server has returned the service response. A two-phase commit can be implemented from the client to the server to allow the client to roll back the server transaction in cases where the client constraint does not pass. Unfortunately, this method doubles the message count for every transaction and reduces the improvements in availability we have already achieved.

Using the method from our previous work mapping course grained services to fine grained services [18] we are able to auto generate compensators. The use of the compensator allows a single round trip message from the client to the server when a constraint fails on the client after a response has been returned by the server. When a client side constraint fails the compensator is invoked to "undo" the state change that was performed by the service on the server. Figure 4 shows an activity diagram with post conditions on both the server and the client.

Figure 4 is a simple example where we care about client state relative to one web service request. In many service oriented designs, a session identifier is added to each web service call so the services can utilize server side state without forcing all previous results to be passed in the service call. The session identifier allows web services to be chained into transactional workflows. The transactional workflow can have split operations to allow simultaneous web service calls and join operations to wait for a set of web services to all complete before moving to the next stage in the workflow. In this type of transactional workflow, the correctness of the client side state is based on the results of several asynchronous services at a join operation.

Figure 6 is an expanded example activity diagram. In this workflow when a user adds value to their card they qualify for loyalty points. We model the loyalty point system as a cross-domain call in a separate realm of authority from the normal web-services used to record value transactions. On adding value, there is a split operation where two asynchronous web-service calls are made. One call is to the in-domain web-services to record the value added to the card. The second call is to an out of domain web-service to record loyalty point activity. There are several points of exception that can occur once the web-services calls are joined back together:

- Card is never touched – In this case, the web service calls may have taken too long, and the transaction needs to be undone through the use of the compensating web services
- Loyalty points on card and loyalty points returned from service are not in sync – In this case there may be a security issue where the loyalty points have been altered by a malicious user or program.
- Card number returned from either service does not match card number touched before and after – In this case the response may be coming from a previous call that took longer to process than expected.

For each of the exception cases with temporal constraints a violation of the invariant in the client requires the compensator on the server to be executed. The compensator is behind the buddy system and the operations are unknown to the client. To facilitate this process we enhanced the buddy system to return a URI for a web service that will execute the undo operation. The client can then call the web service in the case of an invariant failure.

6.2 Cross Domain Constraints

As the internet continues to connect business and consumers, transactions begin to involve external entities in their transaction correctness. For example, many businesses have begun to reward social micro-blogging with loyalty currency. The correctness of these transactions depends on data available via web-services available at Twitter™ and Facebook™. To investigate the type of constraints required with these cross-domain loyalty transactions we investigated two aggressive programs.

The Marriott™ Hotel chain rewards patrons for micro-blogging on Twitter using a specific hashtag of #MRpoints. The reward for the micro-blog post or micro-blog re-post is twenty-five points [9]. The currency value of each post is valued at around $0.42. They cap the daily value of all micro-blog posts at one hundred points. There are no constraints placed on the tweets semantics, timing or the validity of the twitter account.

To evaluate the semantics of the micro-blogs we pulled a sample consisting of a day worth of micro-blogs. The sample consisted of 3,982 unique micro-blogs including the tag and 1,474 re-posts of these blogs. Only four percent of the blogs included reference to a specific hotel.

To evaluate the timing of the micro-blogs we used the same sample from above and compared the first and last post for the day for the user. Eighty-two percent of the users in the sample who posted multiple micro-blogs did so in a timespan of less than five minutes.

To evaluate the validity of the micro-blog user accounts we used the same sample from above. For each poster account, we examined the activity on their twitter account. We defined a valid account as one where the Marriott tweets represented less than twenty percent of the tweets. Seventy-four percent of the user accounts in the sample were classified as not valid in our study.

JetBlue™ Airlines also rewards its patrons with loyalty points for micro-blogging on Twitter [10]. The airlines program is a little different in a few perspectives; they automate the instantiation of the micro-blog so they can insure

the post includes links to the program, they reward badges as a middle layer to earning loyalty currency, and they utilize micro-blogging on Facebook as well as Twitter. As in our case with Twitter we considered three types of validation requires; the blogs semantics, timing of the blog and the validity of the bloggers account.

