VChainsaw: Parameterizing trust in boolean range queries

Abhinav Hampiholi
hampiholi@gatech.edu

Aniruddha Mysore
animysore@gatech.edu

ABSTRACT

Querying a blockchain is a difficult task for users since in practice, blockchains are too large to query locally on user machines. Therefore, current querying systems allow user nodes to send their queries to a powerful 'service provider' (SP) node which stores the entire blockchain and returns query results to the user. The problem with this system however, is that the user needs to 'trust' the SP. This lead to the development of trustless systems which require the SP to return a 'proof' along with the result set. Current SOTA trustless query systems are very strict in their proof requirements. In this paper we introduce a relaxation that allows the user node to ask the SP for a 'weaker' proof in exchange for lower latency.

1 INTRODUCTION

Current methods of querying a blockchain involve copying all the data in a blockchain (often in the order of hundreds of gigabytes) into a centralized traditional database (called the 'service provider') which services queries sent to it by the 'user nodes'. Currently, there are two broad approaches. The first approach is when the user nodes trust the service provider completely and make no attempt to verify the veracity of the result set. The second approach is when the user does not trust the service provider at all and demands a 'proof' that allows it to independently verify the received result set. Both these approaches have their drawbacks. The first approach renders the results unverifiable and requires the user to trust the central database. The second approach while being more in the spirit of blockchain-based, trustless systems involves large overheads both for the service provider (while generating proofs) and for the user (while using proofs to verify results). When considering the user of a large blockchain analytics application, who does not completely trust the provider but requires low-latency query responses, both these solutions fall short. In this work, we introduce a way to allow the user to relax the strict constraints that current verifiable query systems impose in favor of improving the overheads while generating proofs. We define a new, more customizable constraint on the result set and design a new method of proof generation that allows the user node to verify that the result set indeed satisfies the constraint. The metrics we optimize for are query latency: the time it takes for a user to obtain a verified approximate query result and the size of the 'proof' that is used to verify the result. Our method gives more freedom to users and allows them to 'parameterize' the amount of trust they are willing to grant the service provider. This is especially useful in a setting where the user is more interested in faster response times rather than the 'completeness' of the result set. For example, if an analyst is interested in the average transaction fee of all transactions on chain in the month of November 2022, they may be satisfied with an average accurate to the nearest dollar (not the exact average). In order to compute this (approximate) average they may not be interested in proving to themselves that the 'complete' set of all transactions in November 2022 was returned in the result set. As long as a sizable fraction (say half) of all transactions in November are provably returned, the user can be reasonably confident that the average that they compute on this result set is accurate. Our method gives users the flexibility to make such trade-offs between the 'strength of the proof' and the query latency.

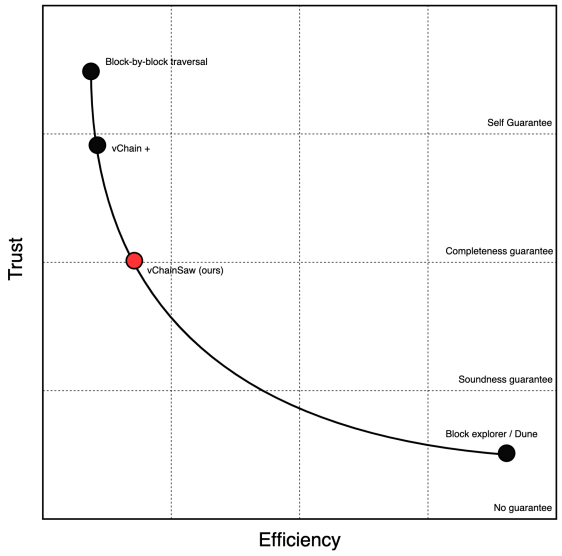


Figure 1: Solution space of vChainSaw

2 RELATED WORK

When blockchains were first introduced, they were not designed to handle large queries. Rather, they were designed to be an immutable, tamper-resistant, and decentralized log of data. These properties made the blockchain particularly well-suited to serving as a ledger for "cryptocurrencies" - and more recently as a distributed virtual machine for executing "smart contract" code. However, since the data within blockchains lack any "schema" or an indexing structure and every application running on the blockchain may structure its data differently, most queries have to be answered by traversing the blocks. This requires running a full node which is prohibitively expensive for most users. The exponential growth in the volume of data being stored on-chain further compounds this. There has been an increase in the number of applications being built on top of blockchains, and various stakeholders - like traders, institutional investors, healthcare providers, and regulatory authorities - are interested in analyzing this data.

