DB-Risk: The Game of Global Database Placement

[Demonstration proposal]

Victor Zakhary Faisal Nawab Divyakant Agrawal Amr El Abbadi
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106
{victorzakhary,nawab,agrawal,amr}@cs.ucsb.com

ABSTRACT

Geo-replication is the process of maintaining copies of data at geographically dispersed datacenters for better availability and fault-tolerance. The distinguishing characteristic of geo-replication is the large wide-area latency between datacenters that varies widely depending on the location of the datacenters. Thus, choosing which datacenters to deploy a cloud application has a direct impact on the observable response time. We propose an optimization framework that automatically derives a geo-replication placement plan with the objective of minimizing latency. By running the optimization framework on real placement scenarios, we learn a set of placement optimizations for geo-replication. Some of these optimizations are surprising while others are in retrospect straight-forward. In this demonstration, we highlight the geo-replication placement optimizations through the DB-Risk game. DB-Risk invites players to create different placement scenarios while experimenting with the proposed optimizations. The placements created by the players are tested on real cloud deployments.

Keywords

Geo-replication, transactions, placement

1. INTRODUCTION

Cloud applications strive for a 24/7 service with fast response time. Geo-replication has become necessary to achieve these objectives; a 24/7 service might be disrupted when datacenter-scale outages occur. Geo-replication helps alleviate this disruption by allowing requests to be directed only to operating datacenters. Fast response time is a challenge when the cloud application's user base is dispersed across the world. Geo-replication brings data closer to the user and increases the level of read availability.

When geo-replicating a cloud application, the system administrator is faced with an important design decision: at which datacenters should the data be placed? Cloud providers offer more than a few datacenters for deployment (*e.g.*, Amazon AWS hosts applications in 10 datacenters around the world each with multiple availability zones). With multiple cloud providers, the possibilities for geo-replicated deployments are the subsets of tens of datacenters. The placement decision affects many performance characteristics.

In this work, we focus on the effect of geo-replication placement on the response time for transactional workloads with serializability as the correctness guarantee. The response time of transactions on geo-replicated data is directly affected by the *deployment topology*, which we define as the relative locations of datacenters and the communication latencies between them. The topology affects response time differently for different replication protocols. Our study focuses on majority protocols, but can be applied on centralized protocols, coordination-free techniques [2], and Helios [3]; a recent geo-replication proposal.

We propose an optimization framework to make geo-replication placement decisions. The optimization framework, *framework* for short, constructs a model of the system. The model includes the topology, workload, and user distribution. Then, the model is used in an optimization problem formulation with the objective of minimizing **transaction latency** in addition to adhering to fault-tolerance and quality-of-service guarantees.

The framework has been built to allow *breaking* the rules of the replication protocols. Thus, the framework could yield outcomes with "tweaks" that are not part of the original protocol. Sometimes these tweaks are not feasible. But other times, these tweaks are viable optimizations to the original protocol that will result in better transaction latency. Some of the viable optimizations were more common than others, and upon closer examination it becomes clear that they exemplify a set of best practices, or optimizations, for geo-replication placement. We present three of the optimizations that we have found to be rewarding in geo-replication. The first optimization is quite novel while the other two optimizations were introduced before. The three optimizations, altogether, achieve in some deployments the minimum transaction latency.

We propose the game DB-Risk[1] to demonstrate the challenges in geo-replication placement, and showcase our framework and geo-replication placement optimizations. The players are presented with a geo-replication environment with clients distributed globally. Players then compete by

[1]The game and name are inspired from the famous game "Risk: The Game of Global Domination".

SIGMOD'16, June 26-July 01, 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2882903.2899405

placing their replicas at datacenters with the objective of minimizing the transaction response time for clients. The game proceeds in multiple rounds with the opportunity to apply a subset of the geo-replication optimizations to lower the response time. The winner is the player who places replicas to achieve the lowest response time. DB-Risk is augmented with our framework to compare the winner's placement to the calculated optimal placement. We also incorporate a real deployment on Amazon AWS to validate the performance of the chosen placements.

2. BACKGROUND

System model. Our model of geo-replication consists of a topology of datacenters and clients executing transactional workload. Data is fully replicated to a subset of datacenters. Users issue transactions that consist of read and write operations. Each client execute transactions back-to-back. The execution of transactions depends on the replication protocol. We focus in this work on majority based protocols. Next is a description of the variation we use for the majority protocol.

