Styx++: Reliable Data Access and Availability Using a Hybrid Paxos and Chain Replication Protocol

Ather Sharif
asharif@cs.washington.edu
Paul G. Allen School of Computer Science & Engineering | DUB Group, University of Washington
Seattle, Washington, USA

Emilia F. Gan*
efgan@cs.washington.edu
Paul G. Allen School of Computer Science & Engineering, University of Washington
Seattle, Washington, USA

Miranda Wei*
weimf@cs.washington.edu
Paul G. Allen School of Computer Science & Engineering, University of Washington
Seattle, Washington, USA

ABSTRACT

HCI research often involves accessing and storing information in databases. However, in case of a database node failure, researchers could experience significant work delays, monetary costs, and data loss. How can researchers who have little or no knowledge of systems and infrastructures ensure that their data collection source is reliable and maximally available for accessing and storing data? To answer this question, we surveyed 11 HCI researchers. Using the findings from the survey, we developed Styx++—an easy-to-integrate open-source solution that bundles together existing tools and concepts, providing HCI researchers with a reliable distributed system for their database needs. Styx++ is a hybrid solution involving both the Paxos and Chain Replication Protocol, providing strong consistency and high availability to minimize the risks of single-point failures in a traditional database system setup. Our evaluation of Styx++ against benchmark solutions shows promising results of an increase in reliability without substantial performance degradation.

CCS CONCEPTS

• **Computer systems organization** → **Reliability**; **Availability**; • **General and reference** → *Design*; • **Information systems** → **Parallel and distributed DBMSs**.

KEYWORDS

Paxos, Chain Replication Protocol, database, reliability, consistency, availability

ACM Reference Format:
Ather Sharif, Emilia F. Gan, and Miranda Wei. 2022. Styx++: Reliable Data Access and Availability Using a Hybrid Paxos and Chain Replication Protocol. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts (CHI '22 Extended Abstracts), April 29-May 5, 2022, New Orleans, LA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3491101.3519635

*These authors contributed equally to this work.

CHI '22 Extended Abstracts, April 29-May 5, 2022, New Orleans, LA, USA

ACM ISBN 978-1-4503-9156-6/22/04.
https://doi.org/10.1145/3491101.3519635

1 INTRODUCTION

Databases are used widely for accessing and storing information in research projects. This is particularly true for fields such as Human-Computer Interaction (HCI), in which recording the data from users accurately and reliably from studies is crucial [18]. Specifically, databases are vital in performing data collection, analysis, and manipulation, which govern all aspects of society [9, 23].

However, databases, similar to any other computer system, can fail anywhere and at any time. In case of a database node failure, whether temporary or permanent, access to data terminates until the node is restored or replaced. Therefore, even with mitigation strategies, a database node failure comes at a considerable cost to the researchers. Additionally, for researchers collecting a large amount of data, such as through crowdsourcing, there might be significant work delays (*e.g.,* setting up a new node, waiting to retrieve from a stored backup, etc.). These challenges are not only an inconvenience to researchers but could also incur monetary costs. Furthermore, most HCI researchers are unfamiliar with the knowledge and concepts (such as Distributed Systems) needed to tackle these challenges.

To address these concerns, we developed Styx++—an open-source easy-to-integrate solution that ensures data consistency and high availability. Using existing technologies, Styx++ gives HCI researchers the convenience and reliability of a well-implemented distributed system without requiring them to design and maintain the necessary infrastructure themselves. In a nutshell, Styx++ improves the reliability of the system, enabling another database node to replace a failed node automatically without any significant performance degradation.

In developing Styx++, we surveyed 11 HCI researchers who utilize databases to access and store data in their research work. We found that most researchers are not interested in acquiring knowledge about systems-specific concepts but would be willing to incorporate a tool with minimum learning overhead and easy integration. We used these findings to develop Styx++, ensuring that it was easy to set up and manage. Styx++ is a hybrid between Paxos [13] and Chain Replication Protocol [28], with a high degree of similarity to a Vertical Paxos [15] system. In particular, we make use of Paxos for the `Controller` node that manages the configuration of the chain and Chain Replication Protocol for the database nodes. This setup benefits us in ensuring strong consistency and supporting the `fail-stop-restart` model, which allows recovered nodes to be re-added to the chain.