To evaluate the timing of the micro-blogs we pulled a sample consisting of a week's worth of micro-blogs from twitter that used the tag #TrueBlueBadges. The sample only consisted of twenty-eight posts and zero re-posts (JetBlue did not reward badges for retweets). To understand why participation was so low we schedule a job that pulled the tweets every minute for a weeks' time. To our amazement sixty-three percent of the posts were deleted shortly after the post. Unlike the Marriott program that rewarded the points at a later point in the day, the JetBlue program would reward the credit immediately on the posting. Our new sample data consisted of ninety-seven original posts. Sixty-one were deleted, and twenty-one were part of batches of posts done in less than five minute time periods. We concluded that only fifteen percent were valid based on the timing activities.

To evaluate the semantics of the micro-blogs we used the same sample from above and found that no posts included additional data from the link template created by the JetBlue website.

To evaluate the validity of the micro-blog user accounts we used the same sample from above. For each poster account, we examined the activity on their twitter account. We defined a valid account as one where the Marriott tweets represented less than twenty percent of the tweets. Twenty-eight percent of the users were classified as not valid in our study. We attribute the higher percentage of validity to the ability of a user to cleanse their twitter account after an immediate post.

7. Empirical Results

We modeled a small urban transportation system with 100,000 users averaging 2 trips a day for 50 weeks a year. Each user is assumed to replenish his or her value once a week. We loaded the model into a Microsoft SQL Server 2008 server. A function was developed that takes a single argument, the card id, and returns the maximum sequence for that card id. SQL Server does not support sub-queries in check constraints but does support functions. The function was called from inside the constraint to enforce that new tuples have a sequence greater the current maximum for that card.

We tested insert timings of loads of concurrent transactions in blocks of 100 with the constraint implemented in the SQL Server with lazy replication and the *Buddy System* implementing the constraint with four clusters (Figure 7). Without the *Buddy System*, the SQL Server implementation performed well as long as there was an index on the card id. The index allowed the system to seek on the index tree to the subset of records for one customer. The database system did not use synchronization when performing the check constraint. The asynchronous check means that current consistency with lazy replication and the SQL implementation is not guaranteed. With the Buddy System, higher availability was achieved by distributing the inserts to all four clusters while guaranteeing the consistency.

We added cross-domain constraints that would award a patron extra value if they had tweeted about their experience on the subway while on the trip. The tweet was validated to be between the start and end of the trip and required a special hashtag.

All of the cross-domain constraints can be distributed and do not require checking at the buddy dispatcher. Therefore the cross-domain calls to twitter and Facebook could be made from the RDBMS system behind the cluster. Out of the box no SQL database allows for social micro-blogging constraints. We developed these constraints using SQL Servers .NET assembly technology which allows us to embedded .NET that calls out to twitter and Facebook to validate the constraints.

8. Related Work

Constraint specification and enforcement have been a part of relational database model research since Codd [6] originally wrote the specification. Recently work on auto generation of SQL code to enforce these constraints from the UML model has been done by Heidenreich, et al. [7] and Demuth, et al. [8]. In both these works, the focus is on the generation of the SQL code for relational databases for the invariants. Distributed databases and web services require additional work to ensure the constraints can be guaranteed while increasing availability of the service and data through the distribution of the load among the clusters available. Services also have constraints that need to be true before a service is called and after a service is completed.

Research has been conducted for decades on strict and lazy replication in RDMS. Recent research can be grouped into one of three goals: 1.) to increase the availability with strict replication, 2.) to increase consistency with lazy replication, and 3.) to use a hybrid approach to increase the availability. Our previous work, *The Buddy System*, increases availability will provide consistency and durability.

- *Increasing Availability with Strict Replication:* Several methods have been developed to ensure mutual consistency in replicated databases. The aim of these methods is eventually to provide one-copy serializability (1SR). Transactions on traditional replicated databases are based on reading any copy and writing (updating) all copies of data items. Based on the time of the update propagation, two main approaches have been proposed. Approaches that update all replicas before the

transaction can commit are called eager update propagation protocols; approaches that allow the propagation of the update after the transaction is committed are called lazy update propagation. While eager update propagation guarantees mutual consistency among the replicas, this approach is not scalable. Lazy update propagation is efficient, but it may result in violation of mutual consistency. During the last decade, several methods have been proposed to ensure mutual consistency in the presence of lazy update propagation (see [6] for an overview.) More recently, Snapshot Isolation (SI) [7, 8] has been proposed to provide concurrency control in replicated databases. The aim of this approach is to provide global one-copy serializability using SI at each replica. The advantage is that SI provides scalability and is supported by most database management systems.

