Exploring Locality in Ethereum Transactions

Matthieu Pigaglio
UCLouvain, Belgium

Michał Król
City, University of London, United Kingdom

Etienne Rivière
UCLouvain, Belgium

***Abstract*—Early open blockchain designs faced low throughput, high latency, and prohibitive costs for setting up a full node. New designs improve this with innovative mechanisms for handling transactions and the blockchain state, often assuming locality properties in the workload of transactions. Temporal locality allows efficient space management such as light nodes or snapshot-based bootstrap. Disjoint access parallelism, which depends on spatial locality, enables parallel processing of non-conflicting transactions. We analyze locality properties and their interplay in the largest transactional workload available to date, that of Ethereum. Our results show that, although transactions generally display good locality, a minority of accounts are responsible for caching- or parallelism-unfriendliness, calling for specific identification and handling in future blockchain designs.**

I. Introduction

Blockchain networks such as Cardano [1], Ethereum [34], or Solana [35] attract considerable attention due to their capacity to enable trust in large-scale, decentralized systems despite the lack of mutual trust between participants. An open blockchain allows any participant to join in building a globally-replicated and eventually tamper-proof distributed ledger containing *transactions*. Early blockchain designs introduced important concepts enabling decentralization and robustness, such as proof-of-work consensus in Bitcoin [27] or smart contracts in Ethereum [34], enabling a large variety of applications over blockchains such as for supply chains or digital health [18].

A significant body of work proposes new blockchain designs that exploit the *nature* of the transaction workload in open blockchains, to ensure higher decentralization [8], [9], [22], [29], [37] (i.e., to make it easier and less costly to enter as a new participant) and better performance [4], [5], [11], [14], [15], [24], [25], [30], [32] (i.e., to support higher throughputs). Common to these proposals is the exploitation of *locality* properties in the workload of transactions.

Several designs exploit *temporal* locality in the pattern of accesses to the ledger state when validating transactions and executing smart contracts. These properties are of paramount importance for the design of caches in computer systems, following the *working set* model [13]. Light clients [9], such as CoVer [8] or Subset nodes [29], exploit temporal locality properties to allow nodes with only a subset of the ledger state to participate in the network. Similarly, Fynn *et al.* [37] or Ethanos [22] propose *snapshotting* mechanisms that allow bootstrapping new full nodes with a previous state materialization, based on the assumption that incremental checkpoints only include a relatively small subset of the entire ledger state.

Other approaches leverage the inherent *disjoint-access parallelism* (DAP) of transactions, i.e., transactions accessing *disjoint* subsets of accounts. This is a form of spatial (non)locality, allowing disjoint working sets that can be processed in parallel. Forerunner [11], Block-STM [15], OptSmart [5], ParBlockchain [4], or the work of Dickerson *et al.* [14] employ speculative execution on a single node to speed up the processing of transactions; all assume that the workload exhibits sufficient DAP. Other systems leverage distributed execution via sharding, where the load of processing transactions is split between different sub-chains [32] or use multiplexed consensus [24]. Dynamic assignment algorithms for transactions (e.g., OptChain [28]) or accounts (e.g., Shard Scheduler [25]) detect and exploit DAP while processing transactions, assigning disjoint working sets to different shards.

Contributions and outline. We propose a thorough analysis of the largest transactional workload to date, that of the Ethereum [34] blockchain, with a focus on locality properties. Our study considers the historical evolution of locality properties and other system-relevant characteristics over time, from the early days of Ethereum until its recent switch to proof-of-stake consensus (the "merge" [2]).

We provide a thorough evaluation of metrics of interest for a transactional workload, which we formalize in a model (Section II) and extend to allow historical analysis (Section III). After a study of general metrics characterizing transactions (Section IV), we present in-depth analysis of temporal locality properties (Section V) and of DAP/spatial locality properties (Section VI). We finally review related studies (Section VII) and conclude (Section VIII).

II. Background

We cover background material and provide a generic model of an account-based blockchain (§II-A). Note that our focus in this paper is in understanding the properties of a large transactional workload, but not to focus on the specificities of the Ethereum protocol. Therefore, we only briefly cover Ethereum designs aspects that are relevant to our study (§II-B).

A. Generic model

Our generic model focuses on accounts and transactions.

Accounts. The state of a ledger is a map between a set of *accounts* A and their associated state. An account has a specific address $a \in A$. There are two types of accounts: (1) externally owned accounts (EOA) are under the responsibility of an external user of the blockchain, who can request transfers or the execution of smart contracts. (2) Smart contract accounts (SCA) represent executable programs stored on the chain.

Transactions. Transactions are state mutations that result from the transfer of assets and the execution of smart contracts. We note T the set of all transactions and t a specific transaction. T admits a total order, i.e., $T = (t_1, t_2, \ldots, t_n, \ldots, t_{|T|})$. The effects of a transaction t deterministically depend on the state of a number of accounts at the time of its execution, which we denote as its read set $t.\mathsf{read}$. The transaction t in turn mutates the state of a number of accounts, forming its write set $t.\mathsf{write}$. The two sets may overlap.

We distinguish between four types of transactions. (1) A *transfer* transaction is emitted by the owner of an EOA to transfer an amount of cryptocurrency to another EOA or SCA account. (2) A contract *deployment* transaction is emitted by an EOA and installs a new smart contract. (3) An *external* transaction is made by the owner of an EOA to an SCA (smart contract) to trigger its execution. An external transaction can in turn trigger an arbitrary number of (4) *internal* transactions. In contrast with (1)–(3), internal transactions (4) are not registered on the ledger but determined when executing or validating external transactions, depending on the current state A. An important implication is that the content registered in the ledger is only a *subset* of T, the set of all transactions.