We evaluated Styx++ against a benchmark solution using `K6` [4]—an open-source performance testing tool. We report that even

though Styx++ increased the total number of database nodes, the overall performance of the system did not significantly degrade. We discuss the findings from the survey and performance tests in this paper.

The main contributions of our work are as follows:

(1) Styx++, an easy-to-integrate open-source solution that bundles together existing tools and concepts, providing HCI researchers with a reliable distributed system for their database needs. We present its design and architecture, functionality, and operations. Additionally, we open-source our implementation at https://github.com/athersharif/styx.
(2) We evaluated Styx++ using performance testing against a benchmark solution. Our findings show that Styx++ improved the reliability of the overall system without significantly degrading the performance of the system. Specifically, on average, from over 1000 iterations, Styx++ only took about 56*ms* and 342*ms* more time compared to the benchmark server.

2 RELATED WORK

In this paper, we present Styx++, a tool that combines the concepts and implementations from Paxos [8, 13, 14], Chain Replication Protocol [12, 28], and Vertical Paxos [15, 27], which we discuss below. Several distributed database systems [7, 21, 22] exist that attempt to improve data availability [5, 10, 19, 24, 26]. However, to the best of our knowledge, we are the first to explore a solution that provides HCI researchers with a fully implemented distributed system to minimize the risks of single-point-of-failure database systems.

2.1 Paxos

In a distributed system, a common practice is to store data in several other computers ("replicas"). When and if the main computer ("controller") becomes unresponsive, one of the replicas serves as the controller, ensuring that data is continually available without any disruption. Therefore, it is of pivotal importance that the stored data stays consistent amongst all the replicas. The Paxos protocol [13, 27] ensures consensus among replicas by having nodes within a Paxos group communicate to reach an agreement on what is committed to each replica. Styx++ is built using `Consul` [3], which implements Raft [20]—a consensus algorithm based on Paxos.

2.2 Chain Replication Protocol

The Chain Replication Protocol [28] is a replication protocol that guarantees strong consistency and consists of a chain containing the `head`, `tail`, and middle nodes. `Write` operations are delivered to the `head` of a chain of nodes, from where it propagates towards the tail with each node in the chain executing the update. On the other hand, `read` operations are served from the `tail` node, with the implicit assumption that every node that exists in the chain will be up-to-date. Styx++ constructs a chain of nodes, linked in a way that resembles the linked nodes in the Chain Replication protocol. In contrast to the standard Chain Replication Protocol that follows a `fail-stop` model, Styx++ supports a `fail-stop-restart` model, supporting nodes to rejoin the chain when they recover from failure.

2.3 Vertical Paxos

When a node fails in a distributed system, part of the recovery process involves setting up a replacement node that contains the knowledge of the state of the system. This state transfer process can be slow, leading to disruptions in the availability of the system. Vertical Paxos [15] was developed to address this issue, enabling state transfer and continued operation of the system to occur concurrently. In a Vertical Paxos implementation, an external controller—a replicated state machine, manages the global system state [15]. Styx++ contains a `Controller` node, which is synonymous with the auxiliary configuration controller in Vertical Paxos in both concept and functionality.

3 PRELIMINARY SURVEY

To inform our design choices for Styx++, we conducted a preliminary survey with 11 HCI researchers. The survey was advertised using a word-of-mouth strategy and shared on relevant social media channels. First, we asked respondents to report their area of research and their expertise with distributed systems and applications. Then, we asked about specific backup practices of databases they had used in the past. We also asked about their intended actions and potential implications in a hypothetical database server failure scenario. We concluded by eliciting their knowledge of current solutions that would provide system availability and data consistency in case of server failures and their interest in using such systems. Participants were entered into a drawing for a $20 Amazon gift card as compensation for their time in taking the survey.

We used an open coding procedure to qualitatively analyze responses to two free-response questions: recovery steps taken in a hypothetical database server failure scenario (Q1) and inconveniences faced in such a situation (Q2). One co-author iteratively developed a codebook with thematic codes, identifying 8 codes for Q1 and 4 for Q2; codes were not mutually exclusive. A second member of the research team independently coded the full data set. Inter-coder reliability (IRR), measured with Cohen's κ, was 0.70 (for Q1) and 1.00 (for Q2), considered "substantial" and "almost perfect," respectively [16].