We observe that there are several ways of querying blockchain data that are currently in use by practitioners [4] and discuss them here. A popular blockchain system is the HyperLedger system [5]and it uses an additional NoSQL CouchDB database to maintain

the latest state of the blockchain. However, the results of querying this database are not validated against the blockchain and thus may be corrupt. Furthermore, since it stores only the latest state of the chain, there is no efficient way to make queries dealing with the history of the chain. Another popular alternative is to copy the entire blockchain into a traditional relational database and allow users to query it. This is the approach followed by popular block explorers such as EtherScan[1] and blockchain analytics tools such as Dune[2].

However, the above methods are centralized and implicitly require the users to trust the maintainer of the databases. Based on this observation, there have been several approaches to develop **verifiable query systems** that can provide verifiability guarantees in the form of cryptographic proofs. vChain [6] is a state-of-the-art verifiable query system in which the service provider returns not only the result set but also a 'verifiable object (VO)' [9]. The user node can use the VO to verify that the objects in the result set are 'correct'. Cryptographic proofs have also been used as a 'fingerprint' to provide a 'verifiable query layer' in the cloud [7]. Recently, there has also been a focus on using Searchable Symmetric Encryption or SSE to solve the public verification problem [8]. Performance remains the major bottleneck in all of these approaches that sacrifice query response at the altar of verifiability.

Our approach aims to allow the users to relax the completeness constraint and thus gain an improvement in performance. As visualized in Fig 1 we add another point along the 'trust vs. query efficiency' Pareto optimality curve.

3 PRELIMINARIES

3.1 Query Model

In this paper we will be dealing with Boolean range queries over a blockchain. We will now define our model.

Definition 1.1: A *block* is a list of objects $o_1, o_2, ..., o_n$ where each object o_i is a tuple (b_i, V_i, W_i) consisting of a positive integer block number $b_i \in \mathbb{Z}_+$, numberical attributes represented by a vector of integers V_i and set attributes represented by a vector of strings W_i. For example, {1, [300000, 2000], {"Atlanta", "USA}"} could be an object representing a $300000, 2000 sq.ft house in Atlanta in a blockchain comprising of houses and their prices.

Definition 1.2: A *blockchain* is a list of blocks.

Definition 1.3: A *query* is a tuple $< [b_s, b_e], [\alpha, \beta], \delta >$ where b_s and b_e and the start block and end block between which the query is to be processed. α and β are ranges for the numerical attributes V_i and an object can be included in the result set only if it's numerical attributes fall within the ranges specified by α and β. Similarly, δ is a boolean function (in CNF) on the set-attributes W_i. An object is valid only if it's set attributes satisfy the function δ. An example query is [_, {[200000, 300000], [2000, 3000]}, {"Atlanta" ∨ "Augusta"} ∧ "US"]. This query requests all homes (from any block) that cost between $200000 and $300000, and have sizes between 2000 and 3000 sq.ft in either Atlanta or Augusta.

3.2 Mathematical background

Bilinear Mapping

Let $\mathbb{G}$ and $\mathbb{H}$ be two cyclic multiplicative groups of the same prime order p. Let g be a generator of $\mathbb{G}$. A bilinear mapping is a function $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{H}$ such that

(1) $\forall u, v \in \mathbb{G}, e(u^a, v^b) = e(u, v)^{ab}$
(2) $e(g, g) \neq 1$

Cryptographic Multiset Accumulator

A multiset is a set where elements are allowed to be included more than one time. For example, $\{1, 1, 1, 2, 3, 4, 4\}$ is a valid multiset. A multiset accumulator is a function `acc(.)` that maps a multiset (of any size) to an element in a cyclic multiplicative group in a collision resistant manner. It is analogous to cryptographic hash functions such as SHA-256 which map arbitrary length strings to a fixed length number. In addition to outputting a compact digest of a multiset, the accumulator can be used to determine if two multisets are disjoint.

