



TCSS 558: APPLIED DISTRIBUTED COMPUTING

**DE-OPTIMIZING DISTRIBUTED DATABASES
FOR A MILITARY USE CASE**

Student:

David P. Coughran Jr.
Sapper46@uw.edu

Committee Chair:

Prof. Dongfang Zhao

Committee Member:

Nicole Guobadia

School of Engineering and Technology

March 18, 2025

Abstract

Research in distributed databases has focused on optimizing the performance of reads, writes, latency, isolation levels, and throughput. Over time we have seen successful advances that push the boundaries of impossibility results while delivering better overall system performance. However, in this field, researchers have mostly explored use-cases stemming from finance, social media, ecommerce, and banking. There are fewer publications which consider military operations and the unique requirements for distributed databases in this type of context. It follows that optimizing distributed military databases, along the lines of the research supporting non-military use-cases, may in fact lead to reduced system performance for military operations.

In this paper, we present three concepts. First, we will present a use-case for testing the performance of a distributed military database. Empirically, we will show through multiple wargame simulations that higher isolation levels can lead to suboptimal outcomes. Our results will demonstrate isolation levels above READ COMMITTED are associated with a 39-121 percent increase in tactical latency, measured as the average time difference between when a reconnaissance sensor creates data and when a frontline tactical unit accesses that data. Higher isolation levels are also associated with longer times to destroy targets and more targets surviving to the end of the simulation. Lastly, our performance metrics will offer new terms and variables to consider for future testing of distributed military databases.

Contents

1	Introduction	4
2	Defining a Military Use-Case	5
3	Use-Case Implementation	8
4	Evaluation	13
5	Conclusions	15
6	Education Statement	16

1 Introduction

Distributed databases are a mature technology which have substantially impacted our everyday lives. Anyone who has sent a “status update” from their phones, checked their bank account balance, or bought a new pair of shoes from a website has indirectly touched a distributed database even though the system is largely transparent. Because we derive so much utility from distributed databases, there is a strong incentive for researchers to continue developing new algorithms to better read, write, and organize data on this type of system.

Take, for example, three papers which have been published in the last four years. Each marks a progressive step forward in terms of pushing the boundaries of impossibility results, improving performance, and offering higher level of privacy to clients. In 2021, Konwar, Lloyd, Lu, and Lynch built out the framework for the SNOW theorem of optimized reads. They successfully demonstrate an algorithm, incorporating SNOW, that provides the strongest guarantees across the spectrum of isolation levels with bounded latency.[3] Liu, Multazzu, Wei, and Basin pick up this research three years later. Their innovation was to apply a similar methodology, but this time, to optimize writes. Their contribution was to develop an improved algorithm, NOC2, that manages better latency and throughput with isolation levels that exceed the industry standard of READ COMMITTED.[4] Lastly, in 2025, Eloul, Satsangi, Zhu, Amer, Papadopoulos, and Pistoia, with backing from JP Morgan Chase, further evolved distributed ledger technology in distributed databases. They examine optimized blockchain transactions that maintain privacy between the various clients while keeping isolation levels above READ COMMITTED.[1]

These examples constitute three very powerful datapoints in the trajectory of research. In all these papers, the researchers describe the benefits their methodologies could bring to modern day use-cases. This includes banking, social media, ecommerce, and finance. In fact, the researchers use real world examples from these industries to describe how their methodologies work and explain the process by which their algorithms execute the transactions.

Despite this success, we must keep in mind that state-of-the-art for one use-case, such as banking or social media, is not necessarily state-of-the-art for another. The goal of this paper is to consider distributed databases for military operations, a use-case which has received demonstrably less research attention than its peers. Our goal is to offer new ways to examine military

system performance, mission outcomes, and other prominent features for a distributed military database. This is important to ensure researchers develop better methods that also benefit less prominent, but still incredibly important, industries.

This paper focuses specifically on isolation levels as our variable of interest. This is a deliberate choice as the ‘drumbeat’ of research in distributed databases seems to be achieving higher and higher isolation levels while offering similar performance in terms of latency, throughput, and reliability. This is clear in the massive proliferation of acronyms around algorithms or impossibility guarantees that deliver higher isolation levels: LORA, RAMP, NOC, NOC2, Eiger, Eiger-PORT, et al. However, our research will demonstrate that, in a simple military use-case, higher isolation levels can inhibit performance. Stated differently, applying a higher isolation level, which remains the goal for so many researchers, can lead to a suboptimal outcome.