3.1 Results

Notably, none of our respondents identified as being Proficient or an Expert in distributed systems: six out of 11 respondents (54.5%) reported being a Novice, three (27.3%) reported being a Beginner, and the remaining reported being Competent. However, all respondents had worked with reading or writing information in a database, and five respondents (45.5%) reported being Competent in developing applications involving databases.

Six respondents (54.5%) managed backups for their databases using automatically generated backups. Three (27.3%) manually exported, and the remaining two respondents (18.2%) did not have backups. 10 respondents (90.9%) reported the importance of acquiring distributed systems knowledge for their research as "not important."

Eight respondents did not know of any out-of-the-box solutions providing system availability and data consistency without knowledge of distributed systems. Nine (81.8%) respondents expressed it would be unimportant to be concerned about database server

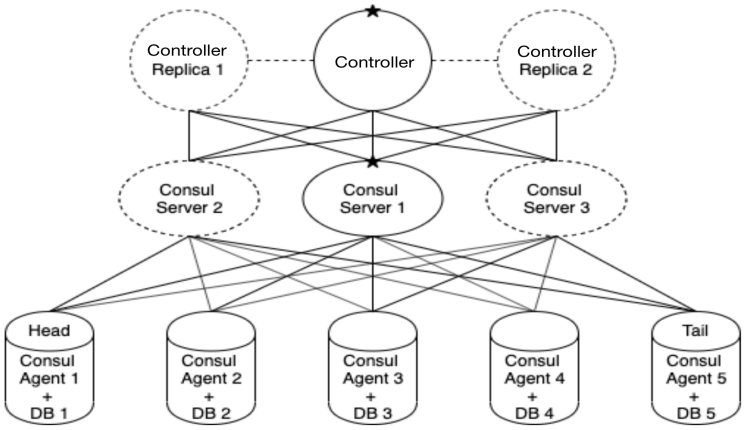

Figure 1: Styx++ employs a three-tier system: Tier 1 contains the `Controller` and its replicas, which act as the configuration managers for Styx++, Tier 2 contains the `Consul` server and its replicas, which are responsible for monitoring and health checking the database nodes, and Tier 3 contains the database nodes that consist of a `Consul` agent that the `Consul` server communicates with and the database server that executes a client request. The first node in the chain is the `head` whereas the last node in the chain is considered as the `tail`. The star represents the leader of the Paxos group.

crashes or failures; the remaining two participants were neutral. Respondents expressed that they would be interested in such a system only if the effort of usage were minimal.

Finally, in our survey, we presented respondents with a hypothetical study scenario of a crowdsourced study that was constantly recording data in a database; however, the server hosting their database crashed before the most recent backup was made. We asked respondents what steps they would take to recover the server and the data. Four respondents said they would check the logs to determine what happened, whereas three said they would restart the database. Four respondents said they would restore from an older backup, accepting the data loss. Three respondents said they would try to recover the data through workflows besides the database, such as contacting participants directly to request the data or writing additional scripts.

For nine respondents, the main concern was the time lost. Additionally, four respondents (36.4%) mentioned the implications of potentially missing data, three (27.3%) mentioned monetary loss of the project, and two expressed (18.2%) negative emotions such as frustration. To our surprise, one respondent said there would be no implications at all.

4 STYX++

4.1 Overview

Styx++ is an easy-to-integrate solution that bundles together existing tools and concepts to provide HCI researchers with a reliable distributed system for their database needs. Styx++ implements a hybrid of Paxos [13] and the Chain Replication Protocol [28] to provide strong consistency and improved availability for database nodes by reducing the chance of a single point of failure. Styx++ takes input from users in a simple question and answer format and outputs a fully deployable system configuration.

4.2 Architecture

Styx++ consists of two main components: `Controller` and `Node Chain`. Figure 1 illustrates the architecture of the Styx++ system. The `Controller` in Styx++ is based on the implementation of an auxiliary configuration controller in Vertical Paxos [15] and is responsible for: (1) managing the `Node Chain`; (2) handling requests from the client; and (3) keeping a log of all the requests and their statuses.