B. Ethereum

Ethereum [34] is the second largest blockchain after Bitcoin [27], and the largest one supporting smart contracts. It adopts the account-based model as defined in the previous subsection. We use the content of the Ethereum ledger as the data source for our analysis. The order of transactions in T is the result of Ethereum's consensus, which was based on proof-of-work until the recent switch to proof-of-stake ("the merge" [2]). We consider all transactions from the beginning of Ethereum to the merge. This represents 1.7 Billion transactions involving 180 million different accounts (177M EOAs and 3M SCAs). Proof-of-work was used over the evaluated period., which can lead to forks and *uncle* blocks (i.e., blocks part of discarded branches), but we consider only the content of the longest chain.

In Ethereum, smart contracts may execute arbitrary code compiled for the Ethereum Virtual Machine (EVM). To bound execution times, the blockchain uses the principle of gas, expressed in the embedded cryptocurrency. Gas is associated with transactions and consumed by miners when validating them. While this mechanism details are unimportant for our study, the amount of gas associated to a transaction execution is an interesting estimation of the amount of computation it requires: each EVM instruction, and the transaction as a whole, map to a specific amount of gas [7]. We note the amount of gas for a transaction t as $t.g$, and the total of all spent gas for the set of transactions T as $T.g$.

III. Methodology

We detail now how we structure and enable our analysis of the Ethereum dataset. First, we present how we enrich the base model of Section II with additional, calculated metrics (§III-A). Then, we detail how we enable a time-based analysis of the dataset via *windowing* and the associated model extension (§III-B). Finally, we present EthEx, our tool for efficient collection and analysis of this massive dataset (§III-C).

A. Calculated metrics

The base model contains the set of transactions T and the set of accounts A. Both are collected by *re-executing* the content of the ledger. We enrich this base model with additional information as follows.

Age-based metrics for accounts. We are interested in the history of the use of each account by the sequence of transactions. The history $a.\mathsf{H} = a.\mathsf{H_R} \cup a.\mathsf{H_W}$ of an account $a \in A$ is the sequence of transactions where a is used in their read or write sets, i.e., $a.\mathsf{H_R} = \{t \in T \mid a \in t.\mathsf{read}\}$ and $a.\mathsf{H_W} = \{t \in T \mid a \in t.\mathsf{write}\}$.

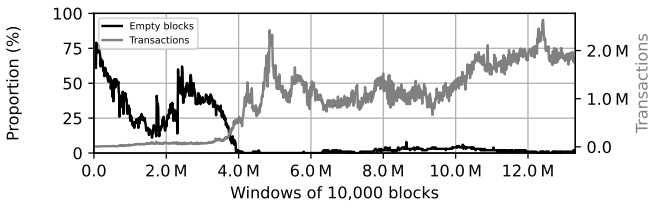
This history starts with the creation of the account. An account starts to exist the first time it is used in the write set of a transaction. We note it $a.\mathsf{creation} = \min(i \mid t_i \in a.\mathsf{H_W})$; its last use is $a.\mathsf{last} = \max(i \mid t_i \in a.\mathsf{H})$. The *lifetime* of an account is the span of its use in transactions, i.e., $a.\mathsf{lifetime} = a.\mathsf{last} - a.\mathsf{creation}$. Finally, the *frequency* of use of the account over its lifetime is $a.\mathsf{freq} = |a.\mathsf{H}|/a.\mathsf{lifetime}$.

We use these metrics to relate accounts used in the read and write sets to the characteristics of the accessed accounts, and primarily determine whether assumptions on temporal locality are validated in the data set.

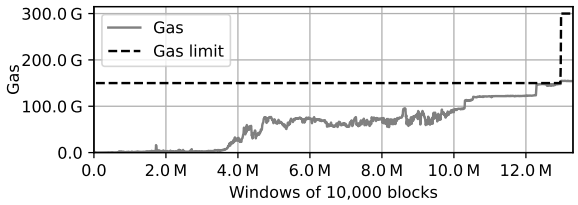
Interaction graph. We are also interested in studying the interactions between accounts *via* transactions. We define an interaction graph IG where the set of vertices $V_{IG} = A$, the set of accounts. Undirected edges of this graph $E_{IG} \in A \times A$ represent interactions between accounts, i.e., that they have been accessed as part of the same transaction(s). We note $I(a_i, a_j) = \{t_x, t_y, \ldots, t_z\}$ the set of transactions in which both a_i and a_j are involved, i.e., $I(a_i, a_j) = \{t_i \in T \mid \{a_i, a_j\} \subseteq (t_i.\mathsf{read} \cup t_i.\mathsf{write})\}$. The weight $e(a_i, a_j).\delta$ of an edge e in IG is the number of common transactions, i.e., $\forall e(a_i, a_j) \in E_{IG}, e(a_i, a_j).\delta = |I(a_i, a_j)|$.

B. Time analysis and windowing

We are interested not only in analyzing the Ethereum workload *as a whole*, but also to study the evolution of locality properties over the history of the workload. To study this evolution, we partition the workload in a series W of *windows*, consecutive in time, i.e., $W = (w_1, \ldots, w_{|W|})$. We choose to form these windows with a fixed number of *transactions*, 1,000,000 by default. The first window is $w_1(T) = (t_1, \ldots, t_{1000000})$, the second is $w_2(T) = (t_{1000001}, \ldots, t_{2000000})$, and so on. Our approach contrasts with previous work [6], [31], [36] using partitioning by windows of *blocks*. This choice of using windows of blocks would go, in fact, against our will to study the nature of the transaction workload *as is*, i.e., not as it was specifically processed by Ethereum but as it could be processed by alternative designs (for instance, using parallel execution [25], [30], [32]). Relying on *blocks* would result in highly heterogeneous windows, and be a source of bias, as we illustrate in Figure 1. Figure 1a



(a) Fraction of empty blocks and number of transactions.



