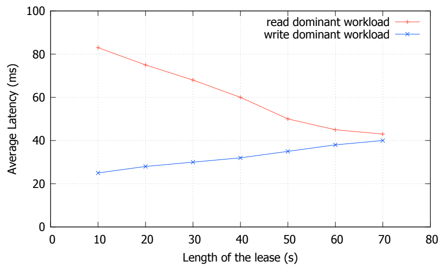

Figure 8. Impact of varying lengths of leases on the average request latency

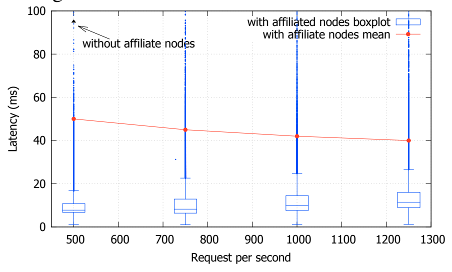

Figure 9. Impact of varying intensity of skewed workload on the request latency

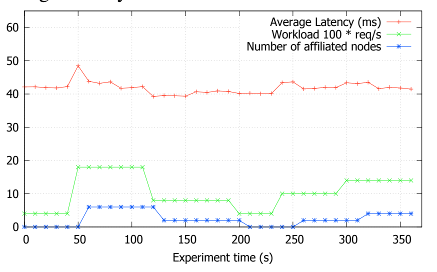

Figure 10. Elasticity experiment of GlobLease

of milliseconds for an affiliated node to join the overlay and several seconds to transfer the data to it. The system stabilizes with affiliated nodes serving the read workloads in less than 10 sec. Fig. 10 shows that GlobLease is able to handle highly spiky and skewed workload with stable request latency, using fine-grained replica management in the affiliated nodes. For now, the process of workload monitoring, key pattern recognition, and keys distribution in affiliated nodes are conducted with pre-programmed scripts. However, this can be automated using control theory and machine learning as discussed in [13], [14], [15].

VI. Related Work

A. Distributed Hash Tables

DHTs have been widely used in many storage systems because of their P2P paradigm, which enables reliable routing and replication in the presence of node failures. Selected studies of DHTs are presented in Chord [5], Pastry [16], Symphony [9]. The most common replication schema implemented on top of DHTs are successor-lists, multiple hash functions or leaf-sets. Besides, ID-replication [8], [17] and symmetric replication [18] are also discussed in literature. Our approach takes advantage of DHT's reliable routing and self-organizing structure and is different from the existing approaches in two aspects. First, we have implemented our own replication schema across multiple DHT overlays, which aims at fine-grained replica placement in the scenario of geographical replication. Our replication schema is similar to [8] but differs from it in the granularity of replica management and the routing across replication groups. Second, when GlobLease is deployed in a global scale, request routing is handled by selecting link with low latency according to the deployment.

B. Data Consistency

Consistency protocols in geo-replicated scenarios have gained great interests recently. Recent proposed solutions [7], [19], [20] use the classical Paxos algorithm [10], which requires a quorum to agree on operations and transactions. In contrast, we implement a strong consistency protocol inspired by the cache coherency protocol [21]. Our consistency schema distinguishes the read and write performance. We expect to have better performance in reads comparing to the previous approaches, since, most of the times, no global synchronization is needed. We have masters on key basis to handle the write accesses. With the flexibility to migrate the masters to the most intensive written location, the write accesses are also improved in GlobLease.

C. Lease-based Consistency

There are many usage scenarios of leases in distributed systems. Leases are first proposed to deal with distributed cache consistency issues in [22]. Later, the idea of using leases to maintain cache consistency is extended in [23]. Leases are also used to improve the performance of classic Paxos algorithm [24]. Furthermore, leases were explored to preserve consistency in transactions [25], [26], [27]. In sum, leases are used to guarantee the correctness of a resources in a time interval. Because leases are time-bounded assertions of resources, leases are fault tolerant in a distributed environment. In our paper, we explore the usage of leases in maintaining data consistency in a geo-replicated key-value store. Leases are used to reduce the overhead of consistency maintenance across geographical areas where communications among nodes observe significant latency. Evaluation shows that lease is effective in reducing high latency requests by only paying a reasonable overhead.

D. Elasticity Issues

Elasticity is a property of a system, which allows it to scale up and down, i.e., to grow and shrink, in order to offer satisfactory service with reduced cost in the presence of changing workloads. In particular, elasticity of a storage service, which requires data to be properly allocated before serving the clients, is well studied in [15], [14], [13], etc. However, to our best knowledge, most of these works tackle with elasticity in a coarse-grained fashion under the assumption that the changing of workloads are uniformly distributed on each participating node. In this way, the elasticity is achieved by adding/removing nodes based on workload intensity without transparently managing data skewness. In contrast, GlobLease focuses on fine-grained elasticity. Skewed or spiky keys are efficiently and precisely replicated with higher replication degree and start serving the workload with reduced overhead in affiliated nodes.

E. Storage Systems

Many successful distributed storage systems have been built by cutting-edge IT companies recently, including Google's Spanner [4], Facebook's Cassandra [1], Microsoft's Azure storage [28], Linkedin'a Voldemort [29], Yahoo!'s PNUTS [3], and Amazon's Dynamo [2]. GlobLease, as well as Voldemort, Cassandra and Dynamo, employs DHTs for namespace division, request routing, and fault tolerance. GlobLease differs from them mainly in two aspects. First, they will not scale to a global scale because of the successor-list replication, which, to some extent, tightly bounds system deployment. GlobLease solves this issue by replicating multiple DHTs with integration of geo-aware routing. Second, to our best knowledge, none of these systems is able to change the replication degree of a specific key swiftly on the fly to handle highly skewed and spiky workload.

