


Transaction Fee Market Design for Parallel Execution

Bahar Acilan¹ ✉ 

ETH Zürich, Switzerland

Andrei Constantinescu ✉ 

ETH Zürich, Switzerland

Lioba Heimbach ✉ 

ETH Zürich, Switzerland

Roger Wattenhofer ✉ 

ETH Zürich, Switzerland

Abstract

Given the low throughput of blockchains like Bitcoin and Ethereum, scalability – the ability to process an increasing number of transactions – has become a central focus of blockchain research. One promising approach is the parallelization of transaction execution across multiple threads. However, achieving efficient parallelization requires a redesign of the incentive structure within the fee market. Currently, the fee market does not differentiate between transactions that access multiple high-demand storage keys (i.e., unique identifiers for individual data entries) versus a single low-demand one, as long as they require the same computational effort. Addressing this discrepancy is crucial for enabling more effective parallel execution.

In this work, we aim to bridge the gap between the current fee market and the need for parallel execution by exploring alternative fee market designs. To this end, we propose a framework consisting of two key components: a *Gas Computation Mechanism (GCM)*, which quantifies the load a transaction places on the network in terms of parallelization and computation, measured in *units of gas*, and a *Transaction Fee Mechanism (TFM)*, which assigns a price to each unit of gas. We additionally introduce a set of desirable properties for a GCM, propose several candidate mechanisms, and evaluate them against these criteria. Our analysis highlights two strong candidates: the *weighted area GCM*, which integrates smoothly with existing TFMs such as EIP-1559 and satisfies a broad subset of the outlined properties, and the *time-proportional makespan GCM*, which assigns gas costs based on the context of the entire block’s schedule and, through this dependence on the overall execution outcome, captures the dynamics of parallel execution more accurately.

2012 ACM Subject Classification Applied computing → Economics

Keywords and phrases blockchain, transaction fee mechanism, parallel execution

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.23

Related Version *Full Version*: <https://arxiv.org/abs/2502.11964> [1]

Funding We thank the Robust Incentives Group at the Ethereum Foundation for supporting this work through a grant.

Acknowledgements We thank Boyan Zhou for his work on a student project, supervised by the last three authors, which helped shape this work.

¹ The authors of this work are listed alphabetically.



1 Introduction

Scalability, the ability to process more transactions efficiently, has become a central focus in blockchain research, especially given the low throughput of many existing networks. Ethereum, for example, is constrained by its single-threaded execution model, limiting transaction throughput. One promising way to enhance scalability is by parallelizing transaction execution across multiple threads, taking advantage of the multi-core processors common in modern hardware. However, achieving the full efficiency gains of parallel execution requires rethinking the fee market design to better account for *storage key* contention and scheduling constraints. A storage key is a unique identifier (like an address label) for a specific data item in storage.

A *transaction fee mechanism (TFM)* is a core component of any blockchain protocol, determining which pending transactions should be processed and what users must pay for the privilege of having their transactions executed. Traditional fee mechanisms, like Bitcoin’s first-price auction, involve users submitting bids with their transactions, and the transactions with the highest bids per computation are included in the next block. Ethereum initially used a similar purely first-price auction-based model but switched to the more sophisticated EIP-1559 mechanism in 2021, which introduces a fluctuating base fee based on network demand, aiming to improve incentive compatibility and reduce price volatility [12]. Most of the existing literature on transaction fee mechanisms focuses on settings where transactions are executed sequentially and therefore does not account for storage key contention, which is crucial in parallel execution environments.

Thus, viewed in isolation, these traditional TFMs are ill-suited to the complexities introduced by parallel transaction execution. They price transactions solely based on their computational cost, without distinguishing between those that access a single storage key and those that interact with multiple, potentially contested storage keys. This pricing model works in a single-threaded environment but fails to capture the nuances of parallel execution, where transactions may impose vastly different constraints on storage key scheduling. Transactions that touch multiple high-contention storage keys can introduce bottlenecks, while those interacting with isolated storage keys are far easier to schedule efficiently.

Ethereum has yet to adopt parallel execution, but several blockchains, such as Solana, Aptos, and Sui, already employ parallel transaction processing [5, 43, 57, 60, 46]. However, many of these networks have yet to implement fee models that fully account for the challenges of parallel execution. Sui and Solana have introduced fee markets tailored to parallelization, but these mechanisms require users to engage in first-price auctions for congested storage keys [40, 44]. As a result, these fee markets demand a high level of sophistication from users to effectively optimize their fee settings and are also not incentive-compatible. The requirements for an effective fee market that is suitable for parallel execution and the design of such a market have so far remained unresolved.

Our Contributions

In this work, we aim to bridge this gap by outlining the requirements for such a fee market and evaluating possible candidates. We outline our main contributions below:

- We introduce a framework with two main components: a *Gas Computation Mechanism (GCM)*, which measures the load a transaction imposes on the network in terms of both parallel execution and computation, expressed in *units of gas*, and a Transaction Fee Mechanism (TFM), which determines the cost associated with each unit of gas.
- We introduce a list of desirable properties for a GCM and evaluate against them a set of mechanisms that we propose.

- Our analysis identifies two promising candidates. The weighted area GCM achieves a large subset of the outlined properties and, importantly, can be seamlessly integrated with existing TFMs, inheriting their properties. Complementing it, the time-proportional makespan GCM is designed to price execution costs in proportion to the total computational load of a block, allowing more accurate resource pricing at the block level.

