A Distributed Transaction Processing System for Airplane Ticketing

KENNETH PAT (56281088), KEMING LI (86244574), YAZAD SIDHWA (37928848)
UNIVERSITY OF CALIFORNIA, IRVINE

1 ABSTRACT

This project introduces an effective system for handling transactions in a widely spread airline ticketing application. Building upon existing transaction methods, we develop new ways to ensure smooth and fast operations in storage systems that are spread across different locations. The system focuses on three main areas: flights, tickets, and user information. It is carefully designed to process transactions in a way that avoids complications and improves efficiency. We've also implemented a smart strategy for storing data close to where users are, which speeds up response times and balances the workload throughout the network.

The key aspect of our work is the custom-made method we've developed for the unique needs of airline ticketing systems. This method is crucial for maintaining a good balance between keeping data consistent and handling a large number of transactions efficiently. Our approach to distributing the data is also very important. It's based on a deep understanding of how users are spread out geographically, ensuring that the data is stored as close as possible to where it's needed.

Manuscript submitted to ACM

Our report goes into detail about every part of the system, from the initial stages of developing the application and its transaction processes to the strategic way we've distributed the data. We've thoroughly tested our system to measure its response times and efficiency under different conditions and workloads. These tests are vital for showing that our system can work well in real-life situations. In summary, this report not only explains the technical achievements of our transaction processing system but also highlights its potential to change the way distributed applications are used in the aviation industry and other areas.

ACM Reference Format:
Kenneth Pat (56281088), Keming Li (86244574), Yazad Sidhwa (37928848), University of California, Irvine. 2020. A Distributed Transaction Processing System for Airplane Ticketing. In *Proceedings of the 2020 International Computing Education Research Conference (ICER '20), August 10–12, 2020, Virtual Event, New Zealand*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3372782.3406266

2 INTRODUCTION

In the current state of the world, the aviation industry and sector is heavily relying on extremely effective transactions processing systems in this era of virtual connection. Within this report, we go through a baseline system that we developed which helps to showcase how complex transaction systems in the real-world specifically to airlines can truly be. With the increasing sheer volume of data that is being transmitted across different geographical regions and the need for very quick access, normal and more traditional centralized databases fall short when trying to provide the required speed and reliability that is necessary to provide an efficient system of interconnectivity. By utilizing the power of distributed databases and systems, our system aims to overcome all of these issues and limits by ensuring quick transaction processing and consistency of the data across a large span of distance.

3 RELATED WORK

With data stored in distributed data centers, high availability is achieved by a replica of data, and low latency is achieved by placing those data users need closer to the users[8]. It is notable to study distributed transaction processing due to the popularity of geo-distributed storage systems. In distributed system, Atomic Commit Protocols (ACP) are

used to guarantee protocol resilient to communication and site failures, and if all failures repaired, then transactions commits or aborts at all sites. Common metrics to compare different protocols are number of I/O, messages needed to passed over the network, latency, blocking, and recovery complexity. The most common used ACP is Two Phase Commit (2PC) [4], which has two roles in this protocol, including a coordinator and participants. The coordinator send prepare transaction execution request to participants and wait for their replies. If any one of the participants respond abort, then also the transaction; Otherwise the coordinator will send commit message to participants to let them commit locally. The drawback of 2PL is that once the coordinator crashes after sending prepare message to the participants, the whole system is blocked until the recovery of coordinator. Then, three phase commit (3PL) [2] was proposed such that divided the prepare phase of 2PL into two phase, and the new first phase won't execution the transaction until the second phase collecting all the ready responses from participants to save the resources whenever there is an abort. Also mechanism such as election of the new coordinator after the original coordinator is also introduced.

Spanner is a distributed system providing order-preserving serializable transactions while it is slow regarding to latency since it takes too many cross-datacenter roundtrips while executing the transaction, so does Replicated Commit [7] or MDCC [5]. Lynx[8] proposes to chop transaction into transaction chains and response to user right after the first hop of transaction chain commit to reduce the latency. But Lynx was not designed to handle huge amount of transactions in practice. In recent years, caching is proposed to execute transaction more efficient by cutting down remote reads and batching is introduced to amortize the communication cost during committing [1][6]. Moreover, Hackwrench [3] is proposed to repair the conflicted transactions, instead of expensive aborting or execute an entire bath of transactions, regarding to the conflict problem introduced by caching and batching.