- *Increasing Consistency in Lazy Replication:* Breitbart and Korth [9] and Daudjee, et al. [10] propose frameworks for master-slave, lazy-replication updates that provide consistency guarantees. These approaches are based on requiring all writes to be performed on the master replica. Updates are propagated to the other sites after the updating transaction is committed. Their framework provides a distributed serializable schedule where the ordering of updates is not guaranteed.

The approach proposed by Daudjee et al. provides multi-version serializability where different versions of data can be returned for read requests during the period that replication has not completed.

- *Hybrid Approach:* Jajodia and Mutchler [11] and Long, et al. [12] both define forms of hybrid replication that reduce the requirement that all replicas participate in eager update propagation. The proposed methods aim to increase availability in the presence of network isolations or hardware failures. Both approaches have limited scalability because they require a majority of replicas to participate in eager update propagation. Most recently, Irun-Briz, et al. [13] proposed a hybrid replication protocol that can be configured to behave as eager or lazy update propagation protocol. The authors provide empirical data and show that their protocol provides scalability and reduces communication cost over other hybrid
- update protocols. In addition to academic research, several database management systems have been developed that support some form of replicated data management. For example, Lakshman and Malik [14] describe a hybrid system, called Cassandra, which was built by Facebook to handle their inbox search. Cassandra allows a configuration parameter that controls the number of nodes that must be updated synchronously. The Cassandra system can be configured so nodes chosen for synchronous inclusion cross data center boundaries to increase durability and availability.
- *Buddy System:* In our previous work [3, 4, 15], we provide an architecture and algorithms that address three problems: the risk of losing committed transactional data in case of a site failure, contention caused by a high volume of concurrent transactions consuming limited items, and contention caused by a high volume of read requests. We called this system the Buddy System because it used pairs of clusters to synchronously update all transactions. The pairs of buddies can change for each request allowing increased availability by fully utilizing all server resources available. Consistency is increased over lazy-replication because all transactional elements are updated in the same cluster allowing for transaction time referential integrity and atomicity.

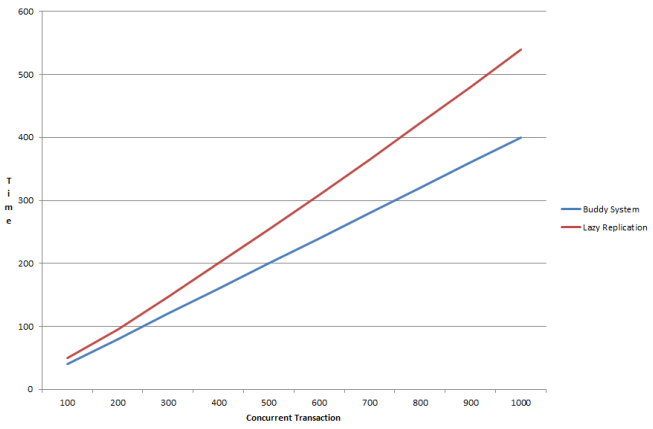

Figure 7. Empirical Results

An intelligent dispatcher was placed, in front of all clusters, to support the above components. The dispatcher operated at the OSI Network level 7. The high OSI level allowed the dispatcher to use application specific data for transaction distribution and buddy selection. The dispatcher receives the requests from clients and distributes them to the WS clusters. Each WS cluster contains a load balancer, a single database, and replicated services. The load balancer receives the service requests from the dispatcher and distributes them among the service-replicas. Within a WS cluster, each service shares the same database. Database updates among the clusters are propagated using lazy-replication propagation.

After receiving a transaction, the dispatcher picks the two clusters to form the buddy pair. The dispatcher selects the pair of clusters based on versioning history. If a version is in progress and the request is modifying the data, then the dispatcher chooses set containing the same pair currently executing the other modify transactions. Otherwise, the set contains any pair with the last completed version. The primary buddy receives the transaction along with its buddy's IP address. The primary buddy becomes the coordinator in a simplified commit protocol between the two buddies. Both buddies perform the transaction and commit or abort together.

The dispatcher maintains metadata about the freshness of data items in the different clusters. The dispatcher increments a version counter for each data item after it has been modified. Any two service providers (clusters) with the latest version of

the requested data items can be selected as a buddy. Note, that the database maintained by the two clusters must agree on the requested data item versions but may be different for the other data items.