(b) Cumulated amount of gas.

Fig. 1: Observed metrics when considering groups of 10,000 successive blocks. The amount of transactions and their complexity varies significantly over successive groups, making this discretization ill-suited for studying transaction history.

(`y1` axis) plots the proportion of *empty* blocks for windows of 10,000 blocks (i.e., with 13,000,000 blocks we obtain 1,300 such windows), while Figure 1a (`y2` axis) plots their number of transactions and Figure 1b present their cumulative amount of gas. We observe that the number of empty blocks is significant in the early history of the transaction workload, and never reaches 0%. The number of transactions per group of blocks fluctuates greatly, which can be explained by the variation in the number of internal transactions, which is capped by a maximal amount of gas allowed in a block, itself evolving with updates to the protocol.

Enriched model and windowing. We note $w_i(\cdot)$ the value of a metric over the i^{th} window; $w_i(T)$ is a set of 1 million transactions and $w_i(A)$ is the set of accounts that appear at least once in the read and write sets of transactions in $w_i(T)$. The interaction graph over a window $w_i(IG)$ uses as vertices $w_i(A)$; edges follow the earlier definition using only transactions in $w_i(T)$. The gas cost of all transactions in a window is defined as $w_i(T.g) = \sum_{t_x \in |w_i(T)|} t_x.g$.

The subset of the history of an account in window w_i is $w_i(a.\mathsf{H}) = \{t \in a.\mathsf{H} \mid t \in w_i(T)\}$. Age-based metrics cannot be derived from the direct application of their global definition to such a window of transactions (e.g., $w_i(a.\mathsf{creation})$ would be the first use of the account in this window, which has nothing to do with its creation).

We define as $W(\cdot)$ the windowed-versions of these metrics to analyze at a coarser grain the lifetime characteristics of accounts. $W(\cdot)$ metrics always return a window identifier. As a result, $W(a.\mathsf{creation}) = \min(i \mid w_i(a.\mathsf{H}) \neq \emptyset))$, $W(a.\mathsf{last}) = \max(i \mid w_i(a.\mathsf{H}) \neq \emptyset))$, and $W(a.\mathsf{lifetime}) = W(a.\mathsf{last}) - W(a.\mathsf{creation})$. The frequency of use is also modified accordingly, to reflect the frequency of windows that include a transaction involving that account over its lifetime, also expressed as a number of windows: $W(a.\mathsf{frequency}) = (|\{i \mid w_i(a.\mathsf{H}) \neq \emptyset\}|)/(W(a.\mathsf{lifetime}))$.

Finally, we define two metrics allowing to study, from the perspective of a given window w_i, the relative history of accounts accessed in w_i. The *age* of an account from the perspective of w_i denotes how recently (i.e., how many past windows ago) the account was *created*: $w_i(a.\mathsf{age}) = i - W(a.\mathsf{creation})$. We also define the *last_use* metric to measure how recently (in number of windows) an account a was *used* for the last time. This metric would typically be used for cache management policies such as Least-Recently-Used (LRU). By convention, the *last_use* metric is 0 for a newly created account: $w_i(a.\mathsf{last\_use}) = \max(j < i \mid a \in w_j(A))$ if $\exists\, j < i \mid a \in w_j(A)$, and 0 otherwise.

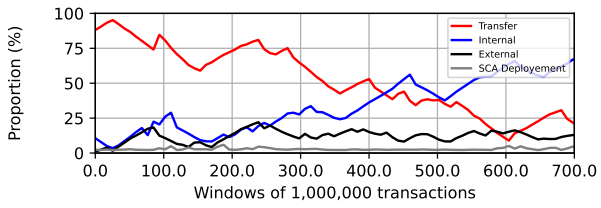


Fig. 2: Evolution of the distribution of transaction types.

C. EthEx analysis tool

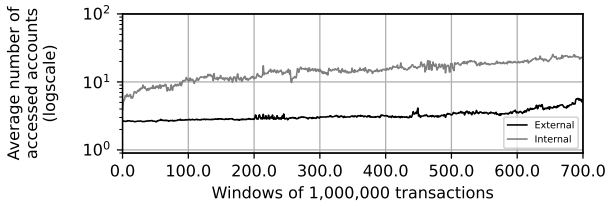
We implemented EthEx, a tool for the analysis and characterization of the Ethereum transaction workload.[1] EthEx builds the information forming our model, including calculated metrics, both for the global and windowed information, on the basis of set of transactions resulting from the validation of the ledger by the Go Ethereum (`geth`) client [3]. The collection and validation/replay of the chain data (about 1.2 TiB and 1.9 Billion transactions) takes about 3 days and results in a dataset for `geth` of about 2 TB. EthEx interfaces with `geth` to build the model, using parallel computation (about 3 hours with 16 threads). The complete, enriched model takes about 600 GB of storage, which we can process linearly to extract the metrics of interest that we present in the rest of the paper.

IV. Analysis: General Metrics

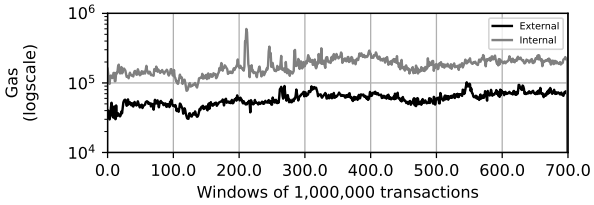
We initiate the study by making general observations on the workload, not directly linked to locality but helping to understand the nature of the transactional workload.