2 Model

We consider a universe consisting of several stateful *storage keys* (e.g., user accounts, storage addresses of smart contracts). Each storage key can be thought of as a system global variable. We write \mathcal{K} for the set of storage keys. For analysis purposes, we assume that \mathcal{K} is infinite. A *transaction* is a sequence of elementary instructions performing computation and interacting with the storage keys. Some of these operations *access* a given target storage key (e.g., read its value, write to it). For simplicity, we assume that the following are known in advance and supplied with the transaction:

- The non-empty set of storage keys $K \subseteq \mathcal{K}$ the transaction accesses, or an overestimate.²
- The total time $t > 0$ it takes to execute the transaction on a single thread.³

Transactions execute concurrently, but *atomically*, meaning that the overall effect of executing a batch of transactions should also be achievable by a sequential, single-core execution. For simplicity, we restrict to *concurrent schedules* following a *simple lock-based execution policy*: each storage key has a lock associated with it; whenever a thread wants to execute a transaction, it first locks all required storage keys, then executes the transaction, and then releases the locks. We assume that acquiring (and similarly releasing) the required locks happens simultaneously and takes no additional time. These simplifications have a desirable side-effect: t and K for each transaction now uniquely determine the set of admissible concurrent schedules, allowing us to ignore other details about the transactions:

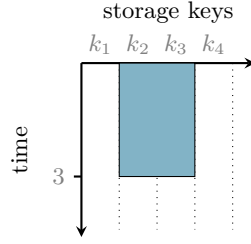
► **Definition 1 (Transaction).** A transaction tx is specified through a tuple (t, K) , where $t > 0$ denotes the time required to execute the transaction and $K \neq \emptyset$ represents the set of storage keys the transaction demands, which are locked for the duration of the transaction. We will sometimes write $t(tx)$ and $K(tx)$ for t and K respectively.

To give the tuple associated with a transaction, we will write $tx \simeq (t, R)$. Note that different transactions might have the same associated tuple. We will also write $tx_1 \simeq tx_2$ to mean that the two transactions have the same associated tuple.

In Figure 1, we illustrate a sample transaction $tx \simeq (3, \{k_2, k_3\})$. Transaction tx thus takes 3 units of time to execute and utilizes storage keys k_2 and k_3 . We illustrate this by a rectangle of corresponding length (i.e., time) and width (i.e., storage keys). Throughout, we will illustrate transactions in this manner to aid in visualizing concepts and results. Note that we will always represent transactions as rectangles (i.e., they use consecutive storage keys). In reality, this is, of course, not the case. Importantly, all our results also hold in the more general setting where a transaction can use any subset of storage keys.

² In Ethereum, transaction accesses are generally not known in advance. Instead, transactions can execute arbitrary logic (constrained by a maximum amount of computational effort) and their execution depends on the blockchain's state at the time of execution. Ethereum currently supports optional *access lists* that allow transactions to specify their accesses [14]. This optional list could be made mandatory to provide an overestimate of accesses. Additionally, it is worth noting that in other blockchains (e.g., Solana [36] and Sui [35]), an overestimate of accesses is typically known in advance.

³ In reality, the time to execute a transaction depends on the hardware it is executed on. Thus, time is estimated in Ethereum by assigning each operation a computational effort. Then the sum of the computational effort of a transaction's operation can be seen as a proxy for time.



■ **Figure 1** Illustration of a sample transaction $tx \simeq (3, \{k_2, k_3\})$.

A *blockchain* is a sequence of *blocks* comprising bundles of transactions: B_1, \dots, B_m . The system starts in some predetermined initial state. The blocks are executed in order, starting from the oldest (the *genesis* block B_1), successively changing the system's state. For simplicity, we assume no cross-block parallelism, so the execution of a block only starts after the previous block's execution has been completed. However, the execution of transactions inside a block happens concurrently. The system's state after executing B_m is the *current* system state. Users wanting to change the system's state compete for inclusion in the next block B_{m+1} and have to pay a *fee* if successfully included. Desirably, the fee should be higher for more complex transactions and higher during periods of high demand due to limited block space. These requirements are typically decoupled and ensured through different means:

1. A transaction's complexity is quantified in units of *gas*:⁴ the more complex a transaction is, the more gas it consumes. Gas encompasses multiple components such as execution, storage space, and data bandwidth. For our purposes, we will only be concerned with *execution gas*.⁵ Currently, the execution gas acts as a proxy for the execution time t ,⁶ but this need not be the case, as we will demonstrate in our paper.
2. The fair competition for block space is ensured through a *transaction fee mechanism (TFM)*: users submit transactions they would like to be included in the next block together with bids of what they would be willing to pay per unit of gas. Importantly, block space is limited, i.e., there is limited space for transactions. The mechanism then determines the set⁷ of transactions to be included in the block, together with a price per unit of gas to be paid by each included transaction (potentially not the same for all transactions).⁸

As such, an included transaction consuming g gas units at a price of p per unit of gas will have to pay a fee of $g \cdot p$ (generally in the blockchain's native currency).

To keep the separation of concerns in (1) and (2), we will keep the formula for the fee $g \cdot p$ and instead vary the *gas computation mechanism* used to determine g :

⁴ For Ethereum, the unit is called wei.

⁵ From this point forward, gas will refer specifically to execution gas.

⁶ Throughout we will assume this approximation to the exact.

⁷ In practice, blocks are ordered, and transactions often compete for earlier positions (particularly in MEV settings). Following a common simplification in the TFM literature, we model blocks as *sets* of transactions. Accounting for intra-block ordering would require extending our model and constraining it to schedulers that honor the specified ordering. Our framework appears reasonably extensible to such settings, and we leave a detailed treatment to future work.

⁸ There are several other components of a TFM, but for the level of detail we need here, this suffices.

► **Definition 2** (Gas Computation Mechanism). *A gas computation mechanism (GCM) takes as input a set of transactions T to be included in a block and determines in a deterministic⁹ manner the amount of gas consumed by each transaction $tx \in T$, written $gas_T(tx)$.*

Currently deployed GCMs associate a fixed, predetermined gas consumption with each transaction, independent of the specific storage keys accessed by the transaction, making them unsuitable for a parallel execution environment. In particular, this is true for Ethereum's current GCM:

► **Definition 3** (Current GCM). *Given a set of transactions T and a transaction $tx \in T$ with $tx \simeq (t, R)$, the current GCM computes the amount of gas used by tx as follows:*

$$gas_T^C(tx) := t.$$

Since this does not depend on T , we often drop the subscript.