The findings in this paper will show, empirically, the tradeoff between system performance in a distributed military database and the application of higher isolation levels. There are distinct research benefits from such an approach. First, we offer quantifiable performance metrics to engineers designing databases for military customers. Second, we present a framework for testing and evaluating a distributed military database. Lastly, this paper introduces new terms, concepts, and variables that offer a roadmap for future research in military use-cases.

2 Defining a Military Use-Case

“Military operations” are a broad professional domain with its own set of experts and practitioners. There are processes for strategy, planning, intelligence collection, orders production, logistics, and communication. Therefore, codifying a “military use-case” is an imprecise undertaking. Because no single use-case can encompass all these tasks, any effort will invariably omit some dimension of a true “military operation.” However, to achieve a basic system we can reliably test, our military use-case focuses on intelligence collection and targeting.

We build our military use-case around an operation where a commander is responsible for surveillance and destroying enemy targets within three assigned areas. To accomplish this task, the commander has sensors and actuators. Sensors perform a reconnaissance role. They monitor a geographic

sector, observe some facet of the physical state (i.e. terrain, electromagnetic signals, weather, enemy troop movements, etc), report this information to a higher headquarters, and nominate targets for destruction. Actuators are the consumers of this gathered intelligence. They rely on reports from sensors to understand the battlefield and, with proper authorization, destroy identified targets. Sensors and actuators operate at the tactical level of the use-case. As a helpful visualization, you can imagine our sensors and actuators as satellites and fighter jets respectively.

At the operational level of our use-case, three forward edge databases compile all the information the sensors gather. Each of the three areas assigned to the commander possesses its own edge database. Only tactical units (i.e. sensors and actuators) assigned to that area can report directly to and update that database. However, to successfully build a view of the entire battlefield, a database manager (one per area) acts as a gateway between the different areas. The database manager reads information from the other databases and uses this to update his tables. This is how the system distributes intelligence between the various areas, while limiting damage should one of the databases become compromised. In total, there are four tables in each database:

- (1) Report Table (compiles the raw information the sensors report),
- (2) View Table (stores tactical metadata from each sensor report),
- (3) Target Table (tracks the total number of identified targets in each sector),
- (4) Permissions Table (stores the number of targets in each sector where the commander has reviewed the intelligence and approved the target for destruction).

At the strategic level, the overall mission commander sits atop the hierarchy of units and data. The commander possesses a strategic level database that aggregates all the information at the operational level, thereby giving the commander the most-up-to-date information across the battlefield. The commander's role is to observe for new targets in the Target Table. After reviewing all the relevant information in the View and Target Tables, the commander gives her authorization to destroy a target. She achieves this by incrementing the number of permissions in the pertinent sector and area. This information populates downward to the operational-level databases, where the actuator can observe the committed update. Again, a database manager acts as the gateway to the strategic level database as a safeguard against system compromise or corruption.

