6.UAP Final Report: Replication in H-Store

Kathryn Siegel

May 14, 2015

This paper describes my research efforts implementing replication in H-Store. I first provide general background on the H-Store project, an ongoing collaborative research project between several universities, including MIT. I then discuss the rationale behind adding replication to the system, followed by the implementation details of the project. Finally, I detail the performance and robustness results of my research, which focus on synchronous, semi-synchronous, and asynchronous replication schemes, as well as future work to improve my implementation.

1 Introduction

H-Store is an online transaction processing system that stores data in the main memory of the machines on which it runs. Its architecture consists of several hierarchical layers that handle data distribution and transaction execution over multiple machines without sharing data across machines. Previous study has shown that H-Store performs well in comparison to traditional relational database management systems. [1] However, H-Store lacks the ability for data to be replicated over multiple machines, a feature found in most RDBMSs; while H-Store once supported replication, this feature was disabled in a previous update and was never re-implemented within the current system architecture. Replication of database partitions over multiple sites is often crucial to maintain availability of data in the face of network or machine failures. My 6.UAP project thus focused on the re-implementation of replication of H-Store for single-partition transactions across different sites.

1.1 The H-Store System

At a high level, the H-Store system is structured as follows. Overlaying the system on each machine is an H-Store coordinator, which handles communication between sites in

the form of RPCs. The H-Store coordinator passes calls down to specific partition executors, which then queues the transaction for execution. The transaction is then picked up from the execution queue and executed on the local partition.

H-Store handles single-partition and distributed transactions. For the purposes of simplicity, my project focused on single partition transactions; these transactions are sent by the H-Store coordinator to one partition, where there is a single transaction execution call that can be intercepted to replicate the transaction on another partition. The nature of distributed transactions complicates this method of forwarding transactions, and was generally out of the scope of this project, with the exception of transactions belonging to system procedures. Because these system procedures are necessary for basic functions such as initially loading data into H-Store, my implementation of replication handles correctly processing these procedures on both primary partitions and their replicas.

1.2 Replication

I evaluated the effects of three different types of replication on the latency and throughput of transaction execution in H-Store. These three replication schemes are as follows:

1) **Synchronous replication:** Execution of a transaction at a primary partition halts until the primary partition receives a notification that the transaction has finished executing at the secondary partition(s).

2) **Semi-synchronous replication:** Execution of a transaction at a primary partition halts until the primary partition receives a notification that the transaction has been received at the secondary partition(s). The secondary partition(s) are left to asynchronously execute the transaction.

3) **Asynchronous replication:** Execution of a transaction at a primary partition does not halt to wait for any acknowledgment that the transaction has reached the secondary partition(s); the primary partition assumes that the transaction will be executed at the secondary partition(s) and continues its own transaction execution processes.

I expected for asynchronous replication to cause only a negligible performance decrease and for semi-synchronous replication to cause a slight performance decrease. Synchronous replication should have significant performance overhead, though, as it essentially doubles the execution time per transaction–the transaction execution at the primary blocks for nearly the entire time that the transaction is executing at the secondary. I compare these three replication schemes with the performance of H-Store when no replication occurs–i.e. the state of H-Store before my implementation of replication.

2 Implementation

My implementation of replication in H-Store intercepts the execute transaction call on the primary partition, packages the relevant transaction information, and sends it to the secondary partition. Depending on the replication scheme, my implementation either blocks until it receives an acknowledgment callback from the secondary or proceeds with execution as normal. Sending replicated transactions required creating new RPCs that wrap the information that must be sent to the secondary. The new RPCs required new classes of callbacks that ensure the transaction is executed at the secondary. Finally, I had to accommodate for necessary, but non-single-partitioned, transactions that must be treated separately from ordinary local transactions.

2.1 Specifying partitions upon setup

Partition replicas are specified in partition configuration plans in the `scripts/reconfiguration/plans/` directory. Ordinarily, partition configuration plans detail which transactions should be directed to which partitions; however, these plans can also define a replication scheme by specifying the replicas for each primary. Furthermore, the ExplicitPartitions class processes this information such that it populates an internal mapping from primary partitions to their secondary partitions. The HStoreSite class can access this mapping through the PlannedHasher and TwoTieredRangeHasher classes; my implementation then accesses primary and secondary partition information through the HStoreSite class.