Majority. We adopt a variant of the majority protocol. A client executes a transaction by performing the reads and buffering the writes. Read requests are sent to a majority of datacenters. The highest version read is used. After executing reads and writes a vote request is sent to datacenters. The vote request consists of the read versions and the buffered write operations. Each datacenter, upon receiving a vote request, attempts to lock objects that are being written. Additionally, it verifies that the read versions were not overwritten. If both are successful, the datacenter sends back a positive vote. Otherwise, a negative vote is sent. The client commits the transaction if a majority of positive votes is received and aborts the transaction otherwise. The client sends the decision *to all replicas*. Once a majority acknowledges the receipt of the decision, the transaction terminates. The transaction latency is the time from the beginning of executing operations until terminating the transaction. We define the commit latency as the time spent committing the transaction, which is equivalent to the transaction latency without the time spent reading the data values.

Cost of coordination. A central concern of this work is optimizing transaction latency by finding the best replica placement. In an earlier work, we have found that the sum of transaction latencies at any two datacenters cannot be smaller than the RTT between these datacenters [3].

Related work. Geo-replication and data placement are areas that have undergone extensive research. Recent works study the problem of data placement for geo-replication [1, 4, 5, 8, 9]. Spanstore [8] is the closest work to our optimization framework. It tackles the data placement problem in geo-replication using an optimization formulation. Ping et. al. [4, 5] propose the use of a utility function to derive a placement that balances between speed and availability. Volley [1] analyzes usage logs and leverages an optimization formulation to place data partitions. Unlike our optimization framework, these works do not support a multi-access transactional workload with strong consistency. Sharov [6] optimizes for transactional workloads using leader based protocols. The optimization focuses on the leader placement, the leader and replica roles, and replica location, roles, and the leader. A transactional workload requires more complex coordination between replicas to detect conflicts. Our framework is designed to engage in the design decisions the optimization techniques to be used in the majority based replication protocols, and illuminates unintuitive optimizations to achieve a better performance.

3. GEO-REPLICATION PLACEMENT

Where copies of data are placed is a critical design decision that affects, among others, the performance of the application. It turns out that it is not straight forward to devise a placement strategy on geo-replicated data. The optimal placement depends on the workload patterns, user base, in addition to monetary and even geo-political considerations. However, even in the simple case of a uniform workload with no restrictions, monetary or otherwise, the placement problem is still complex. The reason of this complexity stems from two characteristics: (1) we consider a transactional workload, which require replicas to coordinate to detect conflicts. (2) The topology of replicas in geo-replication is more complex than traditional replication; each link in the topology could have a significantly different communication latency. These two characteristics decides which replication protocol performs better and should be chosen. Even if we have a finite set of replication protocols to consider (we focus here on majority protocols), each protocol comes with many variations. The large design space makes it difficult to know the combination of a placement strategy and replication protocol variations that will lead to the best performance.

Optimization Framework

To address the large space of placements and replication protocol variations, we developed an optimization framework. The framework takes as input the replicas network topology, workload parameters, and availability constraints. It searches through the space of placements and protocol variations and chooses the ones that minimizes transaction's response latency. The straight-forward way to search through the possible space of replication protocol variations is to hard-code as many of them as possible. However, hard-coding restricts the space of possible variations. What we do instead is only hard-code a single variation of each family of protocols. Then, we intentionally allow the optimization framework to break some of the protocol's behavior. It is possible that the resulting "tweaks" are not feasible, leading to violation of correctness. But it is also possible that we arrive to the right tweaks that will optimize the performance of the replication protocol. In our study, we have repeatedly seen a set of optimizations reoccur. Some of them are straight-forward and some of them are surprising. Next, we summarize the derived placement optimizations for geo-replicated systems.

Placement Optimizations

Here we summarize the geo-replication placement optimizations that we learned from the optimization framework. For the rest of this discussion we assume that data is fully replicated.

Commit hand-off. In a fully-replicated system, a client accessing data typically sends the commit request to the local, or closest, replica. Our optimization framework shows that this is not always the best practice in a geo-replicated environment; it is sometimes better to send the commit request to a datacenter other than the local one. For example, consider Figure 1 that shows an example of geo-

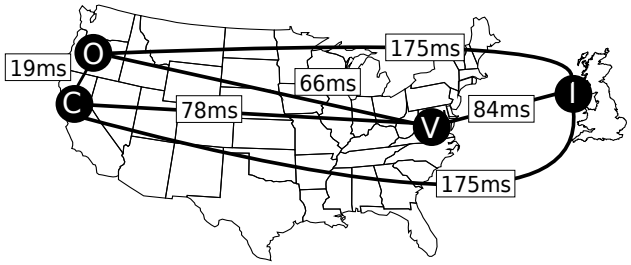

Figure 1: An example of replicating to four datacenters in Oregon (O), Virginia (V), California (C), and Ireland (I)

replication: four datacenters in Oregon (O), Virginia (V), California (C), and Ireland (I). The communication Round-Trip Times (RTTs) between the datacenters are shown in the figure. The RTTs between Ireland and the other datacenters are significantly higher than the remaining RTTs. Assume that a majority protocol is used to commit transactions in this scenario. The commit latency of a majority protocol is two RTTs to the closest majority. Thus, the commit latency of clients in Oregon is $132ms$, in California and Virginia is $156ms$, and in Ireland is $350ms$.