Our goal is to ensure that fees accurately reflect the parallelizability of transactions. Therefore, the gas consumption calculated for a transaction should depend on the set of storage keys it accesses and may also be influenced by factors external to the transaction itself (like interactions between transactions).

3 GCM Properties

Next, we outline several desirable properties that a GCM should possess to provide the right incentives for parallelization. These properties should be viewed as a wishlist – as will become clear later on, no single mechanism can satisfy all of them simultaneously.

We begin with two natural monotonicity properties, one for storage keys and one for time. First, a transaction that requires a subset of the storage keys used by another transaction, while taking the same amount of time, should consume no more gas (Property 1 and Figure 2). Similarly, a transaction that requires no more execution time than another, assuming both involve the same set of storage keys, should consume no more gas (Property 2 and Figure 3).

► **Property 1** (Storage Key Monotonicity). *Given a set of transactions T and two transactions $tx_1 \simeq (t_1, K_1)$ and $tx_2 \simeq (t_2, K_2)$, both not in T , such that $t_1 = t_2$ and $K_1 \subseteq K_2$:*

$$gas_{T \cup \{tx_1\}}(tx_1) \leq gas_{T \cup \{tx_2\}}(tx_2).$$

► **Property 2** (Time Monotonicity). *Given a set of transactions T and two transactions $tx_1 \simeq (t_1, K_1)$ and $tx_2 \simeq (t_2, K_2)$, both not in T , such that $t_1 \leq t_2$ and $K_1 = K_2$:*

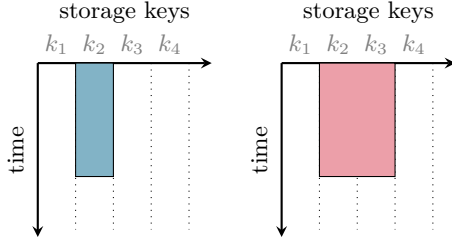
$$gas_{T \cup \{tx_1\}}(tx_1) \leq gas_{T \cup \{tx_2\}}(tx_2).$$

The previous two properties fix one dimension while varying the other. One can also define a seemingly stronger property that allows both to vary, as follows:

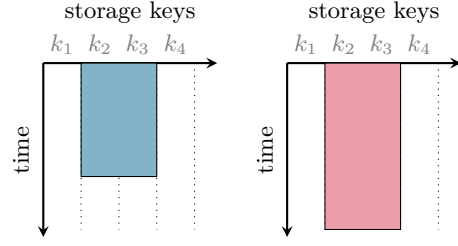
► **Property 3** (Storage Key-Time Monotonicity). *Given a set of transactions T and two transactions $tx_1 \simeq (t_1, K_1)$ and $tx_2 \simeq (t_2, K_2)$, both not in T , such that $t_1 \leq t_2$ and $K_1 \subseteq K_2$:*

$$gas_{T \cup \{tx_1\}}(tx_1) \leq gas_{T \cup \{tx_2\}}(tx_2).$$

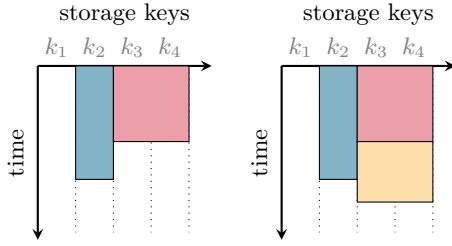
⁹ We focus solely on deterministic mechanisms. This is common in blockchain fee markets, given the inherent difficulty of accessing true randomness on-chain (though notable approaches exist, such as VRFs). While randomized mechanisms can statistically improve over their deterministic counterparts, their ex-post behavior is often hard to predict, leading to higher fee uncertainty, an outcome generally undesirable in blockchain settings.



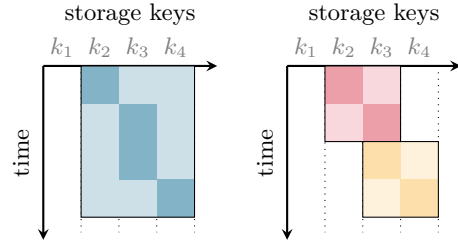
■ **Figure 2** Illustration of Property 1, where $T = \emptyset$, ■ represents a sample tx_1 and ■ represents a sample tx_2 .



■ **Figure 3** Illustration of Property 2, where $T = \emptyset$, ■ represents a sample tx_1 and ■ represents a sample tx_2 .



■ **Figure 4** Illustration of Property 4, where $T = \emptyset$, $T_1 = \{tx_1, tx_2\}$, and $T_2 = T_1 \cup \{tx_3\}$. Here, ■ represents a sample tx_1 , ■ represents a sample tx_2 , and ■ represents a sample tx_3 .



■ **Figure 5** Illustration of Property 5, where $T = \emptyset$, ■ represents a sample tx_3 , while ■ represents a sample tx_1 and ■ represents a sample tx_2 . The darker shaded areas indicate when a transaction operates on a storage key.

Intuitively, if $tx_1 \lesssim tx_2$, by which we mean $t(tx_1) \leq t(tx_2)$ and $K(tx_1) \subseteq K(tx_2)$, then tx_1 should cost no less than tx_2 . However, an attentive reader will observe that the former two are collectively equivalent to the latter (the proof and all subsequent omitted proofs can be found in the full version of this paper [1]):

► **Lemma 4.** *Property 3 holds if and only if Properties 1 and 2 hold.*

Let us now take a moment to briefly evaluate why Properties 1–3 are not merely intuitive, but their violation can lead to harmful consequences and misaligned incentives: suppose $tx_1 \lesssim tx_2$ and T is a set of transactions containing neither of the two. If $\text{gas}_{T \cup \{tx_1\}}(tx_1) > \text{gas}_{T \cup \{tx_2\}}(tx_2)$, a user intending to submit tx_1 might instead pad tx_1 with unnecessary instructions and declare a larger access list to decrease the gas consumption. This would paradoxically reduce the gas usage and, assuming a reasonable TFM is used to compute transaction fees, also lower the transaction fee.