2.2 Interception at the partition executor

The `PartitionExecutor` class has a single method by which it executes ordinary transactions–transactions of the class `LocalTransaction`. This method is `executeTransaction( LocalTransaction ts)`. Immediately before the transaction is executed, my implementation checks whether the partition on which the transaction is executing is a primary partition. If it is a primary, I first generate the appropriate transaction replication callback, then iterate through the corresponding list of secondary partitions and package information about the original transaction to send to each secondary. After the transaction is serialized into a new `StoredProcedureInvocation`, I then send the transaction to the `HStoreCoordinator`, which will deliver the serialized transaction to the appropriate secondary partition. Note that only transactions that write data are replicated; there is no need to replicate read-only transactions.

2.3 Replicating transaction via RPC

The `HStoreCoordinator` accepts the information about the replicated transaction from the `PartitionExecutor` and generates a new request to be passed through an RPC created to forward replicated transactions to secondary partitions. This new `Transaction-ForwardToReplicaRequest` contains information about the sender site, the original transaction ID, and the serialized transaction. The `HStoreCoordinator` sends out the RPC to the destination site; the reception method for this RPC is a newly generated `transactionForwardTo- Replica` method.

The `HStoreCoordinator` of the secondary partition receives the serialized transaction, generates a callback for the transaction containing a `ClientResponse`, and queues it up for execution. If a semi-synchronous replication scheme is used, an acknowledgment is sent to the primary that the transaction was received. Next, the secondary partition's separate work thread will pick up the transaction and execute it with the `PartitionExecutor`, skipping the replication call within the execute transaction method.

After the partition has executed on the secondary, it processes the `ClientResponse` that was coupled with the transaction when it was queued for execution. When it processes this response, I recognize that the transaction has finished executing; if following a synchronous replication scheme, an acknowledgment is then sent back to the primary that the transaction has successfully executed on the secondary.

With both semi-synchronous and synchronous replication schemes, secondary partitions must alert primary partitions when they are at some state in executing the replicated transaction. To do this, I use another RPC; this RPC is sent via the secondary partition's `HStoreCoordinator` to the primary partition's `HStoreCoordinator`. The primary's coordinator contains a mapping of transactions to semaphores; the internal status of these semaphores indicate the number of secondaries the primary must yet hear acknowledgment from before it can proceed with its own execution of the transaction. Upon receiving this ack RPC from the secondary, a permit for the corresponding semaphore is released. All permits being released from a semaphore indicates that all secondary partitions have acked the transaction, and transaction execution resumes on the primary.

2.4 System procedure call handling

System procedures must be handled separately from ordinary transactions, as they are not executed through the same process. Since I focus on single partition transactions, the only system procedure that must be replicated is `LoadMultipartitionTable`. Due to the

existing architecture of H-Store, these system procedures are sent to all partitions, regardless of their status as a primary or secondary partition. To account for this, my replication implementation prevents a transaction from being replicated in the same manner as ordinary local transactions. Instead, I intercept execution of the system procedure earlier in the transaction reception step and prevent execution if it is not occurring on a primary partition. If execution is on a primary partition, I then send an RPC to the secondary with the necessary parameters and information about the `LoadMultipartitionTable` system procedure. The secondary wraps the necessary information into a `ReplicaLoadTableMessage` object and then inserts this `InternalMessage` into the local work queue. This message is then picked up through normal execution and the appropriate data is loaded into the secondary's data store.

2.5 Code-level replication specification

The replication scheme being used can be accessed by a developer with the following call:
`ReplicationType.get(hstore_conf.site.replication_protocol);`
This method returns a `ReplicationType` that is either `NONE`, `ASYNC`, `SYNC`, or `SEMI`; a developer can then use the setting if the need arises.