4 APPLICATION DESIGN

Our application is a airline ticket booking system that fits the needs of the modern-day aviation industry. Nowadays, most airlines still operate their booking system under command line software. Not only is it outdated, it also lacks extensibility for other airlines with codeshare agreements to utilize their booking system. These factors could lead to

many different vulnerabilities in a real-world setting, since different airlines are allowed to sell tickets on flights with codeshare agreements but not actually operating. As a result, our project tries to improve the overall design and approach of it. In this application, users can book ticket from this system, airline companies can establish or cancel flights and notify users, airports can maintain their employee (e.g., ground staff at the airport) status. Besides, airline companies can add/remove aircrafts crew to/from flights. Hence, users are also able to see the flight passager number and aircraft crew number so as to choose a flight that is relatively spacious or one that is more likely to have guaranteed service quality. Moreover, user can update their profiles such as editing their user type as "student", "children", "disabled/pregnant", or "others". Ticket price will also be different for those different type of users and differentiated services are provided for them. Differentiated services are also implemented into this system with class of the seats, such as Business Class, First Class, Second Class, and Economic Class.

4.1 Table Schemas

As shown in Figure 9, the database of this application has 8 core tables, which are Ticket, Flight, Airport, User, Aircraft, Airline, Aircraft Crew, and Ground Staff. The table names and attributes should be self-explanatory.