In general, the multiset accumulator comes with four polynomial time algorithms. We will discuss the specific implementations in the next section.

(1) KeyGen(1^λ) $\rightarrow (sk, pk)$. Outputs a key pair given a security parameter λ.
(2) Acc(X, pk) $\rightarrow a$. Given a multiset X and a public key, it outputs the accumulated value a.
(3) ProveDisjoint(X, Y, pk) $\rightarrow \pi$. Given two multisets such that $X \cap Y = \phi$, it outputs a 'proof' π. More details about the proof in the following subsection.
(4) VerifyProof(acc(X), acc(Y), π, pk) $\rightarrow \{0, 1\}$. Given the accumulations of two multisets and a proof, this function outputs whether or not the multisets are indeed disjoint.

3.3 Proofs of disjointedness

In this section we will go over the details of how the four algorithms in the previous section are implemented. This implementation was first suggested in [3].

KeyGen(1^λ) $\rightarrow (sk, pk)$

Takes a bilinear mapping $(p, \mathbb{G}, \mathbb{H}, e, g)$. Chooses an element $s \in \mathbb{Z}_p$. Returns $sk = s$ and $pk = (g, g^s, g^{s^2}, ...g^{s^q})$ where q = `poly`(λ) that is relevant when proving security and unforgeability.

Acc $(X, pk) \rightarrow a$

Here X is a multiset with elements from $\mathbb{Z}_p$. a is calculated as $g^{\prod_{x_i \in X}(x_i + s)}$ where $s = sk$. Notice that from polynomial interpolation the coefficients of the quantity $P(X) = \prod_{x_i \in X}(x_i + s)$ can be obtained and the value $g^{P(X)}$ can be computed solely using the public key elements $(g, g^s, ..., g^{s^q})$.

ProveDisjoint$(X, Y, pk) \rightarrow \pi$

If $X \cap Y = \phi$, the polynomials $P(X)$ and $P(Y)$ share no roots and are therefore relatively prime. From the extended Euclidean algorithm, there must be two polynomials Q_1 and Q_2 such that $P(X)Q_1 + P(Y)Q_2 = 1$. The proof π is computed as (g^{Q_1}, g^{Q_2}).

VerifyProof(acc(X), acc(Y), π, pk) $\rightarrow \{0, 1\}$

Interpret π as (g^{Q_1}, g^{Q_2}). Consider the quantity $e(g^{P(X)}, g^{Q_1}) \cdot$

$e(g^{P(Y)}, g^{Q_2})$. Since e is a bilinear mapping, this equals $e(g,g)^{P(X)\cdot Q_1}$. $e(g,g)^{P(Y)\cdot Q_2} = e(g,g)^{P(X)Q_1+P(Y)Q_2}$. And if the multisets are indeed disjoint the quantity $P(X)Q_1 + P(Y)Q_2 = 1$. Therefore the verify proof function simply checks if $e(acc(X), g^{Q_1}) \cdot e(acc(Y), g^{Q_2}) = e(g,g)$ and returns 1 if they are.

4 METHODOLOGY

4.1 Algorithm

Vchain+ defines 'correctness' of a result set as follows. A result set is correct iff it satisfies two properties. (1) Soundness: all objects in the result set satisfy the query constraints, are part of the blockchain and have not been tampered with and (2) Completeness: all objects in the blockchain which satisfy the query constraints are part of the result set. In order to do this, the VO contains a sequence of 'mini-proofs', one for each object in the blockchain. A 'mini-proof' can be of two types - a match proof or a mismatch proof, if the object in question satisfies the query, the match proof is the object itself along with it's hash. If the object does not satisfy the query constraint, the mini-proof is a mismatch proof and contains a proof produced using ProveDisjoint (from the previous section) that can be used to verify that the set containing the object and the result set must be disjoint and therefore the object cannot be part of the result set.