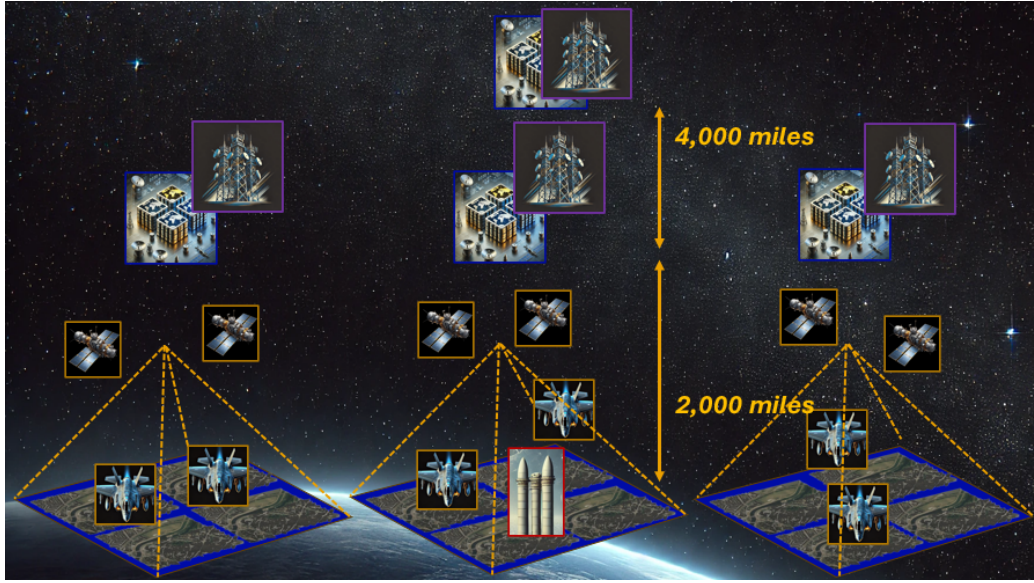


Figure 1: The Military Use-Case

We also consider the spatial dimensions of a military use-case. We apply a conservative estimate that sensors and actuators operate at a distance no less than 2,000 miles from the operational level database they update. Furthermore, we assume the strategic level database is another 4,000 miles away. This implies 6,000 miles between frontline tactical units and the strategic headquarters which commands them. These distances adhere to planning considerations within the U.S. military doctrine of “force projection,” as presented in Department of Defense Publication 3-0: Joint Operations.[2] Figure 1 is a physical rendering to help visualize the sensors, actuators, databases, gateways, and targets.

Now that we have established a specific military use-case for a distributed database, the next step is to build metrics around its performance. To do this, the primary student researcher of this paper interviewed Brigadier General (BG) Paul Sellars, Commander, Washington Army National Guard. BG Sellars interpreted this use-case as representative of a typical mission set belonging to a Division-level Joint Task Force (JTF). This is a level of military echelon he has experience commanding. We asked him, “Given the database system and the operational requirements we have just described, please tell us your overall priorities and how you would evaluate this distributed database’s

performance.” BG responded unequivocally. As the commander, his three metrics would be:[6]

(1) Timeliness of Intelligence Reports (Tactical Latency)- the time in seconds measured between when a sensor makes an observation (i.e. renders an intelligence report) and when this data becomes available to a frontline unit. For future reference, we introduce the term ”tactical latency” to describe this measurement and differentiate it from overall system latency.

(2) Time to Target Destruction (T2D)- the time, measured in seconds, from when a sensor first identifies a target, through the commander’s approval for destruction, to when the actuator receives authorization and destroys the target. We will use the term T2D as shorthand for this metric.

(3) Number of Missed Targets (NMT)- If, in experimentation or testing of system design, BG Sellars would want to know if and how many targets the system allowed to escape. He interprets more missed targets as degraded system performance.

3 Use-Case Implementation

With a clearly defined use-case, the next step is to build out the technical implementation. For databases, we use Postgresql ver 17.4 with each database possessing the four tables described in the previous section. The database at the strategic level serves the role of the main server, while we consider the operational level databases as forward edge servers. The database managers/gateways are python scripts that are responsible for their specific area. We name our areas A, B, and C. Each area has two sectors, Sector 1 and Sector 2. Therefore we have six total sectors, four total databases, and four total database managers.

Our sensors and actuators are also python scripts. We assign sensors to an area, and they share responsibility for surveilling each sector in that area. We also assign two actuators to each area, but unlike the sensors, each actuator is responsible for a single sector. This is in keeping with our earlier visualization that satellites are wider ranging than aircraft which operate within smaller geographical areas. There are six total sensors and six total actuators.

To create the distances the data must travel, we deploy three Amazon AWS T2.Micro Instances. Each instance possesses 1 virtual CPU consisting of a 2.5 GHz Intel Xeon Family processor, 1 GiB of memory, and 30 GiB

of Elastic Block Store (EBS) storage. Our sensor and actuator scripts each run inside their own individual Docker container on the tactical T2 instance. Our tactical instance is virtualized on a server farm in Ohio, United States (Amazon Region US-East-2). The second instance represents our operational level. This instance runs on a server farm in Oregon, United States (Amazon Region US-West-2). The three operational level databases and the database managers each run inside of their own Docker containers. Lastly, at the strategic level, we virtualize our instance in London, United Kingdom (Amazon Region EU-West-2). Once again our strategic database and our strategic database manager each run in their own individual Docker container.