The `Node Chain` closely follows the Chain Replication Protocol [28] and is managed by the `Controller`. The chain is created when the system initializes and then adjusted whenever a node fails or recovers. The addition of support for node recovery allows Styx++ to follow a `fail-stop-restart` model, which extends the functionality of the original Chain Replication Protocol that only supports a `fail-stop` model [28].

4.3 Implementation

We implemented Styx++ replicating a network of Virtual Machines ("VM"s) using `Docker` [6] containers. This method does not accurately replicate latencies that would exist in a real-world network. To address this, when conducting the performance tests, we introduced `sleep` functions on each node with a random value between 20 and 200 ms.

4.3.1 Controller. In Styx++, the `Controller` is synonymous with the auxiliary configuration controller in Vertical Paxos [15]. To eradicate a single point of failure in Styx++, we employ `Consul` [3]—a service mesh solution that provides service discovery, configuration, and segmentation functionality. Additionally, `Consul` provides an easy-to-use interface (Figure 2). However, the user interface uses systems-specific terminologies such as "ACL," "Nodes," and "Intentions." Future work could develop a browser extension to translate these terms that are more broadly understandable.

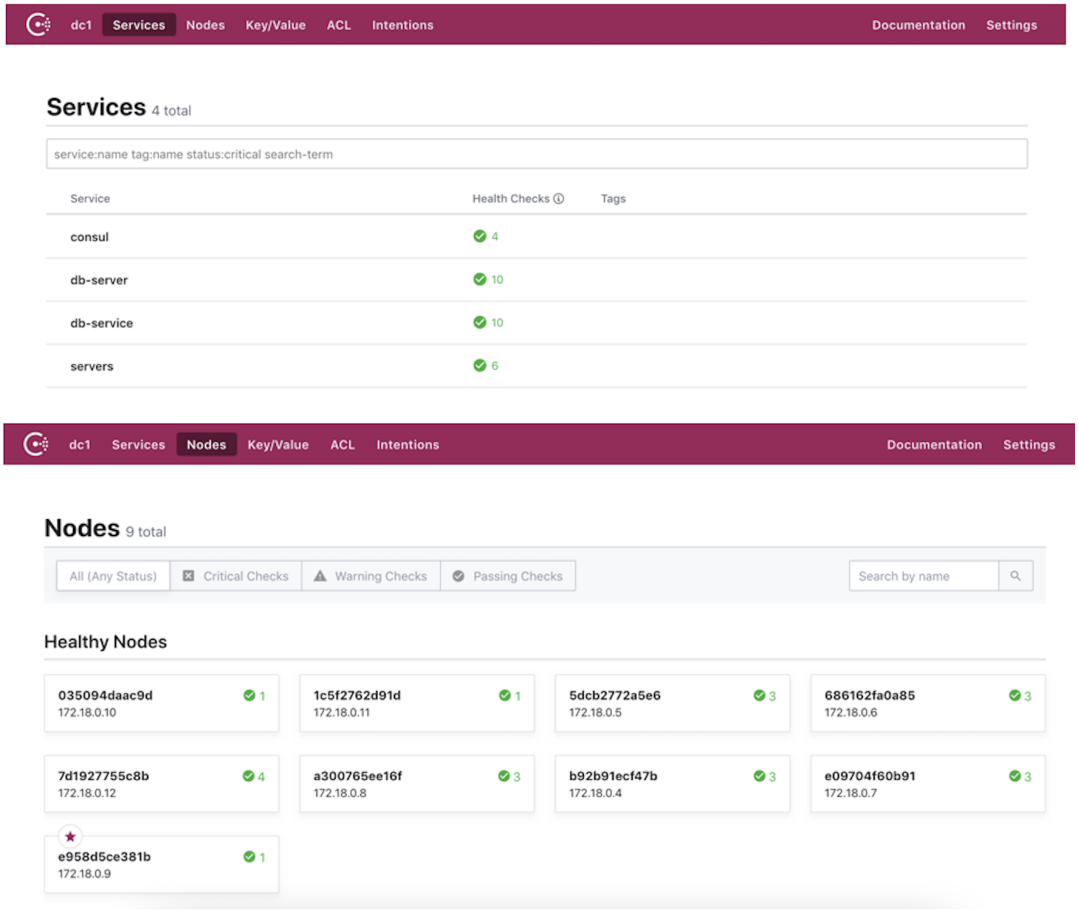

Figure 2: Screenshots showing the `Consul` user interface. The left screenshot (Services) shows the health checks for both the server and services for the database and `Consul`. The right screenshot (Nodes) shows the currently existing nodes and their status. The leader is indicated by the red star.