A mismatch proof is generated as follows. First the query's boolean match condition δ is interpreted as the intersection of one or more sets $\delta_1, \delta_2, ...\delta_k$. For example, if the boolean match condition is "US" $\wedge$ ("Atlanta" $\vee$ "Augusta") this can be expressed as the intersection $\{US\} \cap \{Atlanta \vee Augusta\}$. Then a δ_i is found such that the set attributes W_j of the mismatching object in question o_j is disjoint from δ_i. This means that the o_j cannot satisfy the query and this is evidenced by the proof π returned by `ProveDisjoint(Acc(`W_j`),` $Acc(\delta_i)$`, pk)`. Thus the verification object for the object o_i will be $< \pi, \delta_i >$. Although we have only discussed how the set-valued attributes are handled, the numerical attributes are reduced to set-values using prefix trees and handled similarly. The Vchain paper goes into more detail.

In this paper, we notice that in most queries, the majority of the VO size comes from the 'mismatch proofs' since most objects in the chain will not be part of the result set. Additionally, mismatch proofs require more computation since the set δ_i must be found and the proofs generated. The completeness constraint necessitates the mismatch proofs since the user must not only be convinced that every item in the result set matches the query but also that every item not in the result set 'mismatches' the query. We propose that a proof of 'completeness' for set of results is often unnecessary and introduce a new constraint, N-completeness where N is a user-defined parameter.

Definition 1: A result-set is N-complete if the user can verifiably conclude that the result-set contains atleast N-fraction of all objects in the chain that satisfy the query.

In particular the VChain definition of completeness is equivalent to 1-completeness, since it requires all objects that satisfy the predicate of the Boolean range query to be present in the result set. If the user sets the value of N to 0.5, the proof π that is returned with the result set can be used by the user node to verify that the result set contains atleast 50% of all objects that satisfy the query.

In order to allow the user to check for N-completeness (instead of 1-completeness), we will need to modify the proof technique. We shall describe our methodology by means of an example. Suppose the blockchain that we are querying has 100 blocks each containing 1 object (for the sake of simplicity). Let us say the user makes a query q that matches the first 50 objects and mismatches the rest. Clearly the result set should contain 50 objects. The interesting part is the contents of the VO. If the requirement was for 1-completeness (the default requirement in vChain) the VO would contain 50 match proofs and 50 mismatch proofs. However, if the user requirement is only 0.625-completeness, the verification object can contain 50 match proofs (same as before) and only 20 mismatch proofs (instead of 50 before). The user can use these 50+20=70 proofs to verify the match/mismatch status of 70 out of the 100 objects. The status of the remaining 30 objects is unknown to the user but regardless of whether those 30 objects are matches or mismatches, the user can be sure that even if the 30 unknown objects are matches, 50/(50+30)=0.625 fraction of the 'true' matches have been returned in the result set which is a 'good enough' proof. The general idea is that although an honest implementation of this protocol will always return a complete and sound set of results, the strength of the proof (and size of the VO) can be tweaked based on user preference. If the user only wants to verify that atleast 50% of all valid results are infact in the result set, the service provider will only generate a proof that is 'good-enough' to satisfy the user and will not burden itself with the extra effort of filling in all the mismatch proofs.

In general, suppose the user requests an N-complete proof, where N is a real number in the range $[0, 1]$, and requests a query q. Suppose a fraction f objects in the chain satisfy the query constraints q and the total number of objects in the chain is t. Then the VO will contain a total of $f \cdot t$ match proofs and $(1 - \frac{f}{N}) \cdot t$ mismatch proofs if $N > f$ and 0 mismatch proofs if $N \leq f$. The pseudocode for our algorithms are given in Algorithms 1 and 2. The service provider runs Algorithm 1 and the user node runs algorithm 2.

Algorithm 1 Generating proofs for a list of objects o_i

```
Input: objects O = [o_1...o_m], query q = (_, _, δ), trust parameter n.
MatchProofs = Match(O, q)
matches = len(MatchProofs)
MismatchProofs = {}
for i in [1, m] do
    if len(MismatchProofs) > m - (matches/n) then
        break
    end if
    if o_i mismatches q then
        Interpret δ as δ_1, ..., δ_j
        Find δ_k such that δ_k ∪ W_i = φ
        π ← ProveDisjoint(Acc(W_i), Acc(δ_k), pk)
        MismatchProofs.push(π, δ_k)
    end if
end for
return MatchProofs, MismatchProofs
```