For any of the three properties above, we say that the property is *strictly* satisfied if for $tx_1 \not\sim tx_2$ the conclusion inequality holds strictly. Note that properties holding strictly are even more desirable with respect to the reasoning above: replacing a transaction with a “larger” one is then not only no better but actively worse. Unsurprisingly, Lemma 4 also holds for the strict versions of the properties:

► **Lemma 5.** *Property 3 holds strictly if and only if Properties 1 and 2 hold strictly.*

Next, we introduce another desirable monotonicity property, this time with a different emphasis: if two sets of transactions satisfy $T_1 \subseteq T_2$, the transactions in T_1 should collectively consume no more gas than those in T_2 (Property 4 and Figure 4). To formalize this, given a set of transactions T and a subset $T' \subseteq T$, write $\text{gas}_T(T') := \sum_{tx' \in T'} \text{gas}_T(tx')$ for the total gas consumed by the transactions in T' when included in the set T that constitutes a block.

► **Property 4** (Set Inclusion). *Given a set of transactions T and two sets of transactions $T_1 \subseteq T_2$, disjoint from T :*

$$\text{gas}_{T \cup T_1}(T_1) \leq \text{gas}_{T \cup T_2}(T_2).$$

To understand why this property is desirable, consider a GCM for which it does not hold. Then, there must be a scenario where a set of transactions can reduce their total gas consumption by adding additional transactions. This situation could be exploited through collusion by the users originating these transactions. Importantly, such a possibility is undesirable, as it would primarily benefit sophisticated users capable of orchestrating such arrangements.

Similarly to before, we say that Property 4 holds *strictly* if for $T_1 \neq T_2$ the inequality in the conclusion holds strictly, which is desirable for reasons similar to those discussed above.

We now move on to more complex properties. Given two transactions $tx_1 \simeq (t_1, K_1)$ and $tx_2 \simeq (t_2, K_2)$, a transaction tx_3 is their *sequential composition* (or, more simply, *concatenation*), if it executes the steps of tx_1 followed by the steps of tx_2 . Note that, in this case $tx_3 \simeq (t_1 + t_2, K_1 \cup K_2)$. Our next property states that the concatenation of two transactions should consume no less gas than submitting them individually (Property 5 and Figure 5). The two individual transactions perform the same actions as their concatenation, but not atomically, with no guarantee of their relative ordering or control over what happens between them. Naturally, enforcing atomicity and ordering limits the set of admissible concurrent schedules and requires storage keys to remain locked over longer continuous timespans. In particular, tx_3 requires the storage keys in $K_1 \cup K_2$ to be locked over a continuous span of $t_1 + t_2$ units, while tx_1 and tx_2 submitted individually only require exclusive access to storage keys in K_1 for t_1 units and to storage keys in K_2 for t_2 units. Hence, the “bigger” transaction is at least as hard to schedule as its two constituent “parts” and should hence consume no less gas.

► **Property 5** (Transaction Bundling). *Consider a set of transactions T and three transactions $tx_1, tx_2, tx_3 \notin T$ such that tx_3 is the concatenation of tx_1 and tx_2 , then:*

$$\text{gas}_{T \cup \{tx_1, tx_2\}}(tx_1) + \text{gas}_{T \cup \{tx_1, tx_2\}}(tx_2) \leq \text{gas}_{T \cup \{tx_3\}}(tx_3).$$

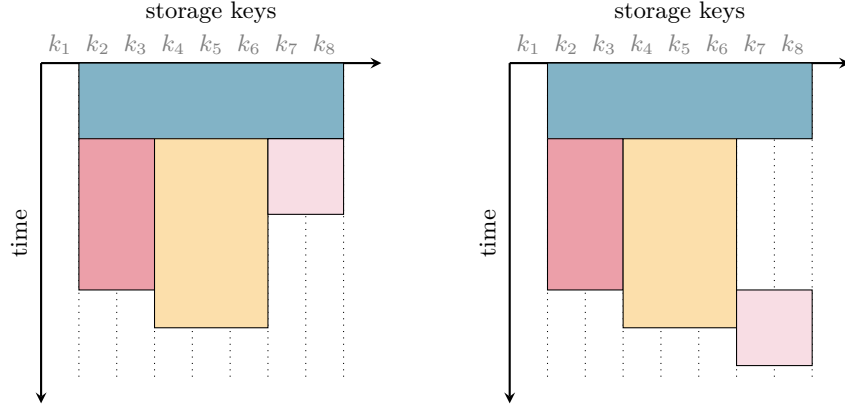
Let us again consider the risks of having a GCM that does not satisfy the previous property. In such a scenario, a group of users could collude to collectively consume less gas by combining their transactions into a single transaction rather than processing them individually. This outcome would be undesirable, particularly because it disproportionately benefits sophisticated users, as we outlined before. Note that this could even be the case for a single user wanting to submit multiple transactions.

We say that Property 5 holds *strictly* if the inequality in the conclusion holds strictly, which is again more desirable than the basic version of the property.

Next, we would like to formalize the intuitive idea that a transaction’s gas consumption fairly reflects its impact on the execution time. To do so, we first need to formalize scheduling more precisely. Let $n \geq 2$ be the number of available threads.

► **Definition 6.** *A scheduler (for n threads) takes as input a set of transactions T , and outputs a concurrent schedule using at most n threads to execute all transactions in T . This schedule specifies the operations each thread should perform and the order in which they should be performed.*

The following conditions should hold for any generated schedule:



■ **Figure 6** Illustration of optimal schedules for a set of four transactions: $tx_1 \simeq (2, \{k_2, \dots, k_8\})$ (shown in ■), $tx_2 \simeq (4, \{k_2, k_3\})$ (shown in ■), $tx_3 \simeq (5, \{k_4, k_5, k_6\})$ (shown in ■), and $tx_4 \simeq (2, \{k_7, k_8\})$ (shown in ■). In the left plot, we show the optimal schedule for $n = 3$, and in the right plot for $n = 2$. Notice how for $n = 2$, we cannot schedule the three transactions tx_2 , tx_3 , and tx_4 to be executed in parallel even though they access pairwise-disjoint sets of storage keys.