The large latency of clients in Ireland is due to the common convention that driving the transaction commitment from the local datacenter is the best practice. However, running this scenario in our optimization framework shows that this convention is not always true. In fact, clients in Ireland are in a better position sending their commit requests to Virginia. Committing a transaction in Virginia takes $156ms$. And sending the request from Ireland to Virginia and waiting for the decision to be sent back takes $84ms$. This means that the commit latency becomes the sum of the two latencies, which is equal to $240ms$. Compared to the original commit latency, this is a 31% improvement in commit latency of clients in Ireland. An insight from the optimization framework led to an improvement in commit latency with a simple change to the original protocol.

Passive replicas. Our second optimization is a byproduct of the commit hand-off optimization. In the scenario in Figure 1, when applying the hand-off optimization, clients at Ireland send their commit requests to Virginia. Clients in other datacenters send commit requests to their local replicas. This means that the replica at Ireland does not receive commit requests. With this knowledge, it is easy to observe that the demand placed on Ireland is lower than the other datacenters. For most of the time, Ireland will only serve read requests and receive the outcome of transactions and not drive the commitment of transactions. Thus, less resources need to be provisioned in Ireland and more resources need to be provisioned elsewhere.

When the framework recommends the hand-off optimization, it is sometimes accompanied by an interesting side effect. The side effect is that the replica that is not receiving commit requests becomes a *passive replica*. A passive replica is a replica that serves read requests but does not engage in the commit protocol. This means that it does not become part of the majority protocol, but more like a cache of data used for reading. In the example in Figure 1, this means that the majority protocol will involve getting votes only from Oregon, Virginia, and California. The number of replicas to constitute a majority has thus been lowered from three replicas when four datacenters were involved to two replicas now that only three replicas are involved. This makes the commit latency lower, since each replica needs to get votes from only one other replica. This makes the commit latency of Oregon and California $38ms$, of Virginia $132ms$, and of Ireland $216ms$. Making Ireland a passive replica reduced the average latency by 39%. It is important to note that this optimization is allowed in this scenario only if the system is required to tolerate only a single datacenter failure.

Optimistic reading. A conventional majority protocol reads from a majority of replicas to ensure that the most recent version is read. However, it is possible to optimistically read from the local replica only, and then validate the read in a majority of replicas in the commit phase [7]. The choice of whether to read optimistically or from a majority is controlled by a trade-off between the latency of read operations and contention. An optimistic read lowers the read latency but it increases contention because the version at the local replica might be stale. Reading from a majority requires a larger latency but ensures getting the most recent version. Running our optimization framework on various geo-replication scenarios reveals that optimistic reads are more favorable. It turns out that due to the large communication latencies between datacenters, a read from a majority is susceptible to a close contention level to the level when reading optimistically. Thus, there was no significant benefit in reading from a majority compared to reading optimistically, while an optimistic read enjoyed the benefit of a low latency.

4. THE DB-RISK GAME

We propose DB-Risk, a game to motivate thinking about the challenges of data placement in geo-replicated environments. It also showcases the capabilities of our optimization framework and the derived geo-replication placement optimizations. The game is designed to be played by two players to introduce a competitive element. The game assumes a majority protocol is used. An actual deployment of the majority protocol and placement optimizations are deployed on Amazon AWS to validate the players choices.

Summary. In DB-Risk, players place replicas in datacenters around the world with the aim of minimizing the transaction latency. The players also choose from the set of placement optimizations to enhance the transaction latency. The winner is the player with a placement and set of optimizations that achieve the lowest average transaction latency.

Rules. The game begins by showing a map similar to the map shown in Figure 2. The map shows a datacenter icon for each location that a player can choose. These datacenter locations correspond to actual datacenter locations of Amazon AWS. Players take turns choosing datacenters. Each datacenter can be chosen only once; a datacenter cannot host both players. The game proceeds in three rounds. The first round starts after placing data on datacenters. The average latency is calculated and displayed for each player. Before the second round, the players are given the opportunity to choose one *advantage card*. There are four advantage cards: three correspond to the placement optimizations and one card allows coordination-free execution of transactions [2]. The coordination-free card affects a randomly assigned per-