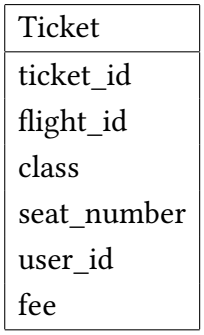

Fig. 1. Ticket Table

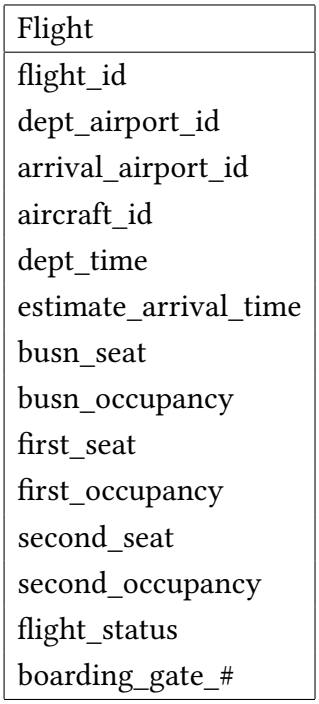

Fig. 2. Flight Table

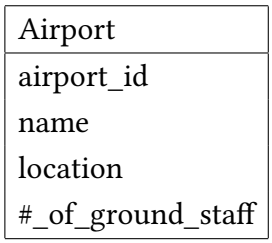

Fig. 3. Airport Table

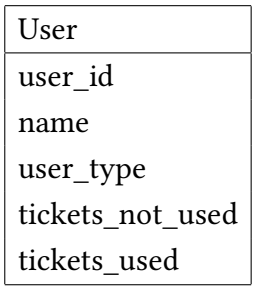

Fig. 4. User Table

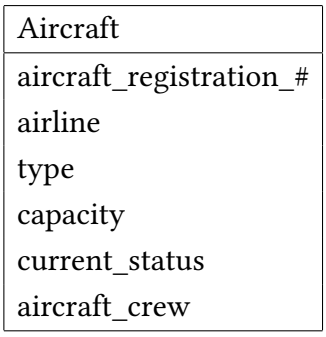

Fig. 5. Aircraft Table

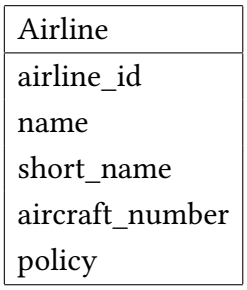

Fig. 6. Airline Table

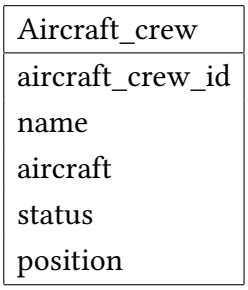

Fig. 7. Aircraft Table

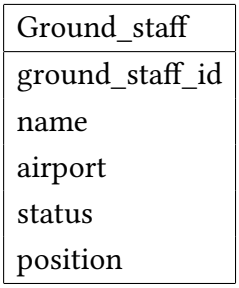

Fig. 8. Ground Staff Table

Fig. 9. Core Tables in The Application

4.2 Data Partition

Due to the geo-distributed nature of users, culture, regional security laws, and etc., many applications have globally distributed data centers storing data for users of different regions/countries, such as TikTok, AWS. For our application, we decided to partition our

data into different regions, which each region contains a central database that handles all transactions from countries in that region. For instance, the central database in the US will serve as the regional center for users in all countries of North America. The rationale behind this partition is to reduce latency between users and servers.