Types of transactions. We first investigate the distribution of the types of transactions and its evolution over time. In total, T contains 1.3 Billion transactions, with 42% of *transfer*, 7% of *external*, 49% of *internal*, and 2% of *deployment* transactions. Figure 2 presents the evolution of this distribution over windows. We observe that transfer transactions dominate the early history, where they represent more than 80% of transactions, to gradually become marginal (around 20% close

[1]We intend to release our tool as well as all material allowing the reproducibility of our results, together with the final version of this paper.

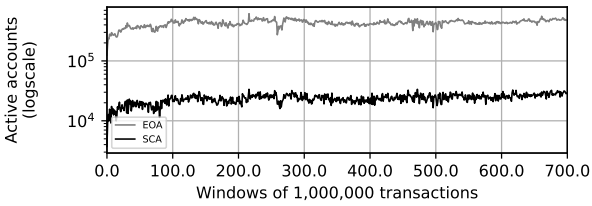


(a) Average size of the accessed state in each window w_i, $\sum_{t \in w_i(T)} |t.\mathsf{read} \cup t.\mathsf{write}|/|w_i(T)|$.

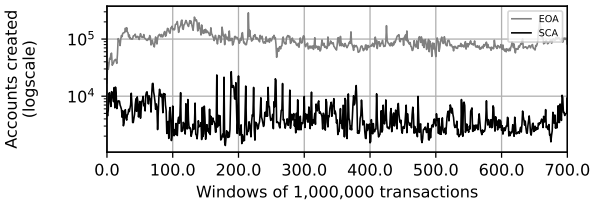


(b) Average *gas* of transactions in each window w_i, $\sum_{t \in w_i(T)} t.g/|w_i(T)|$.

Fig. 3: Evolution of the accessed state size and computational complexity, for internal and external transactions.



(a) Unique accounts used in each window w_i, i.e., $|w_i(A)|$, for externally-owned accounts (EOA) and smart contract accounts (SCA).



(b) Evolution of the number of account creations per window w_i, i.e., $|\{\sum a \in A \mid W(a.\mathsf{creation}) = i\}|$ for EOA and SCA accounts.

Fig. 4: Evolution of the working set size (accessed accounts) and new state size (created accounts).

to the merge). Transactions linked with smart contracts have become dominant. We observe, however, that this is mostly a result of a steadily increasing number of *internal* transactions; the number of *external* transactions is quite stable.

Transactions complexity. The complexity of transactions can be expressed as the size of their accessed state and as their computational requirements. The former is reflected by the number of accounts they access, i.e., for a transaction t, $|t.\mathsf{read} \cup t.\mathsf{write}|$. The latter can be approximated by the amount $t.g$ of *gas* its execution requires. We focus here on *internal* and *external* transactions. Figures 3a and 3b present the average state size and average computational complexity and its evolution over time for these two categories. We make the following observations. The number of accounts accessed by internal transactions is significantly higher than the number of accounts accessed by external ones, with about a x4 difference between the two–note the logarithmic y axis scale in Figure 3a. This number increased over time for both types of transactions. In contrast, the computational complexity of individual transactions is relatively stable over time, with internal transactions spending roughly two times as many computational steps as external ones, with no notable long-term trends. The progressive dominance of internal transactions (Figure 2) shows, however, that each *external* transactions triggers increasingly more *internal* ones. Our observation is that transaction complexity significantly increased over time, both considered from the point of view of individual ones for the cumulated work associated with each external transaction.

V. Temporal Locality

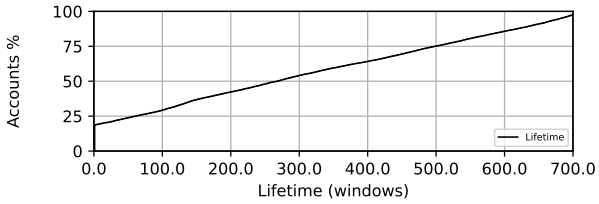
We study in this section the temporal locality of the workload. We follow the traditional *working set* model [13]. Instead of memory pages, we consider accesses to individual accounts. Instead of time intervals, that would depend on a specific consensus protocol, we consider windows of transactions.

Temporal locality is the property that recently-accessed data is likely to be accessed again in the near future. Popular cache management strategies rely on this property, e.g., using Least-Recently-Used replacement policies. In the context of blockchains, temporal locality is assumed by designs for light clients [8], [9], [29] and for speeding up the bootstrap of new nodes using snapshotting techniques [22], [37]. These designs rely, indeed, on the fact that state accesses are more likely to target a subset of recently-accessed accounts.

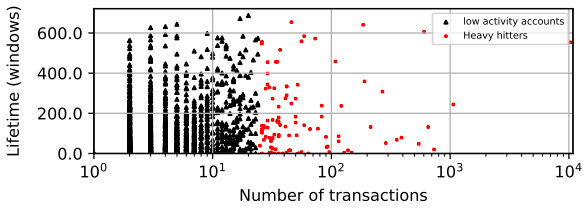
Working set size and account creations. We start by observing the size of the working set and its evolution over time. The size of the working set over a window w_i is the number of unique accounts accessed by its transactions, and is plotted in Figure 4a, distinguishing between EOA and SCA accounts. We observe a one order-of-magnitude (OOM) difference between the number of EOA accounts and the number of SCA accounts accessed in each window. We contrast the stability we observe in the working set size with our previous observation that internal and external transactions came to dominate the workload in later windows; the complexity of individual calls to smart contracts has increased while these calls access similar numbers of accounts.