The system of containers and python scripts codifies the reporting rules inherent to our military use-case. Sensors can only populate columns in the Report, View, and Target Tables that correspond to their areas. For example, a Sensor in Area B can only update the B Column in the Report, View, and Target Table of the Area B database. The Area B database manager updates the remaining columns by reading from the other databases (A and C). (Note: It is helpful to think of database columns A, B, and C as corresponding to Areas and database rows (1 and 2) corresponding to Sectors within that Area. For shorthand, “B2” refers to Area B Sector 2.) Each Area has its share of the overall intelligence picture and database managers pull from other areas to ensure it has the complete view of the battlefield. The database managers at the operational level repeatedly update information from the other two databases, agnostic to any of the data it might possess. We offer Figure 2 as a visualization of the interaction of reads and writes on database transactions.

At the strategic level, the database manager is essentially reading from each of the operational level databases and using that data to construct the main server. It reads the A column from all the tables in Area A, the B Column from all the tables in Area B, and the C column from all the tables in Area C. This follows the idea that, whenever possible, we want to read data and intelligence with minimal separation from the device that created it. Additionally, the strategic database manager updates the permission tables when it sees a new target, i.e. a row and column in a target table that is greater than the corresponding row and column in the permission table. As a system constraint, the strategic database manager can only increment the Permission Table by 1 in a single transaction. This creates a formal transaction cost to simulate the idea that the commander and her staff would need time to process the target intelligence before authorizing its destruction. Therefore, they must focus on one target at a time.

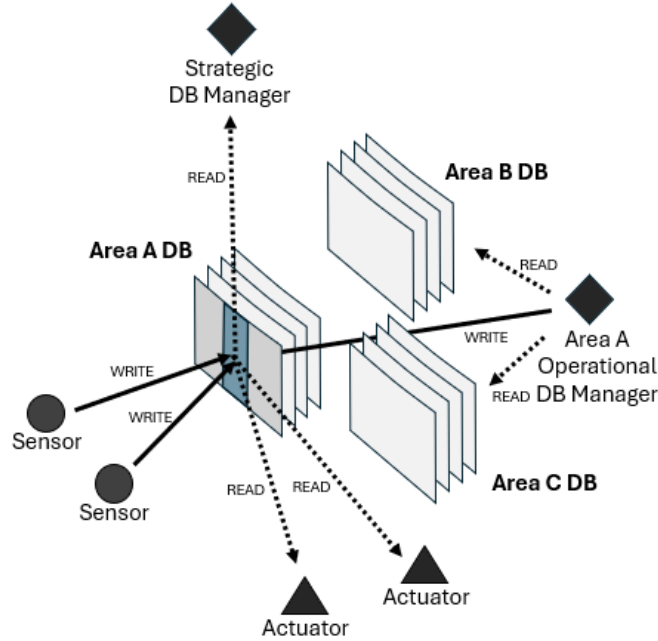


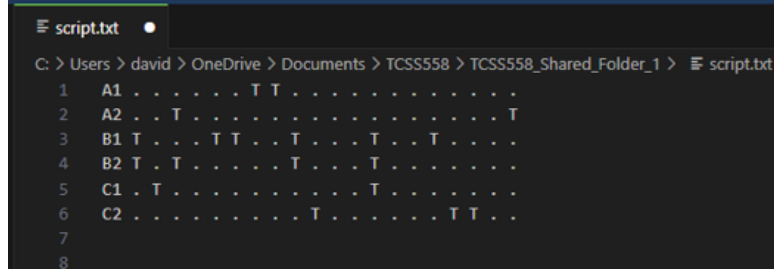
Figure 2: Data Distribution

As a python script, sensors perform two tasks. First, they check one of the two sectors in their area for targets. The script randomizes the specific sector they surveil with an equal 50 percent probability. Specifically, the sensor reads a text file that tells them how many total targets have presented themselves in that sector thus far into the simulation. Each text file is named for its area and sector, i.e. A1.txt, A2.txt, B1.txt, etc. If a sensor “sees” a new target in this text file, it updates the Target Table within its assigned database. The second task sensors perform is updating the Report and View Tables. We simulate an intelligence report from a sensor as a 2 MB transmission of plaintext from the sensor to the respective column and row of that sector. The View Table represents the tactical metadata of this report. The View Table gives a timestamp (in seconds since the Epoch) of when the Report Table was last updated. In total, the sensors’ behavior is a repeating loop: Check for new targets, report new intelligence, and then repeat.