5 TRANSACTIONS ON THE SYSTEM

5.1 Remove SC-Cycle for Six Transactions

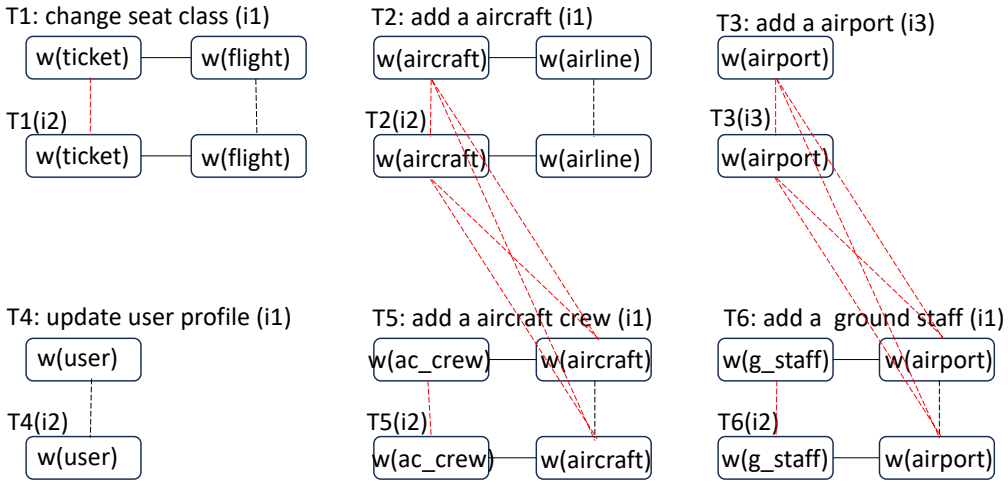

Fig. 10. Example of removing SC-circle from SC-graph

5.2 Transactions with irremovable SC-Cycle

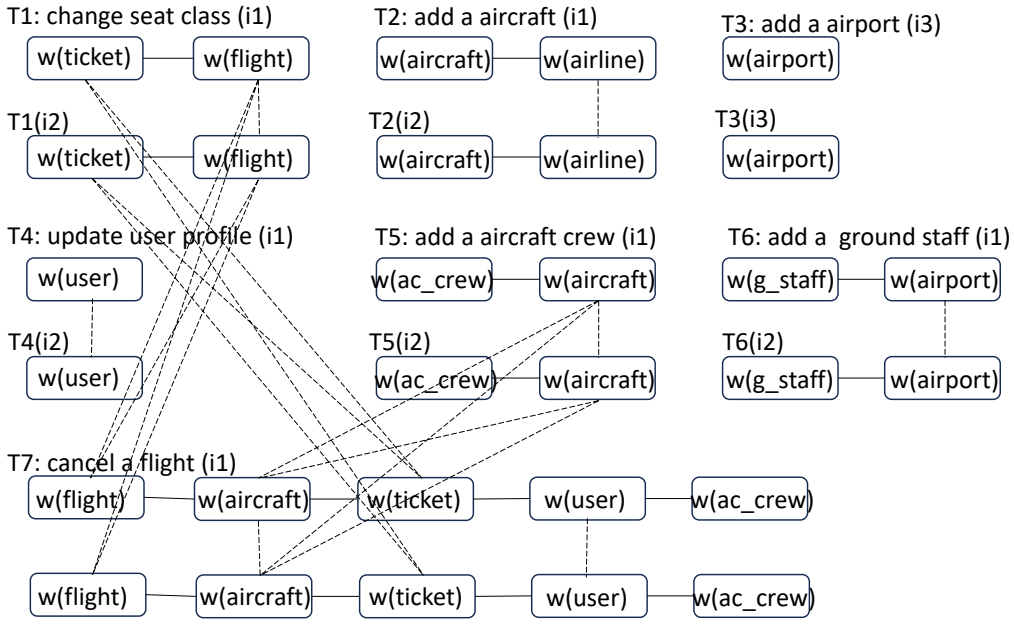

Fig. 11. Example of SC-graph with unremovable SC-circles

6 IMPLEMENTATION

We implemented our transaction system using Python. We selected Python as our programming language for implementation because it has a wide range of libraries, thus making our simulation and testing process much easier. Our codes are centered around three important Python libraries: `socket`, `threading`, and `sqlite3`. The `socket` library is a built-in, low-level networking interface in Python that provides a direct access to network simulation, so that we can simulate sockets, hosts, ports, and IP addresses. Within the library, the `recv` and `send` functions serve as the "read" and "write" operations just like the ones we learned in class. The `threading` is another built-in Python library that focuses on thread-based parallelism. On the other hand, the `sqlite3` is a library that was originally written C, but integrated into Python for usage of disk-based databases. It provides essential database operations to the SQL database engine. Specifically, we used the `connect` function to open a connection to a database, and the `cursor` function to

traverse through the records in the database. On the execution part, instead of keeping track of a list of port numbers and database file names, we defined them with the node ID as command line arguments. By doing so, we allow the users to decide which port, file, or node they would like to run operations in.

7 EXPERIMENT

7.1 Experiment Setting

Our experimental setup was designed to assess the capabilities of our transaction system in a controlled environment. The foundation of our data was done and tested on a MacBook Pro 2023 which has been developed with the recent implementation of the M2 chip known for its computational efficiency and great performance. This provided us with a stable platform to be able to simulate the transactions and processing on our database with various sizes.

The datasets that were used in our experiments composed of 1000 (1k), 5000 (5k), and 10000 (10k) instances, which represented a spectrum of database sizes which are commonly seen in other applications. We were able to leverage the processing power of Python to generate a lot of our data using various scripts which were made to reflect the structure and how complex the airline ticketing system was.

To actually create accurate performance metrics of the system, we utilized various Python scripts and packages mentioned earlier in the report to make precise timing functions to measure different metrics. The latency which is the time taken to complete a single transaction and also the throughput which is defined as the number of successful transactions processed per second. Our methodology was quite tedious as it was made sure to be measured under uniform conditions like network conditions and system loads to make sure our results were reliable and unbiased.

Latency was measured as the time elapsed from the exact moment that the transaction request was sent from our client program over to the server until the acknowledgement of the completion. This was done by wrapping each call within a timing function of sorts the

recorded start and end times which gave us the total duration to create an averaged latency.

Throughput was measured as the number of transactions that were successfully processed per second. To calculate this, we divided the total number of transactions by the total time that the system took to process all of those transactions with a single session. The data we got from this was pretty valuable as it gives a good idea of the systems efficiency.

To add, these tests were done on a macOS with the latest updates of SQLite version 3.39.5 which is a widely used data management system for simplicity's sake.