Under the hood, in order to provide consistency, `Consul` implements Raft [20], which is a consensus algorithm based on Paxos [13]. Given that Raft requires $(n/2) + 1$ servers to form a quorum, our system requires at least 3 nodes that are dedicated to being `Consul` servers.

In our current implementation, a `Controller` runs a lightweight server implemented in `Node.js` [1] and hosts a `Consul` control service that uses the above-mentioned servers to achieve consistency in its key/value store and communicate with the `Consul` agents on the database nodes.

4.3.2 Node Chain. The `Node Chain` is a chain of nodes implemented based on the Chain Replication Protocol [28]. The chain consists of a `head` node, a `tail` node, and n middle nodes that are unidirectionally linked to each other. As in the traditional Chain Replication Protocol [28], the `head` receives the client `write` requests and passes it to the next in the chain until the `tail` node is reached, which then responds to the client. For the `read` requests from the client, the request is sent to the `tail` that responds directly to the client.

Each node runs a database service, a lightweight server that processes the incoming `read` and `write` operations and a `Consul` [3] agent that serves as a health checker (a service used to monitor whether a given node or service is alive and responding) for both the database service and the node server. We used `Postgres` [25] as the database service and `Node.js` [1] as the node server that communicates with the `Postgres` database using the extensive `pg` client library for `Node.js` [2]. To ensure a robust implementation, we support transactional database queries by default. Future work can extend the benefits of Styx++ to databases other than `Postgres`.

4.4 Database Operations

Styx++ classifies database operations into two types: (1) `reads` (for example, `SELECT` query); and (2) `writes` (for example, `INSERT` query). Styx++ executes both these types using database transactions to ensure database consistency and isolation. Additionally, Styx++ also supports operations of the same type in batches, which are also executed as a database transaction.

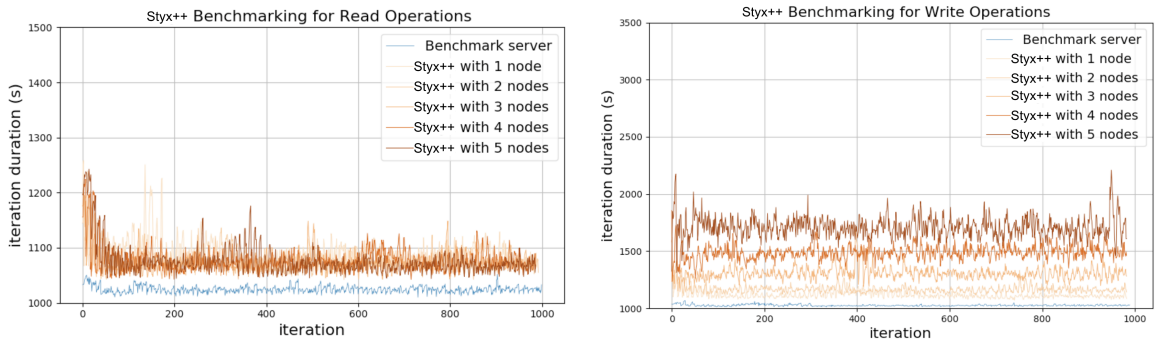

Figure 3: (left) Benchmarking results showing time for read iterations is relatively unaffected as number of nodes increase. (right) Benchmarking results showing time for write iterations takes progressively longer as nodes are added. An "iteration" is a single round-trip operation.

In the Styx++ universe, when the `Controller` receives a request (single or in batch), it begins the execution process by determining if a chain update operation is in progress (see subsection "Node Failure and Recovery"). All incoming requests are queued until the chain has finished updating. After completion, the `Controller` fetches the current chain from the `Consul`'s key/value store and generates a hash for the request. At this point, the next steps are determined by the type of operation and are explained in the subsections below.