- Each transaction is assigned to a single thread;
- A thread can work on only one transaction at a time;
- There is no preemption: once a thread starts executing a transaction, it completes the transaction without context switching;
- Transactions with overlapping storage key access sets cannot be executed in parallel: one must finish before the other can begin.

Note that some of these conditions are natural for scheduling in general, while others arise from us assuming the *simple lock-based execution policy*.¹⁰ For our purposes, we are not concerned with which thread executes which transaction, but only that no more than n transactions ever execute simultaneously (so we can draw concurrent schedules as in Figure 6, which illustrates the concept). To determine gas consumption, our notation will need to capture even less – given a scheduler s (for n threads), we write $v^s(T)$ for the *makespan* of the schedule produced by s for the set of transactions T (i.e., the time required to execute the schedule in parallel using n threads). It is instructive to read the paper having in mind as a prime example s being the *optimal scheduler* (for n threads), which returns a schedule for n threads that minimizes the makespan. However, implementing such a scheduler is computationally prohibitive in practice, so greedy heuristics are typically used instead. To make our results apply even to non-optimal schedulers, we assume a set of minimal, reasonable properties for the scheduler:

- (S1) *Monotonicity in T* : for any $T \subseteq T'$, we have $v^s(T) \leq v^s(T')$. Intuitively, scheduling no fewer transactions takes no less time.
- (S2) *Monotonicity under bundling*: consider any set of transactions T and three transactions $tx_1, tx_2, tx_3 \notin T$, such that tx_3 is the concatenation of tx_1 and tx_2 , then we have $v^s(T \cup \{tx_1, tx_2\}) \leq v^s(T \cup \{tx_3\})$. Intuitively, replacing two transactions by their concatenation makes scheduling no easier.

¹⁰Using locks is one way to enforce this policy, but in our case – where the contents of a block are known before execution commences – it can also be achieved without locks, as long as the execution environment ensures that threads strictly follow a pre-determined schedule. We chose this name because it is largely suggestive of the intended semantics.

(S3) *Monotonicity in t and K* : consider any set of transactions T and two transactions $tx_1 \simeq (t_1, K_1)$ and $tx_2 \simeq (t_2, K_2)$, both not in T , such that $t_1 \leq t_2$ and $K_1 \subseteq K_2$, i.e., $tx_1 \lesssim tx_2$, then we have $v(T \cup \{tx_1\}) \leq v(T \cup \{tx_2\})$. Intuitively, “larger” transactions are no easier to schedule.

(S4) *Empty set*: $v(\emptyset) = 0$.

The optimal scheduler can be easily seen to satisfy these properties.¹¹ Our positive results will apply to any scheduler satisfying the properties, while our negative results will be for the optimal scheduler itself. Henceforth, we drop the s superscript for brevity and follow this convention for who s is.

An attentive reader may notice that (S1)–(S3) partially resemble Properties 1–5 for GCMs. This resemblance is not coincidental, but it is important to emphasize that they address different aspects: the former are properties of the scheduler, while the latter are properties of the GCM. A GCM can, in fact, satisfy Properties 1–5 without depending on the scheduler at all (e.g., the current mechanism). Conversely, for mechanisms defined in terms of the scheduler, (S1)–(S3) play a crucial role in establishing their properties, including Properties 1–5.

Armed as such, we now return to our latest goal: formalizing the idea that a transaction’s gas consumption should fairly reflect its impact on execution time. We do this as follows:

► **Property 6 (Scheduling Monotonicity)**. *Given a set of transactions T and two transactions tx_1 and tx_2 , both not in T , such that $v(T \cup \{tx_1\}) < v(T \cup \{tx_2\})$.*¹²

$$\text{gas}_{T \cup \{tx_1\}}(tx_1) \leq \text{gas}_{T \cup \{tx_2\}}(tx_2).$$

Intuitively, transactions with higher marginal contributions to the execution time should consume no less gas.¹³ As usual, we define a *strict* version of this property, where the latter inequality also becomes strict (i.e., higher marginal contributions in execution time imply higher gas consumption). One might be tempted to believe that Scheduling Monotonicity implies Storage Key-Time Monotonicity. However, proving this requires a non-strict inequality in the premise $v(T \cup \{tx_1\}) < v(T \cup \{tx_2\})$.

There is a second way in which gas consumption should adequately indicate the effort required to execute transactions (Property 7). Specifically, the gas consumption of all transactions in a block should *collectively* account for the total time needed to execute the block. This can be seen as properly tracking the time consumed by the execution environment to execute the block.

► **Property 7 (Efficiency)**. *Consider a set of transactions T and recall the definition $\text{gas}_T(T) = \sum_{tx \in T} \text{gas}_T(tx)$, then:*

$$\text{gas}_T(T) = v(T).$$

¹¹ All but the second straightforward, while for the second, any admissible schedule for $T \cup \{tx_3\}$ can be turned into a same-makespan admissible schedule for $T \cup \{tx_1, tx_2\}$ by replacing tx_3 by tx_1 followed immediately by tx_2 .

¹² Interestingly, unlike our earlier properties, writing $v(T \cup \{tx_1\}) \leq v(T \cup \{tx_2\})$ here may be too strong, as applying the property twice would then imply that if $v(T \cup \{tx_1\}) = v(T \cup \{tx_2\})$, then $\text{gas}_{T \cup \{tx_1\}}(tx_1) = \text{gas}_{T \cup \{tx_2\}}(tx_2)$, which is not necessarily desirable. Writing $v(T \cup \{tx_1\}) \leq v(T \cup \{tx_2\})$ would also unintentionally make the strict and non-strict versions of the property incomparable.

¹³ Technically, what we wrote above in Property 6 are not marginal contributions, but can be made to be by subtracting $v(T)$ from both sides of the inequality in the antecedent.

Next, we introduce two practical properties. The first requires that transaction submitters be able to estimate a transaction's gas consumption in advance. We formalize this by requiring that $\text{gas}_T(tx)$ does not depend on T (Property 8).