7.2 Effect of Data Size

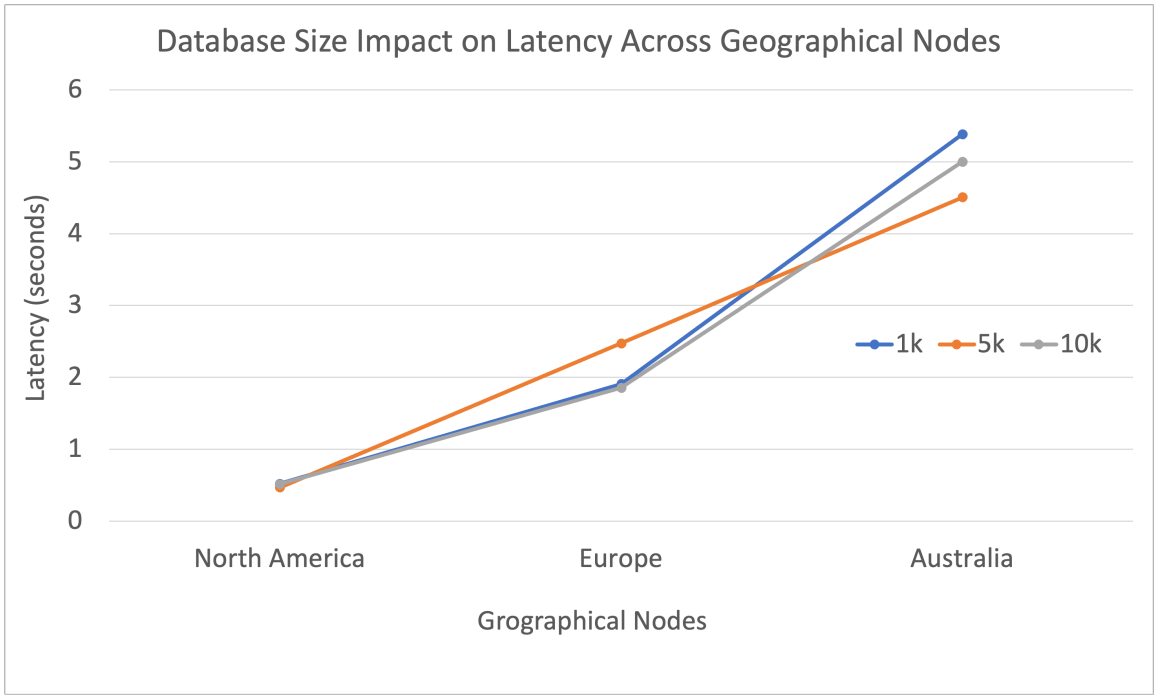

Fig. 12. Graph displaying the impact of different data base sizes on latency across different geographical nodes