Algorithm 2 Verifying proofs for a list of m objects o_i

Input: MatchProofs = $[m_1 ... m_a]$ MismatchProofs = $[s_1, s_2, ..., s_b]$, query $q = (\_, \_, \delta)$, trust parameter n.
for m **do** in MatchProofs
  Check if m matches hash(m) from the block header
  Check if m matches query q
end for
for s=(π, δ_a) in MismatchProofs **do**
  VerifyProof(Acc(o_i), Acc(δ_a), π, pk)
end for
Check if len(matches)/(len(matches) + (m - len(mismatches))) > n
If all checks pass, return 1 else 0

4.2 Analysis

The total number of proofs in the verification object depends on three factors. 1) the total number of objects in the range in question 2) the number of objects that match the query and 3) the trust parameter n. In the below figures we have plotted the total number of proofs needed in the verification object on the y-axis against the number of objects that match a query (out of a 100) on the x-axis.

From figure 2 below we see that if the trust parameter n is larger than the fraction of matches, the number of proofs reduce with reducing n. However, if the value of n is smaller than the fraction of matching objects, the number of proofs remains unchanged with reducing n. This means that in queries with a small match fraction the effect of changing n is more pronounced.

5 EVALUATION PLAN

5.1 Thesis

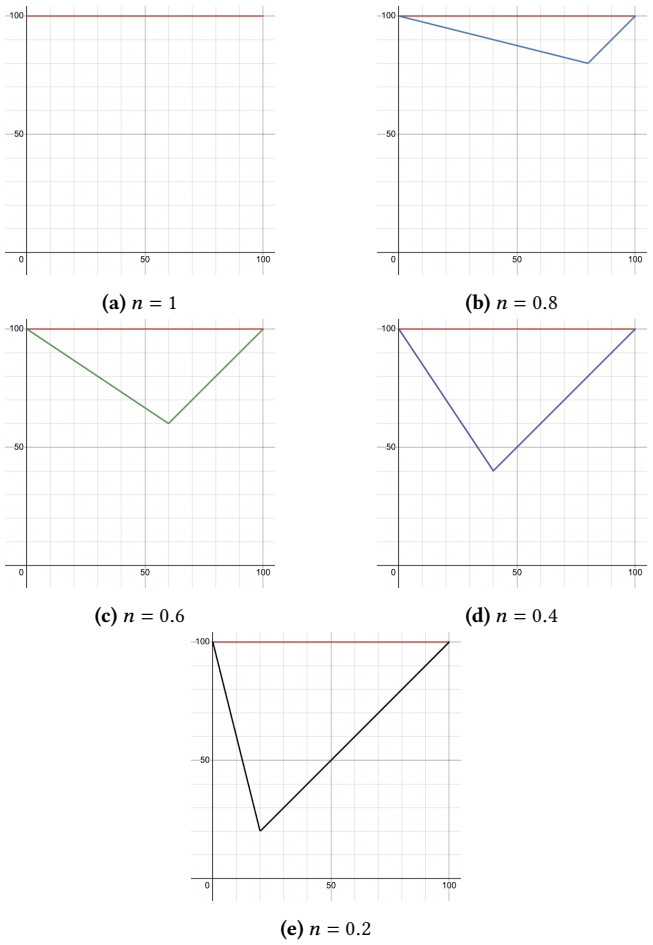
Current blockchain querying systems have an "all-or-nothing" view of trust - in particular, the user node is either completely trusting of the service provider and does not verify the results at all, or is completely untrusting of the service provider and verifies both soundness and completeness of the result set. We propose to introduce a parameter that allows the user to customize the level of trust that they are willing to grant the service provider. This is useful, especially in approximate query contexts where completeness constraints can be loosened in exchange for better query response times.

5.2 Claim

In this work, we posit that by relaxing the completeness constraints imposed by current verifiable query approaches, we can reduce the size of the verifiable object and hence lower query response time.

5.3 Evaluation Design

Dependent Variable. The dependent variable that we measure is the end-to-end query latency. End-to-end query latency comprises two components. Firstly, the proof generation time which is the time taken by the service provider to generate the VO. Second, the proofreading time, is the time taken by the user node to verify the result-set using the VO. The dependent variable is directly correlated to the size of the VO, as a larger VO implies a larger proof generation time, as seen by the results in VChain. The size of the VO depends on several parameters, which we discuss in the next section.