► **Property 8** (Easy Gas Estimation). *Given two sets of transactions T_1 and T_2 and a transaction tx belonging to neither T_1 nor T_2 :*

$$\text{gas}_{T_1 \cup \{tx\}}(tx) = \text{gas}_{T_2 \cup \{tx\}}(tx).$$

This property ensures a good user experience by making it easy for users to estimate gas usage. If the gas consumption of a transaction depended on the remaining transactions in the block, this estimation would become significantly more complex. In general, we aim to keep gas estimation straightforward to avoid giving an advantage to more sophisticated users. Additionally, as we will see later, a GCM satisfying this property can be seamlessly composed with existing TFMs, retaining their desirable properties (see Section 6). Sadly, as one might have already guessed, Easy Gas Estimation is incompatible with Scheduling Monotonicity (except for *constant* mechanisms, i.e., GCMs that return a constant independent of T and tx), and also incompatible with Efficiency:

► **Theorem 7.** *Easy Gas Estimation (Property 8) is:*

1. *Incompatible with Scheduling Monotonicity (Property 6) unless using a constant GCM.*¹⁴
2. *Incompatible with Efficiency (Property 7).*

Proof. Consider a mechanism M with Easy Gas Estimation; i.e., $\text{gas}^M(tx)$ is meaningful without a subscript for the set of transactions T in the block. Then:

1. Assume that M has Scheduling Monotonicity; we will show that M is a constant mechanism.

Call two transactions $tx_1 \simeq (t_1, K_1)$ and $tx_2 \simeq (t_2, K_2)$ *incomparable* if neither $K_1 \subseteq K_2$ nor $K_2 \subseteq K_1$. As a first step, we will show that if tx_1 and tx_2 are incomparable, then $\text{gas}^M(tx_1) = \text{gas}^M(tx_2)$. To show this, assume that $K_1 \not\subseteq K_2$. We will show that then $\text{gas}^M(tx_1) \geq \text{gas}^M(tx_2)$. Exchanging the roles of tx_1 and tx_2 will then give the conclusion. Let $k \in K_1 \setminus K_2$ and tx_3 be another transaction such that $tx_3 \simeq (t_3, \{k\})$ for some $t_3 > t_2$, and define $T = \{tx_3\}$. Then, for any number of threads $n \geq 2$ and any scheduler that is optimal for two transactions, we have $v(T \cup \{tx_1\}) = t_1 + t_3 > t_3 = v(T \cup \{tx_2\})$. By Scheduling Monotonicity, this implies that $\text{gas}^M(tx_1) \geq \text{gas}^M(tx_2)$.

We now know that M associates the same gas consumption to any pair of incomparable transactions. Armed as such, consider two arbitrary transactions $tx_1 \simeq (t_1, K_1)$ and $tx_2 \simeq (t_2, K_2)$. Let k be an arbitrary storage key *not* in $K_1 \cup K_2$ and tx_3 be a transaction such that $tx_3 \simeq (1, \{k\})$. Since K_1 and K_2 are non-empty, one can easily see that tx_3 is incomparable with both tx_1 and tx_2 , from which $\text{gas}^M(tx_1) = \text{gas}^M(tx_3) = \text{gas}^M(tx_2)$.

2. Assume for a contradiction that M has Efficiency, then for any transaction $tx \simeq (t, K)$ we have $\text{gas}^M(tx) = \text{gas}^M_{\emptyset}(\{tx\}) = v(\{tx\}) = t$. Hence, by definition, M is the current mechanism, which can be easily seen to not satisfy Efficiency: assume $n \geq 2$ threads and consider the set of transactions $T = \{tx_1, tx_2\}$ where $tx_1 \simeq (1, \{k_1\})$ and $tx_2 \simeq (1, \{k_2\})$. In this case, $\text{gas}^M(T) = \text{gas}^M(tx_1) + \text{gas}^M(tx_2) = 1 + 1 = 2 \neq 1 = v(T)$. ◀

Our second practical property requires that gas consumption be efficiently computable, i.e., in polynomial time (Property 9). A GCM that does not satisfy this property would be unsuitable for the blockchain context, where gas computation is intended to be a straightforward component.

¹⁴ Constant GCMs satisfy Scheduling Monotonicity, but not Strict Scheduling Monotonicity.

► **Property 9** (Poly-time Computable). *There exists a polynomial-time algorithm that takes as input a transaction set T and a transaction tx and outputs $\text{gas}_{T \cup \{tx\}}(tx)$.*

4 GCM Proposals

We now propose multiple GCM designs. Motivated by the incompatibilities in Theorem 7, these fall into two categories:

- (C1) Mechanisms with Easy Gas Estimation, in which each transaction’s gas consumption is computed in isolation. Such mechanisms tend to be both straightforward and attractive but can achieve neither Scheduling Monotonicity¹⁵ nor Efficiency.
- (C2) Mechanisms *without* Easy Gas Estimation, which, given a set of transactions T , rely on $v(T)$ or, more generally, $(v(T'))_{T' \subseteq T}$, to holistically calculate gas consumptions for the block T , requiring knowledge of the entire block. Instead, these mechanisms aim to achieve Scheduling Monotonicity and/or Efficiency.

4.1 Mechanisms with Easy Gas Estimation

We already introduced the *current* GCM (Definition 3). For the next mechanism, we assume a globally available vector of *positive* weights for the storage keys $(w_k)_{k \in K}$. For instance, these weights could all be 1. Alternatively, higher-weight storage keys could correspond to storage keys under higher (historical) demand. For our purposes, we only need to assume that the weights are known and do not depend on the block being built. In practice, one could update the weights between blocks to accurately reflect storage key demand (the exact details are not relevant here; see Section 6 for further discussion). Given the weights, we define:

► **Definition 8** (Weighted Area GCM). *Given a set of transactions T and a transaction $tx \in T$ with $tx \simeq (t, K)$, the Weighted Area GCM computes the amount of gas used by tx as follows:*

$$\text{gas}_T^{WA}(tx) := t \cdot \left(1 + \sum_{k \in K} w_k \right) \quad (1)$$

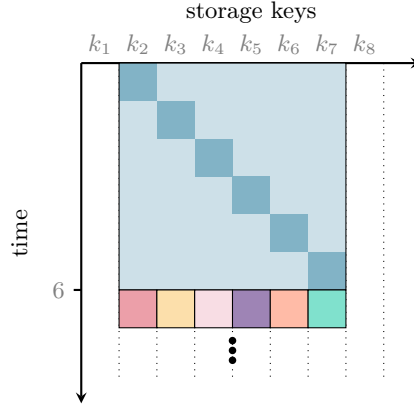
Since this does not depend on T , we often drop the subscript.