We observe the number of accounts *creations* in each window in Figure 4b. This figure uses a logarithmic scale for the ordinates, for the sake of comparison with Figure 4a. We report in the following text the ratio between the number of accounts accessed and the number of accounts created.

We observe that between 19% and 25% of EAO accounts and between 15% and 23% of SCA accounts accessed in each window are *created* in that window, implying a high degree of

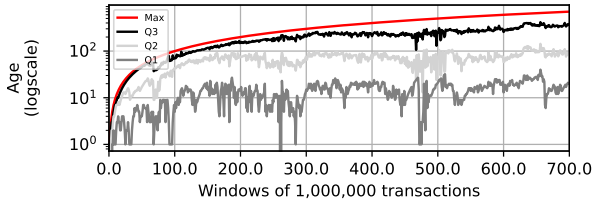


(a) CDF of accounts lifetime, i.e., $W(a.\text{lifetime})$.

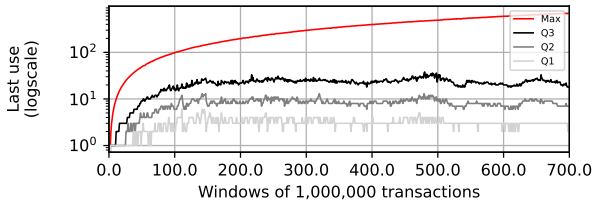


(b) Scatter plot linking accounts' lifetime $W(a.\text{lifetime})$ (in windows) and their volume of transactions. We consider a subset of 100,000 random accounts. We plot all points for accounts with $\geq$25 transactions (red dots, "heavy hitters"), while we plot only 1% of the points for accounts with fewer transactions (black dots).

Fig. 5: Distribution of accounts lifetimes, and interplay between lifetimes and numbers of transactions.



(a) Distribution of the age (in windows) of accounts accessed in window w_i, i.e., $\forall a \in w_i(A), w_i(a.\text{age})$.



(b) Distribution of the last use (in windows) of accounts accessed in window w_i, i.e., $\forall a \in w_i(A), w_i(a.\text{last\_use})$.

Fig. 6: Evolution of the distributions of locality metrics represented as quartiles (25$^{\text{th}}$, 50$^{\text{th}}$, 75$^{\text{th}}$ perc. and max.).

volatility. Over the complete chain, 22% of transactions issued by EOA accounts are from new accounts (19% for SCAs). Volatility impacts temporal locality, as the working set undergoes significant changes for every window. An incremental snapshot mechanism would, for instance, require saving this new state; a caching mechanism would have to avoid that new content evince warm content likely to be accessed again.

Lifetimes and activity. We now study the lifetime of accounts expressed as a number of windows, i.e., $W(a.\text{lifetime})$ for an account a. The cumulative distribution function (CDF) of these lifetimes is represented by Figure 5a. First, we observe that 19% of all accounts are involved in a single window. In fact, 18% of the accounts are even involved in a single *transaction*, i.e., they are used once and never again. Second, we observe that the median lifetime is quite long, at about 260 windows in a dataset formed of 700 such windows. Lifetimes are, in fact, distributed almost uniformly between short-lived and long-lived accounts, spanning all intermediate values. While the lifetime of an account indicates how spread in time this account has been used, it does not shed light on how *frequently* the account was used. A frequently-accessed account over a short time period has, indeed, a good temporal locality. An account accessed sporadically over a long period of time has, conversely, poor temporal locality. We relate the two metrics in the scatter plot of Figure 5b. Note that to favor visual clarity the figure presents a subset of the data, as detailed in its caption. First, we can observe that accounts with long lifetimes are not necessarily very active, and the other way around. Second, long-lived accounts exist for different volumes of transactions. Accounts represented as black dots at the top-left of the graph have poor temporal locality, as they are accessed sporadically and over a long period of time. Accounts represented as red dots at the bottom-right are, in contrast, good candidates for locality: they are accessed often over a short period of time. If we take as a reference the point (25 transactions, lifetime of 200 windows), we observe that the 100,000 accounts considered are distributed as follows: 22% are in the top-left corner, 68% in the bottom-left corner, 2% in the top-right corner, and 8% in the bottom-right corner. This means that, using this reference point, about 76% of accounts *should* have good temporal locality of access, while the remaining 24% *may* have bad temporal locality of access, with the 2% of accounts in the top-right corner potentially being the most impacting for solutions assuming locality (note that while we select an arbitrary reference point, conclusions are similar using other choices). To assess this impact, however, it is necessary to also study the *temporal* component of these accesses. A frequently accessed account with a long timespan may, indeed, have a different locality impact depending on whether it is accessed in bursts or regularly throughout its existence. We study these properties next.

Evolution of temporal locality metrics. We now focus on the *evolution* of metrics of interest for temporal locality over the different windows of the dataset. We report in Figure 6 the evolution of distributions for different metrics, as the evolution of quartiles for these metrics over subsequent windows.