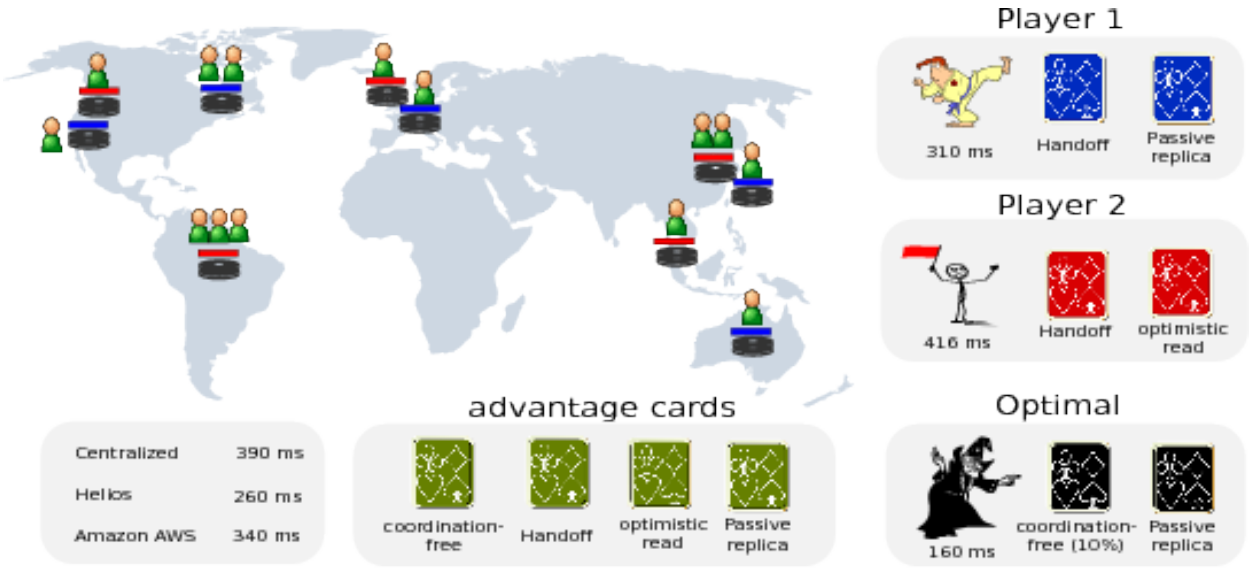

Figure 2: Two players place their application in datacenters and choose optimizations to win with the lowest average transaction latency

centage of the transactions. Then, the new average transaction latency is shown for both players. The third round is identical to the second round. The players choose another placement optimization and the final average transaction latencies are shown. The winner of the game is the player with the lower average transaction latency.

Comparisons. After announcing the winner, the optimization framework shows the optimal placement and the set of placement optimizations used to minimize transaction latency. In addition, the optimization framework also calculates the optimal placement for the centralized protocol and Helios (lower-bound latency) [3], and shows the transaction latency attained by them.

Validation. During the game, the transaction latencies are calculated using the system model in the optimization framework. But, for each game, the winning deployment is validated on a real deployment of the majority protocol and placement optimizations on Amazon AWS.

5. CONCLUSION

In geo-replication, the location of replicas plays a significant role in performance. We have developed an optimization framework that derives the optimal placement of replicas. Unlike prior work, our optimization framework models a transactional workload. Additionally, our framework is designed to engage in the design decisions of the replication protocol. Geo-replication placement optimizations are derived using the optimization framework. Most surprising is the hand-off optimization, which shows that sometimes it is rewarding to send commit requests to a farther datacenter rather than the local one. We proposed DB-Risk, a game to motivate the placement problem and to showcase our optimization framework and placement optimizations.

6. ACKNOWLEDGMENTS

This work is partially supported by NSF Grants 1018637, 1528178, and 1442966.

7. REFERENCES

[1] S. Agarwal et al. Volley: Automated data placement for geo-distributed cloud services. In *NSDI*, 2010.

[2] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Coordination avoidance in database systems. In *VLDB*, 2014.

[3] F. Nawab, V. Arora, D. Agrawal, and A. El Abbadi. Minimizing commit latency of transactions in geo-replicated data stores. In *SIGMOD*, 2015.

[4] F. Ping, J.-H. Hwang, X. Li, C. McConnell, and R. Vabbalareddy. Wide area placement of data replicas for fast and highly available data access. In *the International Workshop on Data-intensive Distributed Computing (DIDC)*, 2011.

[5] F. Ping, X. Li, C. McConnell, R. Vabbalareddy, and J.-H. Hwang. Towards optimal data replication across data centers. In *the Distributed Computing Systems Workshops (ICDCSW)*, 2011.

[6] A. Sharov, A. Shraer, A. Merchant, and M. Stokely. Take me to your leader!: online optimization of distributed storage configurations. *Proceedings of the VLDB Endowment*, 8(12):1490–1501, 2015.

[7] R. H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM TODS*, 1979.

[8] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha. Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services. In *SOSP*, 2013.

[9] Z. Ye, S. Li, and X. Zhou. Gcplace: Geo-cloud based correlation aware data replica placement. In *Symposium on Applied Computing*, 2013.