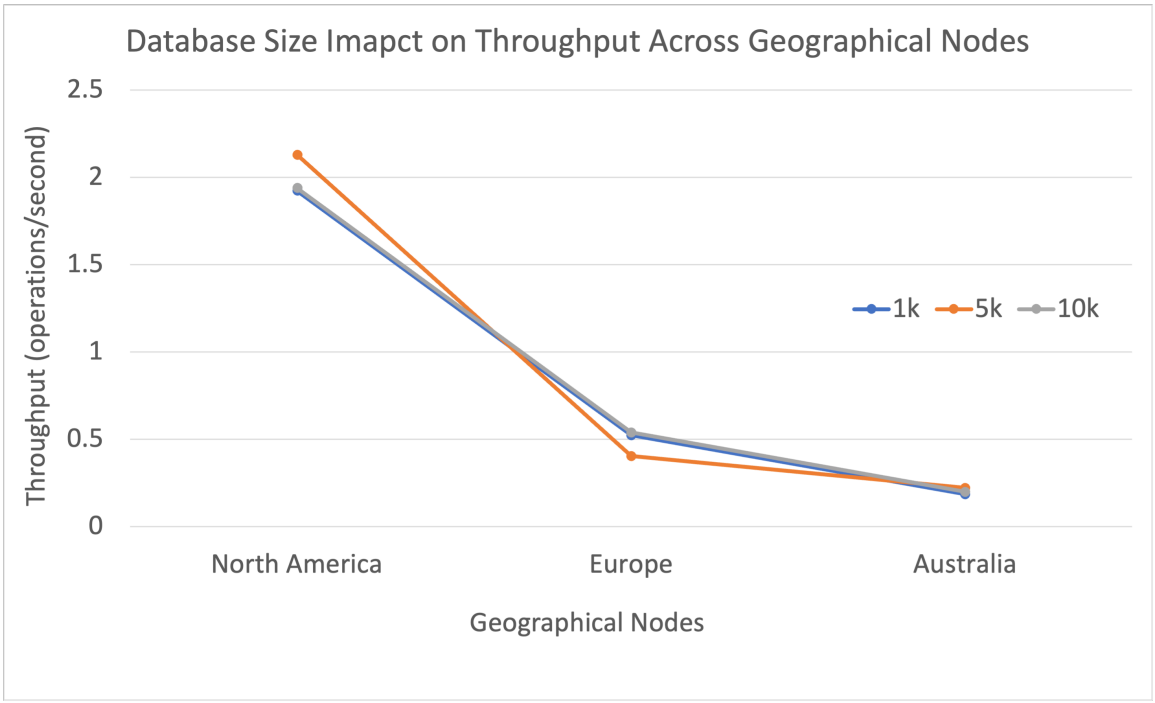

Fig. 13. Graph displaying the impact of different data base sizes on throughput across different geographical nodes

As we can see here in the graphs above, it illustrates the database size impact on the latency and throughput across various geographical nodes. The stable latency suggests that the simplicity of the transactions, which would include operations like simple inserts and update operations, did not exactly require complex querying or more heavy computations. In our simulation where the overhead for these computations are quite low mostly has to do with the case of utilizing SQLite. SQLite has quite a lightweight architecture so its system along with the lack of network overhead likely is contributing to what we can see in the data and how minimal of an impact the database size has on latency. The throughput can also be explained using the same logic as it is quite synonymous to the latency graph. Additionally, the operations within our transactions written in our client program did not rely a lot on indexing or join operations which would have potentially changed our results in terms of processing time.