First, we study the spread of accesses to older or newer data over time, i.e., whether transactions in each window tend to access accounts created in the recent or in the distant past. Figure 6a presents the evolution of the distribution of ages, for accounts accessed by transactions in each window. The age of an account in a window w_i ranges from 0, for an account created during w_i, and i for an account created in w_1. We observe that such accounts created in the first window w_1

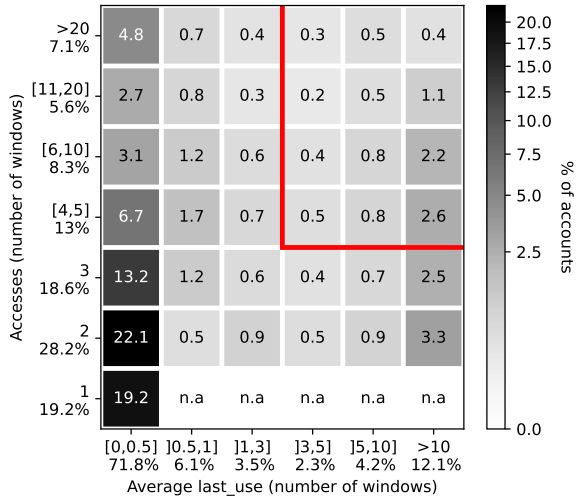


Fig. 7: Correlation between the number of windows in which each account a is used, i.e., $|\{i \mid a \in w_i(A)\}|$ and the average value of the a.last_use metric for these windows, i.e., $(\sum_{i \mid a \in w_i(A)} w_i(a.\text{last\_use}))/|\{i \mid a \in w_i(A)\}|$. Percentages with the x and y ticks represent the distribution of the sum of each column and row. The red lines demarcate the accounts we select as the *cache-unfriendly* (CUA) set.

continue to be used until the end of the dataset, explaining the linear progression for the maximum of the distribution. The median tends, however, to stabilize around about 100 windows, while the 75$^{\text{th}}$ percentile of the distribution (Q3) increases steadily, although sub-linearly. As a result, while half of the accesses are for relatively recent data, the spread of use in terms of creation time increases. This is not necessarily a sign of low temporal locality, as the age of an account does not reflect whether the account was accessed recently or not.

To study the temporal *proximity* of accesses to accounts across windows, we leverage the $w_i(a.\text{last\_use})$ metric, presented in Figure 6b. This metric represents the most recent use of the account outside of the considered window, unless this account is part of the 15% to 25% new accounts in this window (as discussed earlier in Figure 4b), in which case the metric value is 0. We observe that locality properties are clearly present in this case, with 75% (Q3) of accounts being accessed 20 to 30 windows ago, with no significant variations in time, and a median below 10 windows. The max remains i, the window identifier, in conformance with the max in Figure 6a, but accesses to very old data are only a minority.

Correlating account usage and temporal behavior. We finally study the correlation between the temporal behavior of accounts (i.e., how recently was an account previously accessed) and the number of windows in which these accounts are used. Figure 7 presents this correlation for different ranges of (1) the number of windows in which each account a is accessed (one or multiple times) and (2) the average value of the a.last_use over these windows (i.e., the average number of windows in which the account was previously used before being first used in some window), which we refer in the following as ALU. A high ALU relates to poor temporal locality of access, while the number of accesses correlates with the weight of each account in degrading locality.

Consistently with our previous observations, 19.2% of accounts are used in a single window. A majority of accounts (71.8% - 19.2% = 52.6%) accessed over 2 or more windows have excellent temporal locality, with an ALU between 0 and 0.5, i.e., mostly accesses repeated in adjacent windows. Overall, accounts with an ALU of less than 3, and therefore good temporal locality, represent 81.4% of all accounts. Accounts with an ALU greater than 10 represent 12.1% of the dataset. Of these, almost half (48% of the last column) have 2 or 3 accesses only, making their individual impact limited.

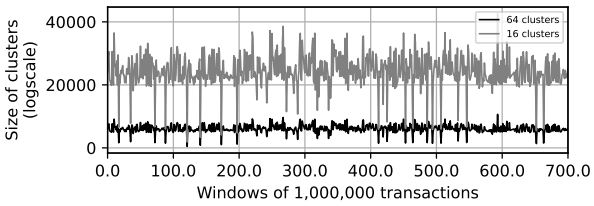
We calculate the Pearson correlation between the relative proportion of each cell in its column (ALU) on the one hand, and its relative position in its line (uses), respectively, ignoring the bottom column for single-use accounts. The obtained correlation factor is negative with -0.06, denoting a very small inverse correlation between the distribution of the two metrics, i.e., higher ALU tend to relate to a distribution shifting to lower number of uses, although only marginally.

Accounts that have poor temporal locality and more than a few uses are those that impact the most locality of accesses. If a system design for a blockchain employs a cache, these are the most likely to cause cache misses due to repeated but distant accesses. We select a subset of all accounts (10.3% of them) as the set of "*cache-unfriendly*" accounts, or CUA. Our intent is to compare the characteristics of these accounts with poor temporal locality properties with the characteristics of accounts impacting spatial locality properties that we study in the next section.

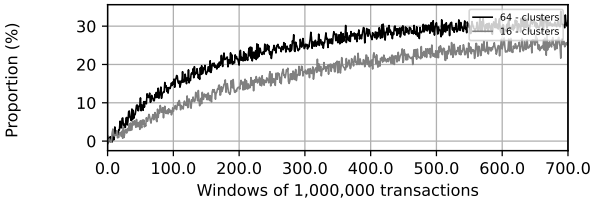
VI. Spatial Locality, Disjoint Access Parallelism

We study the spatial locality of the transaction workload. Spatial locality refers to the property that elements of the data set, accounts in our case, are co-accessed together by identifiable sets of transactions within a defined time period. A corollary of spatial locality is the exhibition by the dataset (or the lack thereof) of natural Disjoint Access Parallelism, or DAP for short. When, within a defined time frame, subsets of operations (transactions) access disjoint subsets of elements (accounts) without accessing accounts from the other subsets, then each such subset can be processed independently. This natural propensity to parallel execution can be exploited by blockchain designs to speed up the processing of transactions, either at the level of a single node through speculative execution [4], [5], [11], [14], [15] or at the level of multiple nodes by forming independent committees (shards) [24], [32] processing disjoint subsets of transactions.