9. Conclusion

In this paper, we propose an extension to the buddy system to handle integrity constraint guarantees. Our solution is based on extracting OCL design constraints from the UML models of the system. The dispatcher can then enforce these constraints using materialized aggregates. Each constraint's aggregate value is updating incrementally as new tuples are inserted into the database. The dispatcher is then able to distribute the requests to any cluster after the request passes the constraint check. We also defined new constraint types required for web service transactions (temporal and cross-domain). A limitation of our work is that we currently only support a subset of possible OCL notation for the expression of the aggregate constraints.

10. References

[1] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *SIGACT News,* vol. 33, pp. 51-59, 2002.

[2] D. Abadi, "Consistency tradeoffs in modern distributed database system design: Cap is only part of the story," *Computer,* vol. 45, pp. 37-42, 2012.

[3] A. Olmsted and C. Farkas, "High Volume Web Service Resource Consumption," in *Internet Technology and Secured Transactions, 2012. ICITST 2012*, London, UK, 2012.

[4] A. Olmsted and C. Farkas, "The cost of increased transactional correctness and durability in distributed databases," in *13th International Conference on Information Reuse and*, Los Vegas, NV, 2012.

[5] M. Aron, D. Sanders, P. Druschel and W. Zwaenepoel, "Scalable content-aware request distribution in cluster-based networks servers," in *Proceedings of the annual conference on USENIX Annual Technical Conference, ser. ATEC '00*, Berkeley, CA, USA, 2000.

[6] E. F. Codd, The Relational Model for Database Management, Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 1990.

[7] P. Gulutzan and T. Pelzer, SQL-99 Complete, Really, Lawrence, Kansas: R & D Books, 1999.

[8] A. Olmsted and C. Farkas, "Coarse-Grained Web Service Availability, Consistency and Durability," in *IEEE International Conference on Web Services*, San Jose, CA, 2013.

[9] Marriott International, "Marriott Rewards Points Plus," [Online].https://www.marriottrewardspluspoints.com/. [Accessed 30 07 2014].

[10] Jetblue Airways Corporation, "Jetblue Bonuses and Badges," [Online]. Available: https://trueblue.jetblue.com/web/trueblue/how-it-works-bonuses-and-badges. [Accessed 31 07 2014].

[11] F. Heidenreich, C. Wende and B. Demuth, "A Framework for Generating Query Language Code," *Electronic Communications of the EASST*, 2007.

[12] B. Demuth, H. Hußmann and S. Loecher, "OCL as a Specification Language for Business Rules in Database Applications," in *The Unified Modeling Language. Modeling Languages, Concepts, and Tools.*, Springer, 2001, pp. 104-117.

[13] M. T. Ozsu and P. Valduriez, Principles of Distributed Database Systems, 3rd ed., Springer, 2011.

[14] H. Jung, H. Han, A. Fekete and U. Rhm, "Serializable snapshot isolation," *PVLDB*, pp. 783-794, 2011.

[15] Y. Lin, B. Kemme, M. Patino Martıınez and R. Jimenez-Peris, "Middleware based data replication providing snapshot isolation," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data, ser. SIGMOD '05*, New York, NY, 2005.

[16] Y. Breitbart and H. F. Korth, "Replication and consistency: being lazy helps sometimes," *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, ser. PODS '97,* pp. 173-184, 1997.

[17] K. Daudjee and K. Salem, "Lazy database replication with ordering," in *Data Engineering, International Conference on*, 2004.

[18] S. Jajodia and D. Mutchler, "A hybrid replica control algorithm combin-ing static and dynamic voting," *IEEE Transactions on Knowledge and Data Engineering,* vol. 1, pp. 459-469, 1989.

[19] D. Long, J. Carroll and K. Stewart, "Estimating the reliability of regeneration-based replica control protocols," *IEEE Transactions on,* vol. 38, pp. 1691-1702, 1989.

[20] L. Irun-Briz, F. Castro-Company, A. Garcia-Nevia, A. Calero-Monteagudo and F. D. Munoz-Escoi, "Lazy recovery in a hybrid database replication protocol," in *In Proc. of XII Jornadas de Concur-rencia y Sistemas Distribuidos*, 2005.

[21] A. Lakshman and P. Malik, "Cassandra: a decentralized structured," *SIGOPS Oper. Syst. Rev.,* vol. 44, pp. 35-40, 2010.

[22] P. Ziemann and M. Gogolla, "Ocl extended with temporal logic," *Perspectives of System Informatics*, 2003.