As script, actuators perform a role both in the wargame simulation and the implementation itself. This was necessary to preserve memory because the T2 instance could only support so many containers running individual

$$\text{Tactical Latency} = \frac{\sum_{i=1}^{\text{num of sectors}} \text{current time} - \text{time stamp of tactical metadata}_i}{\text{number of sectors}}$$

Figure 3: Calculating Tactical Latency



```
script.txt
C: > Users > david > OneDrive > Documents > TCSS558 > TCSS558_Shared_Folder_1 > script.txt
1  A1 . . . . . T T . . . . .
2  A2 . . T . . . . . . . . . . T
3  B1 T . . . T T . . . . T . . .
4  B2 T . T . . . . . T . . . . .
5  C1 . T . . . . . . . . T . . . .
6  C2 . . . . . . . T . . . . T T .
7
8
```

Figure 4: Script for 20 Percent Target Density

scripts. In the simulation, actuators act as the ultimate consumers of the intelligence collected by the sensors. They pull tactical metadata from the View Table of their respective database (i.e. Actuator A2 only pulls from the A database) to gain situational awareness across the entire battlefield. This includes all six sectors across the three areas. Each time the actuator pulls this data, it will measure the tactical latency of the information by comparing it to the current time. This allows the actuator to calculate on average how stale the intelligence is, per sector across the six sectors (See Figure 3).

In terms of targeting, actuators also play an implementation role to introduce new targets to the sector. They accomplish this by reading from a text file named “script.txt.” (Note, this is a script in the sense of what happens in a theater performance and not a script in terms of a programming executable.) In each script, there is a row corresponding to each sector. Actuators follow the script to know when a new target has entered the sector. Consider the script in Figure 4 as an example.

The actuator in A2 understands that in the simulation, two total targets will enter its sector. Each column represents one time interval that we set in the simulation’s initialization. So, in this specific example, if each interval is set to 15 seconds, a new target enters sector A2 at the 30 second mark and the 4 minute 45 second mark. The actuator adds the additional target by updating the text file associated with its sector. In this example it means that at the 30 second mark, Actuator A2 updates the text file A2.txt from

“0” to “1” indicating one total target in that sector. This makes the target “observable” to sensors. At the 4 minutes and 45 second mark, Actuator A2 updates the text file A2.txt from “1” to “2.” Actuators record exactly when they have updated the text file for each new target.

In addition to introducing the new targets, actuators pull data from the Permissions table to determine when the commander has authorized a target for destruction. In this manner, because they know when the target was introduced to the sector, and exactly when they read the destruction authorization from the Permission Table, Actuators can effectively measure how long it took to destroy a target. To summarize, actuators perform two tasks in a constant loop. First, they update their “View” of the battlefield by pulling tactical metadata from their database. Second, they check the Permissions Table of their database to see if any new targets are authorized for destruction.

In our experimentation, we generated three scripts. The first script represents a 20 percent probability for each interval that a new target will enter a sector. So, in Figure 4 and reading across for sector A2, at each interval there was a 20 percent likelihood of a target (“T”) and an 80 percent probability for no target (“.”) in each column. We randomly generated scripts at 20, 30 and 40 percent probability levels to increase the load on our simulation. However, once the script is built for that probability level, we reuse it in experimentation for consistency across the different isolation levels.

At the end of each simulation, the actuators generate a simulation report to provide data for our analysis. They calculate, on average and across the entire simulation, the tactical latency of the intelligence reports when they pull metadata from the View Table. They also calculate the average of how much time elapsed between a target entering the sector and when they observe authorization to destroy it in the Permissions Table. Lastly, our actuators report how many targets survived the simulation. So if four targets enter sector B2, but Actuator B2 only observes authorization to destroy two, this means that two targets have survived. At the end of the simulation, a short script compiles and averages these readouts across all the sectors.