7.3 Effect of Concurrent Request Number

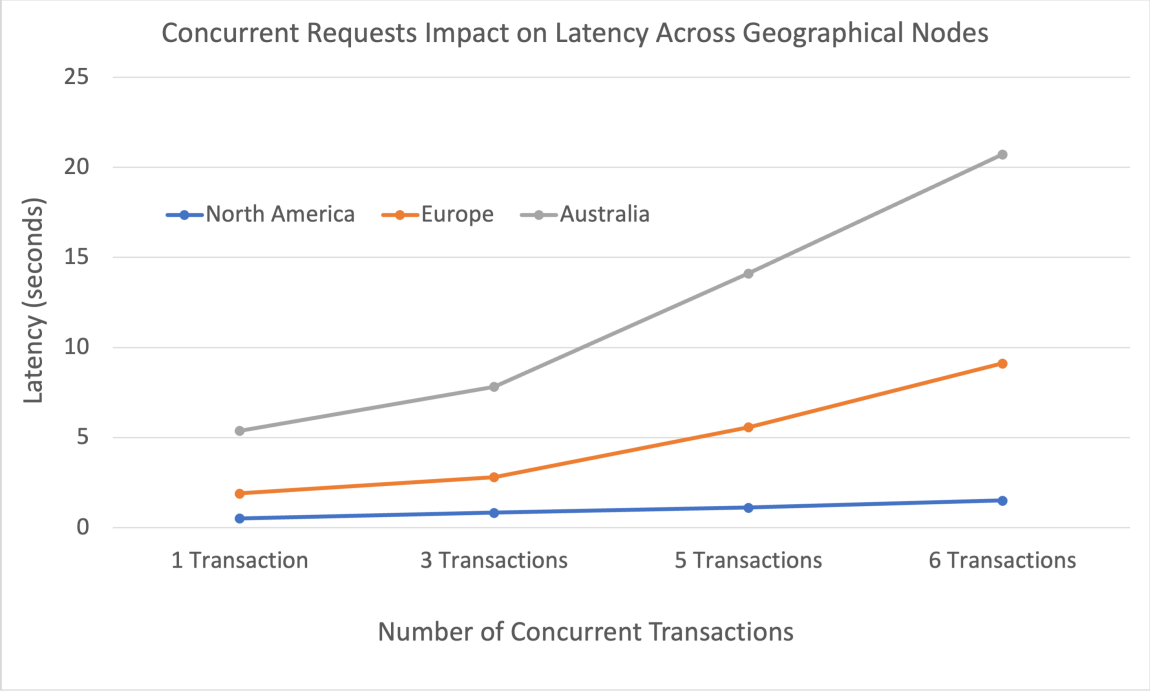

Fig. 14. Graph displaying the impact of different number of concurrent requests on latency across different geographical nodes

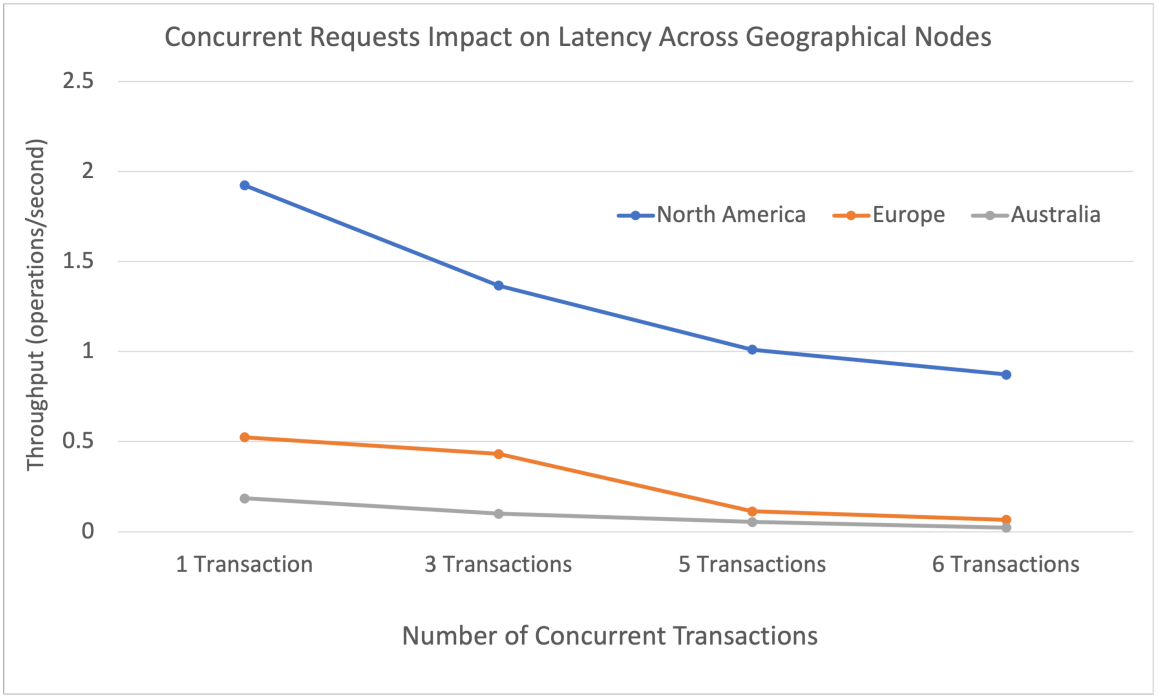

Fig. 15. Graph displaying the impact of different number of concurrent requests on throughput across different geographical nodes

The graphs above showcase the different relationships between the number of concurrent transactions and their impact on the latency and throughput of the system across the three different geographical nodes. As the number of concurrent transactions increases, the latency can clearly be seen rising and quite steeply for that matter in regions farther from North America. For example, for Australia, the trend follows an exponential pattern as we go up the number scale for concurrent requests which seems about right considering the stipulations within our transaction system. Conversely, we can see that the throughput is decreasing at somewhat of a constant rate for Europe and Australia and largely for North America which is sound. The efficiency in the processing power of each operation within the transactions appears to go down within a specific time frame suggesting to us that the capacity to handle a certain amount of concurrent requests has a threshold to which after, the performance degrades.