(a) $n = 1$ **(b)** $n = 0.8$ **(c)** $n = 0.6$ **(d)** $n = 0.4$ **(e)** $n = 0.2$

Figure 2: Variation of total number of proofs with matches per 100. x-axis contains number of matches and y-axis the total number of proofs needed

Independent Variable.

Different data structures such as B-Plus trees, ID Trees, and Tries are used to answer different queries over blockchains. The algorithms used to generate proofs in each of these data structures have different effects on the query latency, as we discuss in more detail in section 6.2. We omit the proofs for these data structures as the independent variable in this study.

Dataset and benchmark. We use the same benchmark used by the current state-of-the-art [6] in our experiments. We generate 10 random queries per experiment and execute them over Ethereum and FourSquare datasets. In particular, we evaluate the CPU time taken to process the query and verify the generated proof. We discuss further details regarding the benchmark in the Evaluation section.

Query Performance (seconds)	10 blocks	20 blocks	30 blocks	40 blocks	50 blocks	60 blocks	70 blocks	**300 blocks**
Baseline Q1	12.28	17.61	23.52	28.73	39.48	45.23	51.59	210.30
Baseline Q2	6.220	8.710	11.66	14.12	19.27	22.73	25.86	105.47
Baseline Q3	7.46	11.2	15.98	18.18	25.77	29.32	35.18	**148.44**
Q3 without bplus tree proofs	7.11	10.86	14.37	17.75	24.26	29.17	33.36	**122.23**
Q3 without trie proofs	7.04	10.53	13.74	16.57	23.88	28.08	31.7	140.99

Table 1: Evaluation over VChain benchmark

We expect that there will be significant improvements in latency when we relax completeness in the case where a 'significant' fraction of the blocks in the chain satisfies the query constraints. This is a direct consequence of the method we use to generate 'good-enough' proofs (as described in the methodology section). However, the threat is what happens when only a small fraction of blocks in the chain satisfies the query. We are unsure of whether the improvements will be noticeable in this case.

6 EVALUATION

We study the effect of relaxing completeness in the following experimental setting.

6.1 VChain Benchmark

This benchmark consists of two datasets. The first, Foursquare (4sq), includes 1M timestamped records containing check-in information of users/ The second is the Ethereum (eth) dataset that consists of the 3.27M transaction records contained in the 58,100 blocks that represent the state of the Ethereum blockchain between Dec 17, 2018 to Dec 26, 2018. The boolean range queries used over these datasets are programmatically generated and for each data point, the original benchmark uses the average processing time and average verification object size over 10 queries. The verification object size varies drastically based on the selectivity of the queries, in the range of 1 to 1000 KBs.

The processing time is the total time taken to generate the proof (a service-provider computation) and verify the result-set (a user computation). We do not account for transfer time over the network in this work.

In our analysis, we aim to execute each query with varying completeness constraints in order to study the effect of varying completeness. We **expect to** observe, that broadly, reducing completeness leads to a reduction in query latency. However, the effect would be more pronounced when the selectivity of the query is larger, i.e. the number of rows in the result-set is higher.

We execute all experiments on a 2017 MacBook Pro, with a 4-core i7-6700K processor.

6.2 Relaxing Completeness

Prior works on verifiable queries propose using different data structures such as B-Plus trees and Tries to efficiently index data to answer different query types such as boolean range queries or keyword-based set membership queries [6]. Each of these data structures use different proof computation approaches, with varying performance profiles. In each experiment, we only generate the match proofs for one of these data structures while omitting the mismatch proofs. The remaining data structures generate both match and mismatch proofs. For our baselines, we execute three sample queries Q1, Q2, and Q3 on the entire system with all proofs being generated. As per Table 1, we find that omitting mismatch proofs for BPlus trees results in a nearly 14% speedup.

7 CONCLUSION

In conclusion, we hypothesized that relaxing the constraints on completeness will reduce query latency and tested our hypothesis by removing B-Tree and Trie mismatch proofs and observed an increase in query latency. While we did not experimentally verify the effect of N on query latency, we analyzed the case mathematically and calculated and plotted the reduction in total proof size while varying n and the number of matches.