To gain intuition, it is instructive to consider the former case, where all weights are 1, i.e., the *unweighted area* mechanism, in which case Equation (1) becomes $\text{gas}_T^{WA}(tx) = t \cdot (1 + |K|)$. This mechanism can also be viewed as the addition of two terms: the *current term*, i.e., the gas consumption of tx under the *current* GCM, namely t , and the *area term*, i.e., the area of a $t \times |K|$ rectangle (which is also how we draw transactions in our diagrams). The *current term* is helpful to ensure no transaction ever costs too little when the weights of all storage keys accessed by a transaction are too close to zero.¹⁶

The *area term*, on the other hand, which is the main component, intuitively corresponds to “negated throughput.” That is, executing the transaction requires holding $|K|$ locks for t time units, during which no other transactions requiring any of those storage keys can

¹⁵Except for constant GCMs.

¹⁶This term has a practical motivation. In reality, transactions will be scheduled on a small finite number of threads. Thus, even if they do not access any high-demand storage key, they occupy space in the schedule proportional to their execution time. Note, moreover, that this term could technically be omitted, and our results would still hold.



■ **Figure 7** Illustration of a worst-case transaction in terms of parallelizability. Transaction $tx \simeq (6, \{k_2, \dots, k_7\})$ (shown in) takes 6 units of time to execute and accesses 6 storage keys. However, the transaction only operates on each storage key for one unit of time each (shown in).

execute. In Figure 7, we illustrate this idea. Transaction tx prevents the execution of other transactions across its entire storage key set but only utilizes each storage key for a short period. While the current GCM charges a transaction based solely on its total computation time (i.e., the height of the rectangle), the weighted area GCM also accounts for the storage keys for which it “negated throughput”, i.e., the area occupied by the transaction. This occupied area prevents other transactions using the same storage keys from being scheduled in parallel.

Along the same line of reasoning, to further reinforce why this mechanism is a reasonable choice, consider a transaction set T . The sum $\sum_{tx \in T} t(tx) \cdot |K(tx)|$ represents the “total area” of transactions in T . Dividing this term by the number of storage keys used in T provides a lower bound on $v(T)$, serving as a rough proxy that does not require knowledge of T when computing individual gas consumptions. Finally, we note that, even in the case of non-unit weights, the term *weighted area* remains meaningful, as the area term still corresponds to the area of the respective rectangle in our diagrams if we give the column of each storage key $k \in \mathcal{K}$ a width of w_k .

4.2 Mechanisms *without* Easy Gas Estimation

We now switch gears towards mechanisms without Easy Gas Estimation that instead aim to achieve Scheduling Monotonicity and/or Efficiency. Intuitively, for a set of transactions T , such mechanisms should start from $v(T)$ and take each transaction’s marginal contribution towards $v(T)$ as its gas consumption. This, however, has to be done carefully, as, e.g., simply looking for each transaction $tx \in T$ at $v(T) - v(T \setminus \{tx\})$ does not help. Note that one can easily construct cases where removing any one transaction from T does not change $v(T)$, so all reported numbers will be 0. However, all these numbers being zero does not mean that no transaction contributes to $v(T)$; it just means that we also have to consider removing multiple transactions to see the differences. This gives us the idea to look at the marginal contribution of each transaction $tx \in T$ when added to each possible subset $S \subseteq T \setminus \{tx\}$, and not just to $S = T \setminus \{tx\}$ as before. Formally, we would like to compute tx ’s gas consumption as an aggregate of $(v(S \cup \{tx\}) - v(S))_{S \subseteq T \setminus \{tx\}}$. We next present two mechanisms based on this idea: the *Shapley* (Definition 9) and *Banzhaf* (Definition 11) GCMs, inspired by corresponding concepts in cooperative game theory.

► **Definition 9** (Shapley GCM). *Given a set of transactions T consisting of $|T| = n$ transactions and a transaction $tx \in T$, the Shapley GCM computes the amount of gas used by tx as follows:*

$$\text{gas}_T^S(tx) := \frac{1}{n!} \sum_{\sigma} [v(P_{tx}^{\sigma} \cup \{tx\}) - v(P_{tx}^{\sigma})] \quad (2)$$

$$= \sum_{S \subseteq T \setminus \{tx\}} \frac{|S|! \cdot (n - |S| - 1)!}{n!} [v(S \cup \{tx\}) - v(S)]. \quad (3)$$

Here, σ ranges over the $n!$ possible ways to order the transactions in T and P_{tx}^{σ} denotes the set of transactions that precede tx in the order σ . The equality between Equations (2) and (3) follows by counting the number of orders σ such that $P_{tx}^{\sigma} = S$, which there are $|S|! \cdot (n - |S| - 1)!$ of.

Another way to understand the Shapley GCM is through the following probabilistic experiment:

1. Select an ordering σ of the transactions in T uniformly at random.
2. Start with an empty set of transactions and add transactions one by one in the order given by σ .
3. Whenever a transaction tx is added, let $S = P_{tx}^{\sigma}$ be the set of transactions just before adding it and compute tx 's *marginal contribution* to the execution time as $v(S \cup \{tx\}) - v(S)$; i.e., by how much did the execution time increase by adding tx to the current set of transactions.
4. The gas consumption of $tx \in T$ is the expectation of its marginal contribution across the experiment.