2.6 High-level replication specification

All three replication schemes are implemented within the H-Store system–synchronous, semi-synchronous, and asynchronous replication. At a high level, my implementation exposes a configuration parameter that can be set from the command line when H-Store is run; this configuration variable allows the user to set the replication scheme as "none," "sync," "semi," and "async." A user simply sets the parameter as follows.

```
-Dsite.replication_protocol={none|sync|semi|async}
```

3 Benchmarks

I evaluated the performance of the replication schemes by measuring throughput and latency when running H-Store for two minutes after a thirty second warmup. I used ten client threads and a client transaction rate of 300,000 transactions per second, intending to place strain on the system and determine the highest throughput possible with each replication scheme. To gather quantitative data, I seeded the database with 10,000 records and ran the standard YCSB benchmark on the database. The benchmark included only reads and

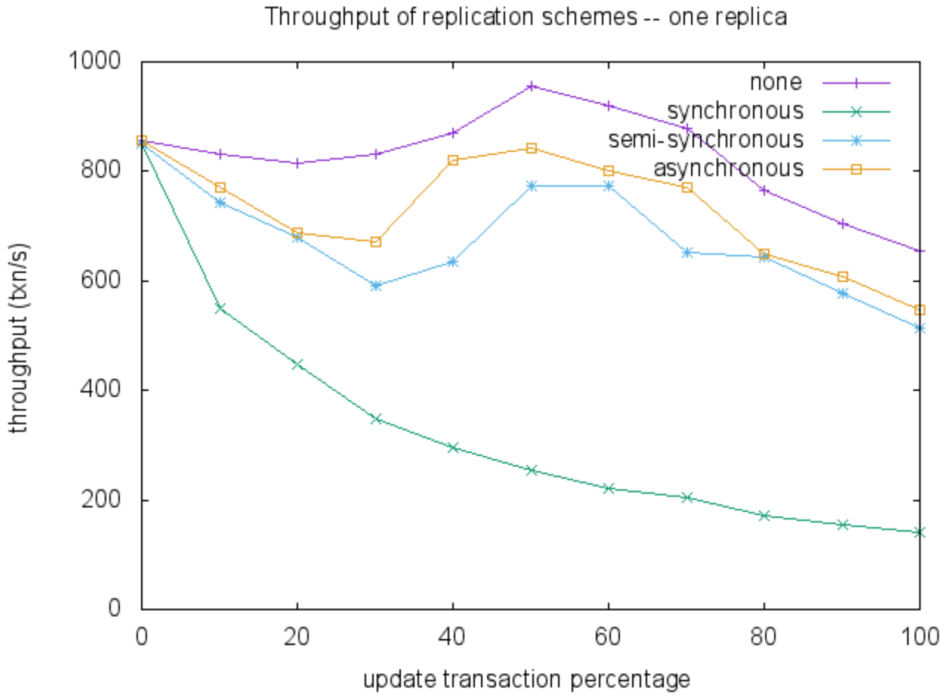

Figure 1: Graph of throughput of synchronous, semi-synchronous, and asynchronous replication schemes, as well as throughput of a control with no replication, on various YCSB workload compositions for a configuration with one primary partition and one replica partition.

updates; by varying the percentages of transactions that are reads and updates, I evaluate the effect of the various replication schemes on databases that experience different types of workloads. I first analyzed the performance of H-Store with a configuration consisting of one primary partition and its replica. These benchmark results are depicted in Figures 1 and 2.

As seen in Figure 1, the synchronous replication scheme resulted in the worst, i.e. lowest throughput. The semi-synchronous and asynchronous replication schemes performed similarly, with the asynchronous replication scheme generally having slightly higher throughput than the semi-synchronous replication scheme. As expected, H-Store had the highest throughput when no replication took place.

Overall, the performance of H-Store running a workload consisting of only reads is consistent across all replication types–this is expected, as I do not replicate reads under any replication scheme. As the percentage of update transactions increases and the percentage of read transactions decreases, the throughput of H-Store under a synchronous replication scheme also decreases; however, this rate of decrease tapers out at higher update transac-