4.4.1 Read. For the `read` operation, Styx++ continues the process described above by identifying the `tail` node. Once identified, the request is then entered into the `tail`'s queue as well as its own key/value store and the `Controller`'s key/value store. Then, the request is sent to the `tail` node for processing and the `Controller` awaits its response.

When the `tail` receives a `read` request, it checks its queue to see if there are any `write` operations that entered the queue before the current request. Checking the queue is crucial for consistency, especially if the node has recovered from a failure. If such requests exist, then the `tail` proceeds to process them first, ensuring that the `read`s are consistent with the state of the database. Once these requests are processed, the `tail` executes the current query and collects the response. The `tail` then updates its own and the `Controller`'s key/value store, marking the request as `completed` and appending the request with the result of the query. Finally, the `tail` sends the response to the `Controller`, which then forwards the `tail`'s response to the client.

Styx++ maintains two separate key/value stores for the `Controller` and the nodes. While `Controller` keeps track of the overall requests, the node keeps track of the incoming requests that it received and may have processed. Such an implementation allows Styx++ to provide consistency when a node fails or recovers. Furthermore, it is also helpful for troubleshooting purposes and for providing insights about the overall system on a node level.

4.4.2 Write. The `write` operation follows a similar process as the `read` operation with the exception of a few key differences. First, as in the traditional Chain Replication Protocol [28], the `write` requests are handled by the `head`, as opposed to the `tail` in the `read` operation.

Second, the `Controller` does *not* wait for the response from the `head` node, as the `head` node forwards the request to the next node in the chain, and the chain processes requests in a unidirectional manner. Instead, the `Controller` periodically checks its key/value store to see if the request has been marked `completed`. If such is the case, it returns the result of the request to the client. In the case of a node failure during request execution, a `timeout` threshold of 60*s* is exercised—after which, a `timeout` error is returned to the client.

Finally, during a `write` operation, the request is passed along in the chain until the `tail` node. Each node, upon completion of the request, updates its own key/value store (similar to the `read` operation), finds the next node in the chain, enters the request into the next node's queue, and forwards the request to the next node in the chain. However, in the case that the node is the `tail` node, upon completion of the request, only the `Controller`'s and the tail node's key/value store is updated, as the tail node is the last in the chain.

4.5 Node Failure and Recovery

Styx++ utilizes the `Consul` health check watcher to get notified when a node itself or the database service has either failed or recovered. When such a notification is received, Styx++ starts the chain adjustment process. During this process, Styx++ queues all the incoming requests to prevent any undesirable inconsistency that might occur as a side-effect. Once the chain is readjusted, the queueing of incoming requests is turned off and the execution resumes as usual.

4.5.1 Failure. We identify three node failures cases: (1) failure of `head`; (2) failure of `tail`; and (3) failure of any other nodes other than `head` or `tail`. When `head` fails, the rest of the chain moves one position to its left. The `tail` node stays at the `tail` position. When the `tail` node fails, the node at the position before `tail` assumes the responsibility of `tail`. Finally, if any other node in the chain fails, all nodes after the position of the failed node move one

	Benchmark	1-node Styx++	2-node Styx++	3-node Styx++	4-node Styx++	5-node Styx++
`read`	1.02	1.09	1.07	1.07	1.08	1.07
`write`	1.02	1.12	1.17	1.31	1.49	1.72

Table 1: Average iteration duration for `read` and `write` operations in seconds over 1000 iterations for Benchmark server and Styx++ servers with a varying number of database nodes (1 - 5).

position to the left. Future work can conduct an in-depth analysis of other possible failure scenarios.

4.5.2 Recovery. A traditional Chain Replication Protocol [28] supports the `fail-stop` model. Styx++ extends this functionality by supporting the `fail-stop-restart` model allowing for the nodes to recover and be re-added to the chain at the `tail` position. We did not explore adding an entirely new node into the chain.