8 FUTURE WORK

There are several experiments that we plan to run to help further evaluate our system.

8.1 Dune Benchmark

The VChain Benchmark uses real-world datasets with synthetic queries. In order to better study the performance of real-world analytics workloads, we look to the 'most-popular dashboards' on Dune, a popular blockchain analytics website that allows users to query the Ethereum blockchain using SQL-like syntax. After examining the queries used in the top 10 most popular dashboards (as of Nov 15, 2022), we note that the average user query is complex, and frequently utilizes aggregation operations over GROUP BY clauses. Since we only support Boolean-range queries we remove any queries that include GROUP BY's from our evaluation.

Although we were not able to evaluate our solution on the Dune benchmark in greater detail, we plan to do so in future work. We expect the fraction of the blocks that are part of the result-set to be much higher on the Dune benchmark than the VChain benchmark as the latter comprises real-world queries. As a consequence, we expect to see that the effect of parameter N is more pronounced. However, if this is not the case - and the fraction of blocks present in result-set is similar to or lower than VChain then we would observe that changing the value of N does not have an appreciable effect on processing time.

8.2 Verification Object Size

In this study, we measure end-to-end query latency as the primary dependent variable. Although this is correlated to the size of the verification object, we hope to also consider VO size directly in future experiments.

8.3 Optimal N-Completeness

In this work, we introduce the new notion of N-completeness - a result-set is N-complete if the user can verifiably conclude that the result-set contains atleast N-fraction of all objects in the chain that satisfy the query. In particular, the VChain definition of completeness is equivalent to 1-completeness, since it requires all objects that satisfy the predicate of the Boolean range query to be present in the result set. In future experiments, we hope to find find the optimal N and how it affects both the size of the verification object and the query latency. ($N \in R, 0 < N \leq 1$)

9 ACKNOWLEDGEMENTS

We thank the instructor and course staff of the 'CS 8803 MDS - Human-in-the-loop Data Analytics' course for their feedback on earlier drafts of the paper.

REFERENCES

[1] [n.d.]. https://etherscan.io/
[2] [n.d.]. https://dune.com/home
[3] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. 2011. Optimal Verification of Operations on Dynamic Sets. In *Advances in Cryptology – CRYPTO 2011*, Phillip Rogaway (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 91–110.
[4] Dennis Przytarski, Christoph Stach, Clémentine Gritti, and Bernhard Mitschang. 2021. Query processing in blockchain systems: Current State and future challenges. *Future Internet* 14, 1 (2021), 1. https://doi.org/10.3390/fi14010001
[5] Ankur Sharma, Felix Martin Schuhknecht, Divya Agrawal, and Jens Dittrich. 2019. Blurring the Lines between Blockchains and Database Systems: The Case of Hyperledger Fabric. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 105–122. https://doi.org/10.1145/3299869.3319883
[6] Haixin Wang, Cheng Xu, Ce Zhang, Jianliang Xu, Zhe Peng, and Jian Pei. 2022. vChain+: Optimizing Verifiable Blockchain Boolean Range Queries. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 1927–1940. https://doi.org/10.1109/ICDE53745.2022.00190
[7] Haotian Wu, Zhe Peng, Songtao Guo, Yuanyuan Yang, and Bin Xiao. 2022. VQL: Efficient and Verifiable Cloud Query Services for Blockchain Systems. *IEEE Transactions on Parallel and Distributed Systems* 33, 6 (2022), 1393–1406. https://doi.org/10.1109/TPDS.2021.3113873
[8] Haotian Wu, Rui Song, Kai Lei, and Bin Xiao. 2022. Slicer: Verifiable, Secure and Fair Search over Encrypted Numerical Data Using Blockchain. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. 1201–1211. https://doi.org/10.1109/ICDCS54860.2022.00118
[9] Cheng Xu, Ce Zhang, and Jianliang Xu. 2019. VChain: Enabling Verifiable Boolean Range Queries over Blockchain Databases. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 141–158. https://doi.org/10.1145/3299869.3300083