The question we wish to answer is whether each window of transactions exhibits DAP. For this, we propose to use a clustering algorithm, specifically METIS [38]. This algorithm solves the k-way graph partitioning problem: it splits a graph $G = (V, E)$ of $|V| = n$ vertices into k subsets $V_1, V_2, \ldots, V_k$



(a) Size of partitions for $k = 16$ and $k = 64$.



(b) Proportion of transactions in $w_i(T)$ corresponding to *at least* one cross-partition edge for $k = 16$ and $k = 64$.

Fig. 8: Results of clustering each $w_i(IG)$ using METIS.

such that for any pair $(i, j) \mid i \neq j$, $V_i \cap V_j = \emptyset$. METIS aims to balance the size of partitions. In addition, it minimizes the sum of the weights of the edges from E that "cross" between different partitions.

In our model, IG is the interaction graph where edges' weights represent the number of transactions jointly accessing two accounts. Edges between partitions of the graph $w_i(IG)$ represent interdependencies between these partitions in this window w_i. A high number, or combined weights, of such edges indicates a less independent transaction workload (less DAP). In sharded blockchain designs [25], [28], [32], transactions accessing state from a single partition are called *intra-shard* transactions, while transactions accessing state from *at least* two partitions are called *cross-shard* transactions. Cross-shard transactions typically require costlier coordination between shard committees and impair parallelism and throughput. The DAP property of the workload is necessary, but not sufficient to ensure parallel execution by minimizing the number of cross-shard transactions. Indeed, clustering provides us with an upper bound of DAP based on a omniscient, complete knowledge: identifying *at runtime* independent subsets of transactions is a much more difficult problem [25], [28].

Clustering windows of transactions. We first evaluate whether METIS is able to partition each graph $w_i(IG)$ in equally-sized partitions (i.e., with each containing a similar number of vertices/accounts). We use two values of k, $k = 16$ and $k = 64$. Figure 8a confirms this with an average cluster size that is stable over different windows. In fact, the maximum difference we observe between the size of two clusters is never higher than 52 accounts (with $k = 16$), indicating excellent load balancing.

Figure 8b represents the evolution of the proportion of transactions involving at least one inter-partition edge. We observe that the number of such transactions has been constantly on the rise, independently of the value of k. This means that

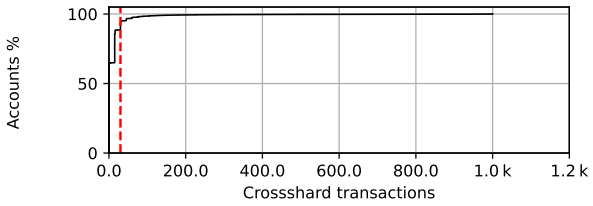


Fig. 9: CDF of the number of transactions requiring inter-partition accesses associated with each account. The vertical red line demarcates the selection point for parallelism-unfriendly accounts.

	Unit	Cache-unfriendly	Parallelism-unfriendly	Overlap
Accounts	Number	3184863	3184863	612759
Avg. lifetime	Windows	26.7	3.4	15.1
Avg. transactions	Transactions	7.5	12.4	8.7
Avg. frequency	Ratio	0.44	0.63	0.52
Avg. last use	Windows	27.1	1.2	7.3

TABLE I: Characteristics of CUA, PUA, and their overlap.

the DAP of the transactional workload has been degrading regularly: with 64 clusters, about 30% of transactions must access accounts across more than 1 partition. In other words, it is impossible for a perfect sharding scheme using omniscient knowledge of upcoming partitions (a very strong assumption that blockchain designs cannot make, as transactions are submitted by clients as a dynamic stream), to avoid that up to 30% of transactions require cross-shard coordination (25% with $k = 16$). This is consistent with observations made by the authors of OptChain [28].

Identifying parallelism-unfriendly accounts. We wish to identify if some accounts play a significant role in reducing DAP. Figure 9 presents the cumulative distribution of the number of transactions involving accesses across partitions, cumulated for each account (i.e., the sum of the weights of edges from an account a to any account in a different partition than a). We consider the entire graph IG. We can see that the impact of accounts on DAP is highly unbalanced. 64% of accounts are, in fact, never involved in inter-partition transactions. Out of these, we observe that 92% are involved in 5 transactions or less, hence being a majority of low-activity accounts. The remaining 8% of these accounts are involved in 20 transactions or less, with the notable exception of 146 accounts with high transaction volume, but always with accounts located in the same partition. 83% of accounts overall are involved in less than 10 inter-partition transactions. We observe that accounts with 6 to 10 such transactions are more long-lived accounts with a value of $w_i(a.\mathsf{last\_use})$ in the range [3,34]. The remaining 17% of accounts with 11 or more inter-partition transactions are those that impact the most DAP.

Comparing cache- and parallelism-unfriendly accounts. Our final observation is to compare the nature of accounts negatively impacting temporal locality (i.e., cache-unfriendly

accounts or CUA) and the nature of accounts negatively impacting spatial locality and DAP. We call the latter *parallelism-unfriendly* accounts or PUA. To compare sets of the same size, we select the top 10.3% of accounts with the most inter-partition transactions in the PUA set. Table I presents the characteristics of the two set, as well as their overlap, representing only 19.7% of the accounts. A first observation is, therefore, that the set of accounts impacting both types of locality are quite different. We also observe that CUA are more long-lived and accessed more sporadically than PUA, the latter having a higher frequency and a higher number of transactions overall. The identification of both types of accounts, for instance by a mechanism applying specific treatment (e.g., using a specific cache management policy aware of the existence of CUA, or using a sharding strategy that processes PUA separately) may need to identify them separately.

VII. Related work

We review the literature on blockchain measurements and group them by general topics.