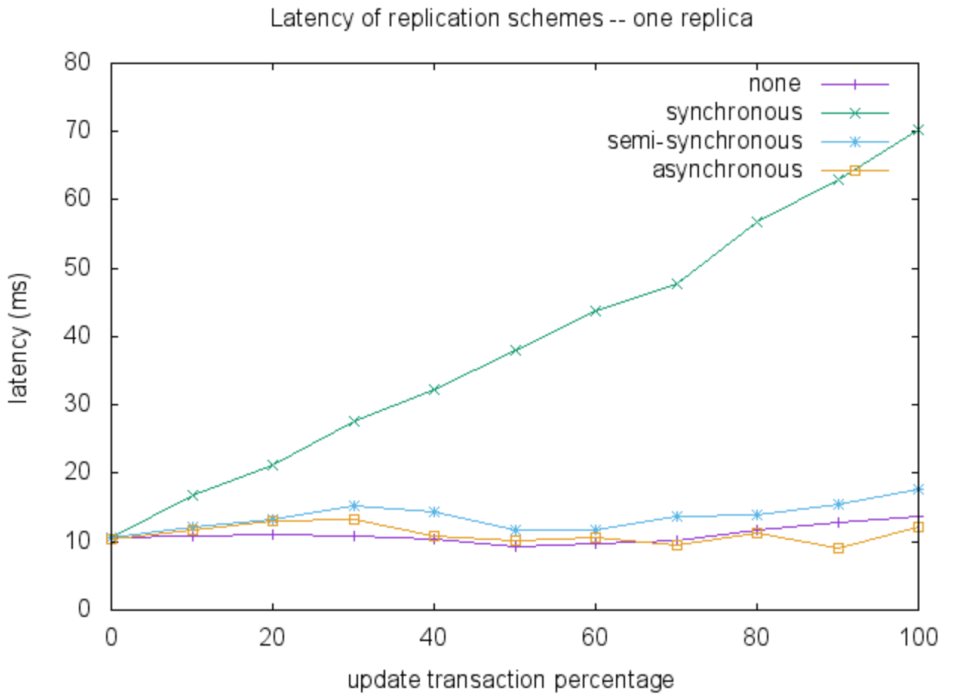

Figure 2: Graph of latency of synchronous, semi-synchronous, and asynchronous replication schemes, as well as latency of a control with no replication, on various YCSB workload compositions for a configuration with one primary partition and one replica partition.

tion workload compositions. The throughput of the other three replication schemes seem to vary within 25% of the throughput under a workload consisting of 100% reads, but generally dip as the percentage of transactions that are updates approaches 100%.

As seen in Figure 2, the synchronous replication scheme by far resulted in the highest latency values. As the workload composition includes more update transactions, the latency of the synchronous replication scheme increases linearly, until it reaches nearly five times the latency of any other replication scheme, as well as the control. On the other hand, the semi-synchronous and asynchronous replication schemes result in latency values generally within 30% of the latency under the control setting, with the semi-synchronous replication scheme generally having the highest latency of the three.

To shed insight on changes in overall performance with more replicas, I next analyzed the performance of H-Store with a configuration consisting of one primary partition and two replicas. These results are depicted in Figures 3 and 4.

As seen in Figure 3, when data is replicated onto two replicas, the throughput data collected when H-Store is run under the semi-synchronous and asynchronous replication schemes show a trend similar to that of the synchronous replication scheme. Notably, this