One important note here is that our experimentation focuses only on the speed by which data travels in the simulation. In other words, in real life, there are sure to be human and other transactional costs in how the data is processed. The commander may need time to deliberate over an intelligence report with his staff, or a pilot may need time to physically fly somewhere after he receives an order to destroy a target. Regardless, we

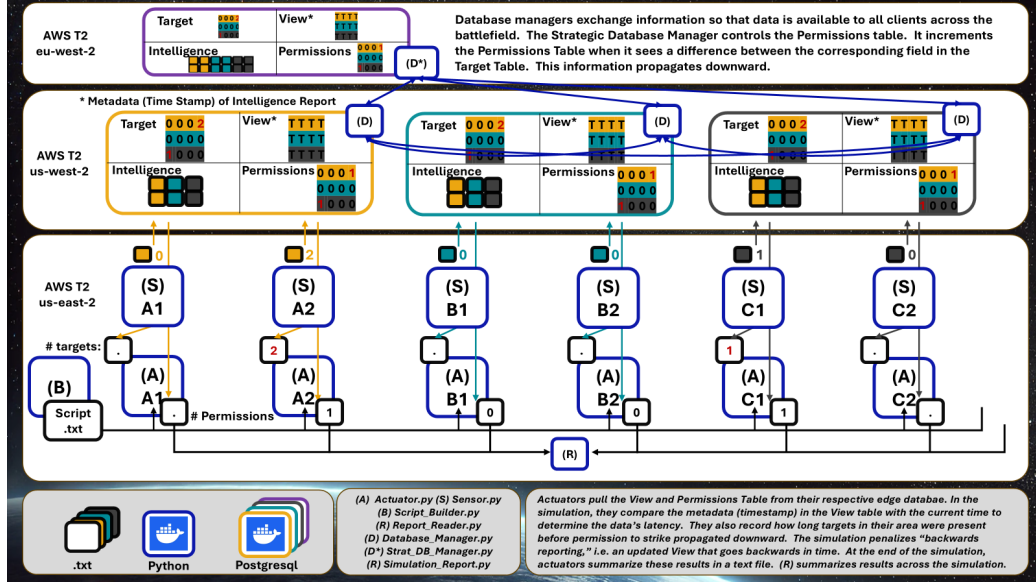


Figure 5: Implementation Data Flow

ignore these extra costs in terms of time and latency by assuming all such factors to be zero. This allows us to effectively measure the speed of data transmission specific only to the use case's technical implementation. Figure 5 is a representation of data flow in the overall implementation.

4 Evaluation

In our experimentation, we repeatedly ran a 300 second simulation (20 total 15 second intervals) while varying the isolation level of database transactions and target density. The results presented in Figure 6 represent the best performing simulation at a given isolation level and target density. In other words, we report the simulation that yielded the best (least) tactical latency, T2D, and NMT. It is necessary to present the data in this manner because of the variability of network conditions across the three different AWS instances. During simulations, we had to contend with periods of high network traffic, system maintenance, and breakdowns in our ssh connections. The results could change based on the time of day we carried out the simulation or if the network was carrying a heavy load. However, it became clear that each simulation had an optimal result at a given isolation level and target density.

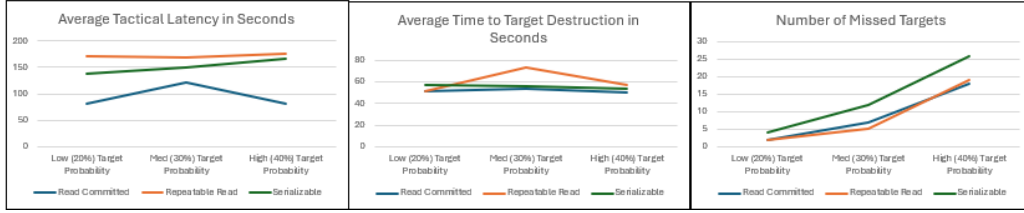


Figure 6: Results

This meant, no matter how many times we ran the simulation, the best results for those simulation conditions were repeatable within 1 or 2 seconds of each other. We could reliably identify the optimal result in four or five simulation runs taking place between the hours of 3 PM and 6PM Pacific Standard Time.

The results confirmed our initial hypothesis: higher isolation levels lead to reduced system performance in terms of tactical latency, T2D, and NMT. Moving to a higher isolation level beyond READ COMMITTED can lead to at most a 121 percent increase in tactical latency at high target density, and at least a 39 percent increase at medium target density. Higher isolation levels are also generally associated with more missed targets and a longer time to target destruction.