4.5.3 State Updates. Before the chain adjustment, the nodes need to have a consistent state. Given a node failure case, where the node before the failed node is m and the node-set containing the node(s) after the failed node is s. For each node t in node-set s, Styx++ achieves its consistent state by (1) taking the difference between the `write` requests in the key/value store of the node m and node t; (2) collecting the requests that exist in node m but are missing in node t; and (3) entering those requests in the queue of node t.

5 PERFORMANCE TESTING

To test the performance of Styx++, we created a benchmark server and 5 Styx++ servers, each consisting of a unique number of nodes between 1 and 5. To replicate a traditional server, the benchmark server also used the `pg` client library [2] to access the database but made direct calls to the database. All servers were programmed in `Node.js` [1]. We used `K6` [4], a performance testing tool, to compare the performance of Styx++ servers with the benchmark server for both the `read` and `write` operations using 10 Virtual Users (VUs) and 1000 iterations. Specifically, we used the `iteration duration` metric for comparison, which accounts for the total iteration duration that comprises the request time, duration, and sleeps. Table 1 shows the iteration duration comparison for both the `read` and `write` operations, while Figure 3 shows benchmarking results for reads and writes, respectively.

Our results show that over 1000 iterations, Styx++ servers took an average of 1.08s (SD=0.01) to perform `read` operations, about 56ms more compared to the benchmark server. For `write` operations, compared to the benchmark server, Styx++ servers took an average of 1.36s (SD=0.25), about 342ms more time, with the average iteration duration proportionally increasing with the increase in the number of nodes. These results indicate that while Styx++ increases the overall iteration duration, the increment is a reasonable trade-off for reliability and availability.

6 DISCUSSION

In this paper, we developed Styx++, a solution that assists researchers in improving the reliability and availability of their database systems without them having to spend significant time and effort in understanding the finer details of distributed systems. Styx++ leverages the theoretical concepts of Paxos and Chain Replication Protocol and the open-source tools such as `Consul` and `Raft` to provide strong consistency and high availability of database nodes. We surveyed 11 HCI researchers to inform our design choices for Styx++. We evaluated Styx++ with performance testing using `K6`, overall finding that Styx++ offers enhanced database reliability without significantly decreasing performance.

The trade-offs between reliability and performance among other factors (such as availability and cost) have been widely discussed [11, 17]. Increasing reliability can result in performance degradation, whereas performance upgrades can decrease reliability. Our performance testing results show that Styx++ servers (varying from one to five nodes) only took about 56ms more than the benchmark server for the `read` operations. However, for `write` operations, the difference on an average was 342ms. As expected, the performance decreased as the number of nodes increased. Therefore, we recommend a Styx++ server with 3 nodes to be a good general fit considering the trade-off between reliability and performance.

6.1 Limitations and Future Work

Styx++'s performance can be improved by investigating other factors and implementation details. For example, `Node.js` is single-threaded; therefore, another run-time environment may result in better performance. To further evaluate the performance and reliability of Styx++, we aim to conduct more user studies with researchers using a mixed-methods approach. Specifically, we plan to employ a user-centered design protocol, utilizing the longitudinal feedback from researchers through task-based experiments and contextual interviews to refine Styx++. Additionally, future studies are warranted to evaluate the learning curve and ease-of-use of the tool.

ACKNOWLEDGMENTS

We would like to thank the reviewers for their helpful comments and suggestions. We also thank Arvind Krishnamurthy and Lequn Chen for their invaluable input and feedback. Lastly, we would like to extend our warmest thanks to Zoey, Gandalf, Tatlim, Mura, and Eno for their feline support and expert *purr*usal of this work.

REFERENCES

[1] 2009. Node.js. https://nodejs.org/

[2] 2010. Postgres. https://node-postgres.com/

[3] 2014. Consul by HashiCorp. http://consul.io/

[4] 2016. Performance testing for developers, like unit-testing, for performance. https://k6.io/

[5] Sérgio Almeida, João Leitão, and Luís Rodrigues. 2013. ChainReaction: A Causal+ Consistent Datastore Based on Chain Replication. In *Proceedings of the 8th ACM European Conference on Computer Systems* (Prague, Czech Republic) *(EuroSys '13)*. ACM, New York, NY, USA, 85–98. https://doi.org/10.1145/2465351.2465361

[6] Charles Anderson. 2015. Docker [software engineering]. *Ieee Software* 32, 3 (2015), 102–c3.