VII. Conclusion

GlobLease aims at achieving low latency data accesses and fine-grain replica management in a global scale. It employs multiple DHT overlays, each of which is a replicated data store in order to reduce access latency in a global scale. Geo-aware routing among DHTs is implemented to reduce data lookup cost. A lease-based consistency protocol is designed and implemented, which is optimized for keeping sequential data consistency with reduced global communication. Comparing to Cassandra, GlobLease achieves faster read and write accesses in a global scale with less than 10 ms latency in most of the cases. The overhead of maintaining leases and the influence of lease length are also studied in our work. Furthermore, the secondary leasing protocol allows us to efficiently control data consistency in affiliated nodes, which are used to serve spiky and skewed read workloads. Our evaluation shows that GlobLease is able to react to an instant 4.5 times workload increase with skewed key distribution swiftly and brings down the request latency in less than 20 seconds. In sum, GlobLease has demonstrated its optimized performance and fine-grained elasticity under sequential consistency guarantee in a global scale.

Acknowledgment

This work was supported by the Erasmus Mundus Joint Doctorate in Distributed Computing funded by EACEA of the European Commission under FPA 2012-0030, and by the FP7 project CLOMMUNITY funded by the European Commission under EU FP7 GA number 317879, and by the End-to-End Clouds project funded by the Swedish Foundation for Strategic Research under the contract RIT10-0043.

References

[1] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 2010.

[2] Giuseppe DeCandia, Deniz Hastorun, and et al. Dynamo: Amazon's highly available key-value store. In *Proc. SOSP*, 2007.

[3] Brian F. Cooper, Raghu Ramakrishnan, and et al. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB*, 2008.

[4] James C. Corbett, Jeffrey Dean, and et al. Spanner: Google's globally-distributed database. In *Proc. OSDI*, 2012.

[5] Ion Stoica, Robert Morris, and et al. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. SIGCOMM*, 2001.

[6] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 1978.

[7] Tim Kraska, Gene Pang, and et al. Mdcc: multi-data center consistency. In *Proc. EuroSys*, 2013.

[8] Tallat M. Shafaat, Bilal Ahmad, and Seif Haridi. Id-replication for structured peer-to-peer systems. In *Proc. Euro-Par*, 2012.

[9] Gurmeet Singh Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony: distributed hashing in a small world. In *Proc. USITS*, 2003.

[10] Leslie Lamport. Paxos made simple. *ACM Sigact News*, 2001.

[11] Richard D. Schlichting and Fred B. Schneider. Fail-stop processors: an approach to designing fault-tolerant computing systems. *ACM Trans. Comput. Syst.*, 1983.

[12] Brian F Cooper, Adam Silberstein, and et al. Benchmarking cloud serving systems with ycsb. In *Proc. SOCC*, 2010.

[13] Ahmad Al-Shishtawy and Vladimir Vlassov. Elastman: elasticity manager for elastic key-value stores in the cloud. In *Proc. CAC*, 2013.

[14] Beth Trushkowsky, Peter Bodík, and et al. The scads director: scaling a distributed storage system under stringent performance requirements. In *Proc. FAST*, 2011.

[15] Harold C Lim, Shivnath Babu, and Jeffrey S Chase. Automated control for elastic storage. In *Proc. ICAC*, 2010.

[16] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proc. Middleware*, 2001.

[17] Lisa Glendenning, Ivan Beschastnikh, and et al. Scalable consistency in scatter. In *Proc. SOSP*, 2011.

[18] Ali Ghodsi, LucOnana Alima, and Seif Haridi. Symmetric replication for structured peer-to-peer systems. In *Databases, Information Systems, and Peer-to-Peer Computing*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007.

[19] Stacy Patterson, Aaron J. Elmore, and et al. Serializability, not serial: concurrency control and availability in multi-datacenter datastores. *Proc. VLDB*, 2012.

[20] Jim Gray and Leslie Lamport. Consensus on transaction commit. *ACM Trans. Database Syst.*, 2006.

[21] James Archibald and Jean-Loup Baer. Cache coherence protocols: evaluation using a multiprocessor simulation model. *ACM Trans. Comput. Syst.*, 1986.

[22] C. Gray and D. Cheriton. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. *SIGOPS Oper. Syst. Rev.*, 1989.

[23] Jian Yin, Lorenzo Alvisi, Michael Dahlin, and Calvin Lin. Volume leases for consistency in large-scale systems. *IEEE Trans. on Knowl. and Data Eng.*

[24] Felix Hupfeld, Björn Kolbeck, and et al. Fatlease: scalable fault-tolerant lease negotiation with paxos. *Cluster Computing*, 2009.

[25] Jed Liu, Tom Magrino, and et al. Warranties for faster strong consistency. In *Proc. NSDI*, 2014.

[26] Nuno Carvalho, Paolo Romano, and Luís Rodrigues. Asynchronous lease-based replication of software transactional memory. In *Proc. Middleware*, 2010.

[27] Danny Hendler, Alex Naiman, and et al. Exploiting locality in lease-based replicated transactional memory via task migration. In *Distributed Computing*, Lecture Notes in Computer Science. 2013.

[28] Brad Calder, Ju Wang, and et al. Windows azure storage: a highly available cloud storage service with strong consistency. In *Proc. SOSP*, 2011.

[29] Roshan Sumbaly, Jay Kreps, and et al. Serving large-scale batch computed data with project voldemort. In *Proc. FAST*, 2012.