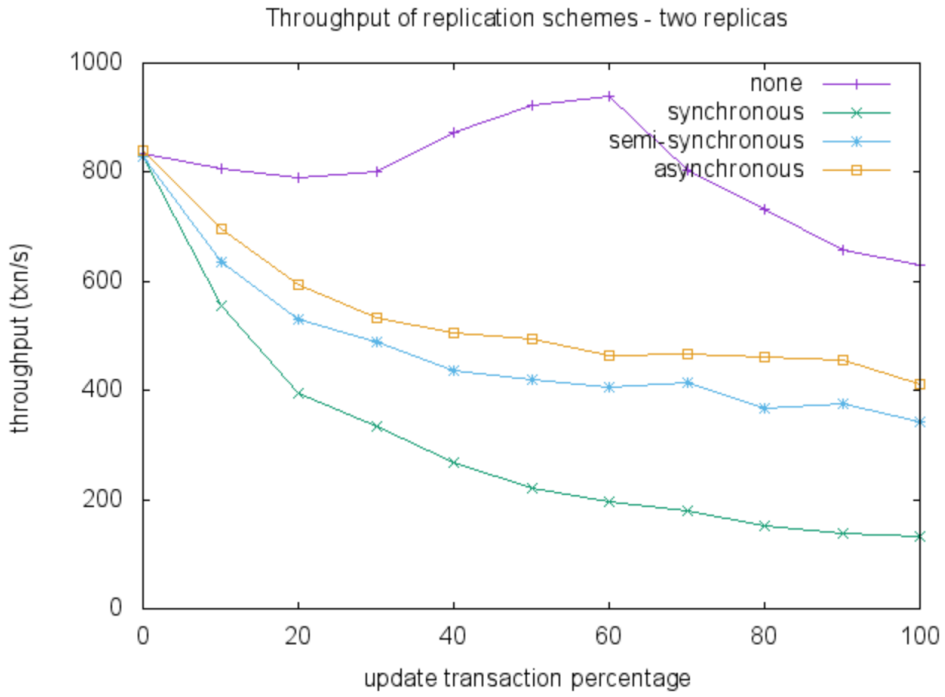

Figure 3: Graph of throughput of synchronous, semi-synchronous, and asynchronous replication schemes, as well as throughput of a control with no replication, on various YCSB workload compositions for a configuration with one primary partition and two replica partitions.

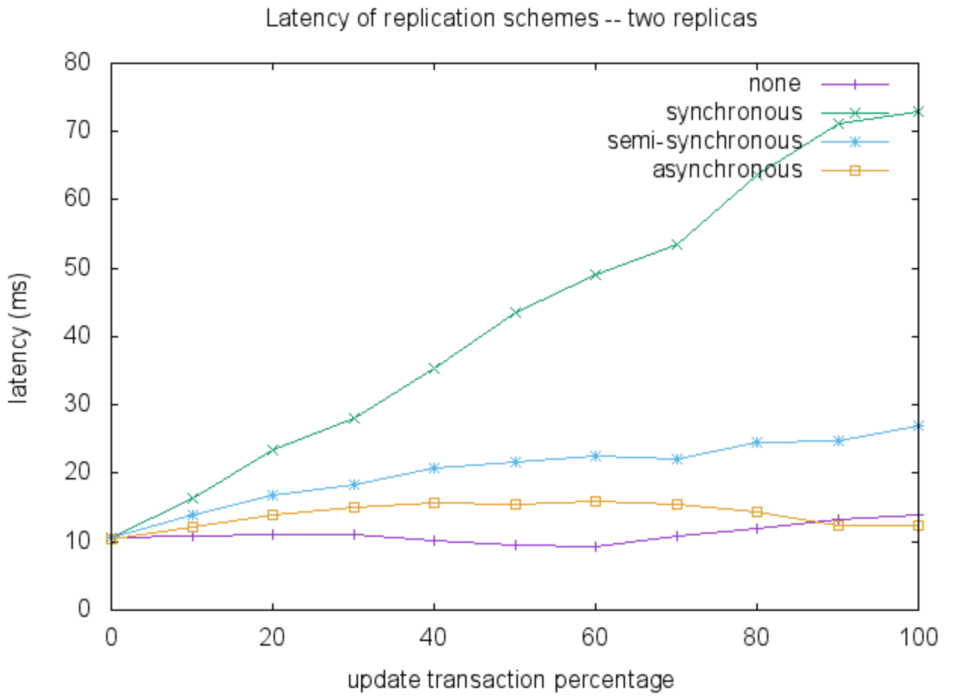

Figure 4: Graph of latency of synchronous, semi-synchronous, and asynchronous replication schemes, as well as latency of a control with no replication, on various YCSB workload compositions for a configuration with one primary partition and two replica partitions.

is different from the throughput data collected when the primary partition had only one replica; as seen in Figure 1, the throughput for the semi-synchronous and asynchronous schemes more closely follows the throughput for no replication. Instead, when there are more replicas, a greater percentage of transactions being updates had a distinctly negative effect on the throughput for these two replication schemes, decreasing the throughput by a factor of two. With one replica, the downward trend in throughput seen in Figure 3 is less pronounced.

Figure 4 shows the effect of an increasing percentage of updates with two replicas is very similar to the effect with one replica. The only notable difference is that with two replicas, latency for semi-synchronous replication increases as the percentage of workload transaction that are updates increases; this effect is not as significant with only one replica. As seen, latency for semi-synchronous replication increases by a factor of nearly three as the percent composition of update transactions increases, whereas with one replica, the increase is around 75%.