[7] Stefano Ceri, Barbara Pernici, and Gio Wiederhold. 1987. Distributed database design methodologies. *Proc. IEEE* 75, 5 (1987), 533–546.

[8] Tushar D Chandra, Robert Griesemer, and Joshua Redstone. 2007. Paxos made live: an engineering perspective. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*. 398–407.

[9] C.L. Philip Chen and Chun-Yang Zhang. 2014. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences* 275 (2014), 314–347. https://doi.org/10.1016/j.ins.2014.01.015

[10] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. 2013. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)* 31, 3 (2013), 8.

[11] Batya Friedman, Nancy Levenson, Ben Shneiderman, Lucy Suchman, and Terry Winograd. 1994. Beyond accuracy, reliability, and efficiency: criteria for a good computer system. In *Conference Companion on Human Factors in Computing Systems*. 195–198.

[12] Scott Lystig Fritchie. 2010. Chain replication in theory and in practice. In *Proceedings of the 9th ACM SIGPLAN workshop on Erlang*. 33–44.

[13] Leslie Lamport. 2001. Paxos Made Simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)* (December 2001), 51–58. https://www.microsoft.com/en-us/research/publication/paxos-made-simple/

[14] Leslie Lamport. 2005. Generalized consensus and Paxos. (2005).

[15] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. 2009. Vertical Paxos and Primary-backup Replication. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing* (Calgary, AB, Canada) *(PODC '09)*. ACM, New York, NY, USA, 312–313. https://doi.org/10.1145/1582716.1582783

[16] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33, 1 (1977), 16.

[17] John Lange, Alexandros Labrinidis, and Panos K Chrysanthis. 2014. Towards automated personalized data storage. In *2014 IEEE 30th International Conference on Data Engineering Workshops*. IEEE, 278–283.

[18] Fei Li and HV Jagadish. 2012. Usability, Databases, and HCI. *IEEE Data Eng. Bull.* 35, 3 (2012), 37–45.

[19] Bruce G. Lindsay. 1987. A retrospective of R*: a distributed database management system. *Proc. IEEE* 75, 5 (1987), 668–673.

[20] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference* (Philadelphia, PA) *(USENIX ATC'14)*. USENIX Association, Berkeley, CA, USA, 305–320. http://dl.acm.org/citation.cfm?id=2643634.2643666

[21] M Tamer Ozsu and Patrick Valduriez. 1991. Distributed database systems: Where are we now? *Computer* 24, 8 (1991), 68–78.

[22] M Tamer Özsu and Patrick Valduriez. 1999. *Principles of distributed database systems*. Vol. 2. Springer.

[23] Aisha Siddiqa, Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Mohsen Marjani, Shahabuddin Shamshirband, Abdullah Gani, and Fariza Hanum Nasaruddin. 2016. A survey of big data management : Taxonomy and state-of-the-art.

[24] Michael Stonebraker, Paul M Aoki, Witold Litwin, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. 1996. Mariposa: a wide-area distributed database system. *The VLDB journal* 5, 1 (1996), 48–63.

[25] Michael Stonebraker and Lawrence A Rowe. 1986. *The design of Postgres*. Vol. 15. ACM.

[26] Jeff Terrace and Michael J. Freedman. 2009. Object Storage on CRAQ: High-throughput Chain Replication for Read-mostly Workloads. In *Proceedings of the 2009 Conference on USENIX Annual Technical Conference* (San Diego, California) *(USENIX'09)*. USENIX Association, Berkeley, CA, USA, 11–11. http://dl.acm.org/citation.cfm?id=1855807.1855818

[27] Robbert Van Renesse and Deniz Altinbuken. 2015. Paxos Made Moderately Complex. *ACM Comput. Surv.* 47, 3, Article 42 (Feb. 2015), 36 pages. https://doi.org/10.1145/2673577

[28] Robbert van Renesse and Fred B. Schneider. 2004. Chain Replication for Supporting High Throughput and Availability. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6* (San Francisco, CA) *(OSDI'04)*. USENIX Association, Berkeley, CA, USA, 7–7. http://dl.acm.org/citation.cfm?id=1251254.1251261