Network Analysis. Multiple studies focused on the networking aspects of the Ethereum Network. They analyze the behavior of the network [26], communication patterns [23], or specificities of network peers [31]. Some papers use graph theory to analyze the evaluation of transaction patterns in the network [6], [12] or explore the characteristics of transactions and their impact on network performance [33]. In contrast, our paper focuses on the on-chain interaction between accounts through transactions without covering the network aspect.

Sharding and On-chain Data Analysis. Another group of work explores on-chain data exploration focusing on generic transactions [10] or smart contract interactions [21]. Hu *et al.* [19] use machine learning techniques to analyze transaction patterns and identify potential security risks. Ethereum transaction datasets have also been used to improve the allocation protocols in sharded designs [32] in terms of efficiency [16] or security [17]. Additionally, BlockSci [20] proposes a tool for analyzing blockchain data that can be used to gain insights into the behavior of Ethereum and other blockchains. Our study takes a broader approach, correlates temporal and spatial data, and provides insight useful for other types of improvement, such as light nodes.

VIII. Conclusion

We presented an analysis of the Ethereum transaction workload in light of its locality properties, both temporal and spatial. In the both cases, we identified that while most of the accounts and transactions exhibit good locality properties, a subset of accounts impair this locality significantly. The set of accounts is largely different from one locality type to the other. We believe our results could be a first step towards differentiated treatment of such accounts (and related transactions) in new blockchain designs, be it towards better decentralization and cheaper node bootstrap, or for efficient parallel processing. As we intend to release EthEx open source, we also hope that the tool will prove useful for blockchain scientists and practitioners to further study the characteristics of the massive Ethereum transaction dataset.

References

[1] Cardano. https://cardano.org/.
[2] The merge: Ethereum switch to proof-of-stake. https://ethereum.org/en/upgrades/merge/. Accessed on February 14, 2023.
[3] Official go implementation of the ethereum protocol. https://github.com/ethereum/go-ethereum.
[4] Amiri et al. Parblockchain: Leveraging transaction parallelism in permissioned blockchain systems. In *ICDCS '19*. IEEE.
[5] Anjana et al. Optsmart: a space efficient opt imistic concurrent execution of smart contracts. *Distributed and Parallel Databases*, 2022.
[6] Bai et al. Evolution of transaction pattern in ethereum: A temporal graph perspective. *IEEE Transactions on Computational Social Systems*, 2021.
[7] Busse et al. Evm-perf: High-precision evm performance analysis. In *ICBC '21*. IEEE.
[8] Cao et al. CoVer: Collaborative light-node-only verification and data availability for blockchains. In *Blockchain '20*. IEEE.
[9] Chatzigiannis et al. Sok: Blockchain light clients. In *FC '22*. Springer.
[10] Chen et al. Dataether: Data exploration framework for ethereum. ICDCS '19. IEEE.
[11] Chen et al. Forerunner: Constraint-based speculative transaction execution for ethereum. SOSP '21.
[12] Chen et al. Understanding ethereum via graph analysis. *ACM TOIT '20*.
[13] Denning et al. The working set model for program behavior. *Communications of the ACM*, 1968.
[14] Dickerson et al. Adding concurrency to smart contracts. In *PODC '17*.
[15] Gelashvili et al. Block-stm: Scaling blockchain execution by turning ordering curse to a performance blessing. In *PPOPP '23*.
[16] Han et al. Analysing and improving shard allocation protocols for sharded blockchains. Cryptology ePrint Archive, Paper 2020/943, 2020.
[17] Han et al. On the security and performance of blockchain sharding. Cryptology ePrint Archive, Report 2021/1276, 2021.
[18] Hasselgren et al. Blockchain in healthcare and health sciences—a scoping review. *International Journal of Medical Informatics*, 2020.
[19] Hu et al. Transaction-based classification and detection approach for ethereum smart contract. *Information Processing & Management*, 2021.
[20] Kalodner et al. BlockSci: Design and applications of a blockchain analysis platform. In *USENIX Security '20*.
[21] Kiffer et al. Analyzing ethereum's contract topology. IMC '18.
[22] Kim et al. Ethanos: Efficient bootstrapping for full nodes on account-based blockchain. EuroSys '21.
[23] Kim et al. Measuring ethereum network peers. IMC '18.
[24] Korkmaz et al. ALDER: Unlocking blockchain performance by multiplexing consensus protocols. NCA '22.
[25] Król et al. Shard scheduler: object placement and migration in sharded account-based blockchains. AFT '21.
[26] Li et al. Ethereum behavior analysis with netflow data. In *APNOMS '19*.
[27] Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, 2008.
[28] Nguyen et al. Optchain: optimal transactions placement for scalable blockchain sharding. ICDCS '19. IEEE.
[29] Paavolainen et al. Adventures of a light blockchain protocol in a forest of transactions: A subset of a story. *IEEE Access*, 2021.
[30] Rana et al. Free2Shard: Adversary-resistant distributed resource allocation for blockchains. *POMACS "22*.
[31] Victor et al. Measuring ethereum-based ERC20 token networks. In *Financial Cryptography and Data Security*, 2019.
[32] Wang et al. SOK: Sharding on blockchain. AFT '19.
[33] Wang et al. Exploring ethereum nodes and transactions. 2019.
[34] Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 2014.
[35] Yakovenko et al. Solana: A new architecture for a high performance blockchain v0. 8.13. *Whitepaper*, 2018.
[36] Zheng et al. Xblock-eth: Extracting and exploring blockchain data from ethereum. *IEEE Open Journal of the Computer Society*, 2020.
[37] Robert Fynn et al. Robust and fast blockchain state synchronization. In *OPODIS '22*.
[38] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.