8 CONCLUSION

Our application is an introduction to a new way of thinking about the significance of databases and concurrency in the aviation industry. Our experiment contains crucial data, technologies, and simulation that are necessary for future extension and usage in real-world scenarios. Throughout the time we spent on this project, we acquired a more thorough and deeper understanding about the impact of transactions and databases.

REFERENCES

[1] Dennis Shasha, Francois Llirbat, Eric Simon, and Patrick Valduriez. Transaction chopping: algorithms and performance studies. *ACM Trans. Database Syst.*, 20(3):325–363, 1995.

[2] Yang Zhang, Russell Power, Siyuan Zhou, Yair Sovran, Marcos K. Aguilera, and Jinyang Li. Transaction chains: achieving serializability with low latency in geo-distributed storage systems. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 276–291, New York, NY, USA, 2013. ACM.

[3] Charlie Custer. Distributed transactions: What, why, and how to build a distributed transactional application. 2023.

[4] Keyang Xiang. Patterns for distributed transactions within a microservices architecture. 2023.

[5] Dennis Shasha, Eric Simon, and Patrick Valduriez. Simple rational guidance for chopping up transactions. In *Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, SIGMOD '92, pages 298–307, New York, NY, USA, 1992. ACM.

[6] Ning Huang, Lihui Wu, Weigang Wu, and Sajal K. Das. DTC: A Dynamic Transaction Chopping Technique for Geo-Replicated Storage Services. 2022.

[7] Python Software Foundation. socket — Low-level networking interface 2023.

[8] Python Software Foundation. threading — Thread-based parallelism 2023.

[9] Python Software Foundation. sqlite3 — DB-API 2.0 interface for SQLite databases 2023.

[10] GIGAZINE. 'Command line' is still flying behind the scenes for online booking of airplane tickets 2018.

REFERENCES

[1] Marcos K Aguilera, Arif Merchant, Mehul Shah, Alistair Veitch, and Christos Karamanolis. Sinfonia: a new paradigm for building scalable distributed systems. *ACM SIGOPS Operating Systems Review*, 41(6):159–174, 2007.

[2] Muhammad Atif. Analysis and verification of two-phase commit & three-phase commit protocols. In *2009 International Conference on Emerging Technologies*, pages 326–331. IEEE, 2009.

[3] Zhiyuan Dong, Zhaoguo Wang, Xiaodong Zhang, Xian Xu, Changgeng Zhao, Haibo Chen, Aurojit Panda, and Jinyang Li. Fine-grained re-execution for efficient batched commit of distributed transactions. *Proceedings of the VLDB Endowment*, 16(8):1930–1943, 2023.

[4] Jim Gray, Raymond A. Lorie, Gianfranco R. Putzolu, and Irving L. Traiger. Distributed transactions: The fault-tolerant approach. *ACM Trans. Database Syst.*, 6(3):223–247, 1981.

[5] Tim Kraska, Gene Pang, Michael J Franklin, Samuel Madden, and Alan Fekete. Mdcc: Multi-data center consistency. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 113–126, 2013.

[6] Yi Lu, Xiangyao Yu, Lei Cao, and Samuel Madden. Epoch-based commit and replication in distributed oltp databases. 2021.

[7] Hatem Mahmoud, Faisal Nawab, Alexander Pucher, Divyakant Agrawal, and Amr El Abbadi. Low-latency multi-datacenter databases using replicated commit. *Proceedings of the VLDB Endowment*, 6(9):661–672, 2013.

[8] Yang Zhang, Russell Power, Siyuan Zhou, Yair Sovran, Marcos K Aguilera, and Jinyang Li. Transaction chains: achieving serializability with low latency in geo-distributed storage systems. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 276–291, 2013.