These results make sense based on how Postgresql manages transactions. Consider an actuator that is reading tactical metadata from its View Table. After the actuator begins the transaction, READ COMMITTED allows updates to the table while the actuator is still reading. At higher isolation levels, the initial snapshot is constant. So while more up-to-date intelligence may be available and indeed committed to the table, at isolation levels above READ COMMITTED the actuator will still only read intelligence from the initial snapshot. This is even more pronounced given the limited computational power we have available to carry out all the transactions, i.e. one thread as per the T2 Instance constraints.[5]

In terms of targeting, the system again performs as expected. At higher target densities, more targets survive to the end of the simulation. Again, this occurs because the database manager at the strategic level can only analyze intelligence and increment the permission tables one target at a time. We also see a marginal increase in NMT at higher isolation levels. We expect this behavior for reasons previously discussed: higher isolation levels do not allow updates to a table snapshot in the middle of a read.

One unexpected result in this experiment is in the “Average Time to Target Destruction” table. We observe that at a 30 percent target density, the REPEATABLE READ isolation level required almost 20 additional seconds to destroy each target. This does not follow our intuition. Targeting is a function of the Target and Permission Tables. Unlike the Report and View Tables, these tables do not carry a lot of data and are the fastest to update. The simulation instructs actuators and sensors to carry out one transaction for intelligence, and another transaction for targeting, meaning system resources are even between the two functions. In Postgresql, writes at isolation levels above READ COMMITTED do incur some additional overhead. This includes checks for commit conflicts, serialization errors, and other anomalies. However, the transactions in this simulation are so simple that none of these problems arise, meaning the extra steps are a small additional expenditure of computational time. So while we might see slightly elevated levels of T2D at higher isolation levels, we expect results being within a few seconds of each other and not as pronounced as the 20 second departure. This suggests that, despite our best efforts, there may have been some additional noise or network traffic in one of our instances when we ran the simulation for REPEATABLE READ at 30 percent target density.

5 Conclusions

In the end, the results presented by this paper should not come as a surprise to anyone. There is little novelty to demonstrating the performance advantages of a READ COMMITTED isolation level. In fact, Postgresql documentation recommends setting transactions to READ COMMITTED if the desired trait is overall speed and efficiency. In this sense, our findings are, at best, a form of incremental research.

However, the true goal of this paper is reminding the research community that what works best in one use-case is not best for all use-cases. Pushing the boundaries of impossibility results for a distributed database in a social media application may not benefit distributed databases intended for military operations. In fact, this paper offers precise empirical data for how costly the tradeoff to a higher isolation level can be in terms of system performance. The data suggests this cost can be quite high, nearly doubling the latency of intelligence reports. This is an important insight for specialists who perform data management for military operations.

Furthermore, this paper also offers a simple, yet effective framework for implementing and testing a military use-case. For researchers with little exposure to military operations, the guidelines in this paper offer simple concepts such as tactical latency, target destruction, and number of missed targets as effective performance metrics. With the important role distributed databases play in our national defense, we must ensure research pushes state-of-the-art in this field as well.

All files and scripts used in the implementation of this experiment are available in our online repository:

<https://github.com/sapper46/TCSS558ResearchProject/tree/main>

6 Education Statement

Dave Coughran is a Master of Computer Science and Systems Candidate at the University of Washington- Tacoma. He holds a Masters of Public Policy from the Harvard Kennedy School of Government and a Bachelors in Mechanical Engineering from the U.S. Military Academy at West Point. Dave is a twenty-two year public service veteran with experience in engineering, special operations, software design, public policy, and diplomacy.

References

- [1] ELOUL, S., SATSANGI, Y., ZHU, Y. W., AMER, O., PAPADOPOULOS, G., AND PISTOIA, M. Private, auditable, and distributed ledger for financial institutes. *arXiv preprint* (2025).
- [2] JOINT CHIEFS OF STAFF. *Joint Publication 3-0: Joint Operations*. Washington, DC, 2023.
- [3] KONWAR, K. M., LLOYD, W., LU, H., AND LYNCH, N. Snow revisited: Understanding when ideal read transactions are possible. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2021), IEEE.
- [4] LIU, S., MULTAZZU, L., WEI, H., AND BASIN, D. A. Noc-noc: Towards performance-optimal distributed transactions. *Proceedings of the ACM Management of Data (SIGMOD)* 2, 1 (2024), Article 9.

- [5] POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL Documentation*, 2025. Version 17.4. Available at: <https://www.postgresql.org/docs/>.
- [6] SELLARS, P. Phone interview with bg paul sellars, commander, washington army national guard, Mar. 2025. Conducted by David P. Coughran Jr., March 1, 2025.