4 Analysis

As expected, H-Store achieved the best performance with respect to throughput and latency metrics when no replication occurred. The second highest throughput and second lowest latency values were reported when H-Store was run with the asynchronous replication scheme. Semi-synchronous replication was third optimal with respect to performance, followed by synchronous replication.

Since performance is of paramount importance in DBMSs, fully synchronous replication is clearly a suboptimal choice. However, the guarantee that it provides is highly valuable; with synchronous replication, every transaction is guaranteed to be replicated at the secondary. Otherwise, the database will not return indication that the transaction was successful. On the other hand, asynchronous replication, the replication scheme with the best performance, provides no such strong guarantee. Given the implementation of replication, then, the user must balance the risk of stale data on the replica with the performance overhead of synchronous replication. Since replication is only useful if there is a risk that the primary will become unavailable, it is unsafe to assume that transactions sent by the primary will always reach the replicas. So, while the most performant replication scheme, asynchronous replication is unlikely to be the most desirable choice.

Semi-synchronous replication strikes a happy medium between fully synchronous replication and asynchronous replication; transactions are acknowledged once it reaches the replica, but it is up to the replica to ensure that the transactions are actually carried out. This replication scheme could be ideal if, in the event of a crash, recovery processes could ensure that any transactions that reached the replica partition are carried out. Indeed, past research on H-Store has investigated novel recovery techniques; perhaps some of these ideas could be expanded to make replication more hardy to partition failures. [2] Since semi-synchronous replication shows throughput and latency metrics similar to asynchronous replication, this replication scheme seems to achieve relatively good performance as well as a data replication guarantee.

Unfortunately, my results indicate that the performance costs of even semi-synchronous and asynchronous replication with more than one replica per primary partition would be significant. Specifically, even these replication schemes cause throughput to halve when there are two replicas per primary; with more replicas per primary partition, this effect would likely worsen. Further study is needed to determine how to avert this slowdown as the number of replicas is scaled up.

5 Future Work

The current implementation of replication does not currently have robust measures for fault tolerance. Namely, if a transaction is lost at the secondary partition, it will not be re-sent by the primary. Instead, if the replication scheme is asynchronous, as discussed in the previous section, the secondary will then have stale data. If the replication scheme is semi-synchronous or synchronous, and the transaction is lost when sent to the replica, the execution of the transaction at the primary will be blocked.

As such, future work could explore several methods to deal with lost transactions due to network or machine failures. With the asynchronous replication scheme, the primary could follow up on transactions sent to the secondary, and re-send them if it does not receive acknowledgment from the secondary after a certain time that the transaction was executed. This strategy would not require the primary to block on any secondary acknowledgment; instead, the primary must make sure that all transactions are eventually replicated on the secondary. Implementing this fault tolerance would require the primary to have a recovery mechanism for the structure that tracks which transactions have been acknowledged by which replicas. Checks to prevent out-of-order execution of transactions on replicas would also have to be in place.

With a semi-synchronous or synchronous replication scheme, implementing a timeout mechanism, after which a transaction would be re-sent to a replica, would avoid the entire system stalling when a transaction is lost. Alternatively, if a timeout is reached, the primary could just report a transaction failure for that transaction without re-sending the transaction to the replica. In that case, to maintain consistency between the primary and the replicas, a two phase commit strategy would be used.

6 Conclusion

Overall, the use of any replication scheme will have a negative impact on H-Store's performance. The extra time required to send a replicated transaction to a secondary naturally decreases throughput and increases latency; waiting for the replicated transaction to be acknowledged by the secondary further impacts performance. However, semi-synchronous replication provides a balance between ensuring that the transaction is fully replicated on a replica and significantly impacting performance. While semi-synchronous replication seems to be the dominant strategy among the strategies I explored, future research is needed to avoid significant decreases in performance as the number of replicas is scaled.

References and Notes

[1] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi, "H-Store: a high-performance, distributed main memory transaction processing system," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1496–1499, 2008.

[2] N. Malviya, A. Weisberg, S. Madden, and M. Stonebraker, "Rethinking main memory oltp recovery," in *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pp. 604–615, March 2014.