The reader familiar with cooperative game theory will have already recognized the immediate connection with Shapley values: if we see each transaction $tx \in T$ as a player in a game with valuation function $v : 2^T \rightarrow \mathbb{R}$, then $\text{gas}_T^S(tx)$ is precisely the celebrated *Shapley value* of player tx , more traditionally written $\phi_{tx}(v)$. One classical property of Shapley values is that, given $v(\emptyset) = 0$, as is the case for us by (S4), their sum equals the valuation of the *grand coalition* T : $\sum_{tx \in T} \phi_{tx}(v) = v(T)$. The proof is straightforward: for any fixed ordering σ of T , the sum of the marginal contributions of the transactions is a telescoping sum, i.e., except first and last terms, all others appear once positively and once negatively, canceling as a result and leaving us with $v(T) - v(\emptyset) = v(T)$. Since the sum does not depend on σ , the same holds when taking the expectation with respect to σ . This already gives us our first property of the Shapley GCM, namely Efficiency:

► **Lemma 10.** *The Shapley GCM satisfies Efficiency (Property 7).*

We postpone studying the Shapley GCM further until introducing our other mechanisms in the section.

A second GCM based on the idea that a transaction's gas consumption should be its marginal contribution to the execution time is the *Banzhaf* mechanism:

► **Definition 11** (Banzhaf GCM). *Given a set of transactions T consisting of $|T| = n$ transactions and a transaction $tx \in T$, the Banzhaf GCM computes the amount of gas used by tx as follows:*

$$\text{gas}_T^B(tx) := \frac{1}{2^{n-1}} \sum_{S \subseteq T \setminus \{tx\}} [v(S \cup \{tx\}) - v(S)].$$

As for the Shapley mechanism, the Banzhaf mechanism can be understood probabilistically, but this time with a separate experiment for each transaction tx . Sample a subset S of transactions other than tx uniformly at random and compute tx 's marginal contribution to the execution time when added to S , namely $v(S \cup \{tx\}) - v(S)$. The gas consumption of tx is then its expected marginal contribution to the execution time. The familiar reader will recognize this as the definition of the *Banzhaf power index* $\beta_{tx}(v)$. For consistency with *Shapley values*, we will instead call these *Banzhaf values*. While more straightforward than the corresponding experiment used in defining Shapley values, the fact that we now have n separate experiments makes the sum of the values no longer well-behaved, losing Efficiency for the Banzhaf mechanism:

► **Lemma 12.** *The Banzhaf GCM does not satisfy Efficiency (Property 7).*

We conclude this section by introducing two additional reasonable mechanisms *without* Easy Gas Estimation. These mechanisms are notably more straightforward than the Shapley and Banzhaf GCMs, as they avoid computing marginal contributions. However, they have other drawbacks that will become more apparent when we begin studying their normative properties alongside the other mechanisms.

► **Definition 13** (Time-Proportional Makespan GCM). *Given a set of transactions T and a transaction $tx \in T$, the Time-Proportional Makespan (TPM) GCM computes the amount of gas used by tx as follows:*

$$gas_T^{TPM}(tx) := \frac{t(tx)}{\sum_{tx' \in T} t(tx')} \cdot v(T).$$

► **Definition 14** (Equally-Split Makespan GCM). *Given a set of transactions T and a transaction $tx \in T$, the Equally-Split Makespan (ESM) GCM computes the amount of gas used by tx as follows:*

$$gas_T^{ESM}(tx) := \frac{v(T)}{|T|}.$$

We also consider the following rather pathological mechanism because of the combination of properties it turns out to satisfy (that none of our other mechanisms do):

► **Definition 15** (Exponentially-Split Makespan GCM). *Given a set of transactions T and a transaction $tx \in T$, the Exponentially-Split Makespan (XSM) GCM computes the amount of gas used by tx as follows:*

$$gas_T^{XSM}(tx) := \frac{v(T)}{3^{|T|}}.$$

5 Analysis of Our GCMs

In this section, we provide a detailed analysis of the normative properties of our GCMs. Our results are summarized in Table 1.

5.1 Mechanisms with Easy Gas Estimation

In this section, we analyze the current and Weighted Area GCMs. By definition, both satisfy Easy Gas Estimation (Property 8) and are Poly-time Computable (Property 9). Since they satisfy Easy Gas Estimation but are not constant GCMs, Theorem 7 implies that *neither*

■ **Table 1** Comparison of GCMs based on their adherence to the defined properties. $<$ indicates that the mechanism *strictly* satisfies the property, $=$ indicates *trivial* satisfaction (by equality), and \leq indicates satisfaction (not necessarily strict). A “ \checkmark ” means the property is satisfied; “ \times ” means it is not satisfied. For computational complexity, “ v ” means the mechanism is as hard to compute as $v(\cdot)$ itself, “ $S(v)$ ” means it is as hard as computing Shapley values for v , and “ $B(v)$ ” means it is as hard as computing Banzhaf values for v .

Property	Current	W. Area	Shapley	Banzhaf	TPM	ESM	XSM
Storage Key Monotonicity (Property 1)	$=$	$<$	\leq	\leq	\leq	\leq	\leq
Time Monotonicity (Property 2)	$<$	$<$	$<$	$<$	$<$	\leq	\leq
Storage Key-Time Monotonicity (Property 3)	\leq	$<$	\leq	\leq	\leq	\leq	\leq
Set Inclusion (Property 4)	$<$	$<$	\times	\times	\leq	\leq	\times
Transaction Bundling (Property 5)	$=$	\leq	\times	\leq	\leq	\times	$<$
Scheduling Monotonicity (Property 6)	\times	\times	\times	\times	\times	$<$	$<$
Efficiency (Property 7)	\times	\times	\checkmark	\times	\checkmark	\checkmark	\times
Easy Gas Estimation (Property 8)	\checkmark	\checkmark	\times	\times	\times	\times	\times
Poly-time Computable (Property 9)	\checkmark	\checkmark	$S(v)$	$B(v)$	v	v	v

satisfies Scheduling Monotonicity (Property 6) nor Efficiency (Property 7). Furthermore, both mechanisms satisfy strict Time Monotonicity (Property 2): increasing the time t of a transaction strictly increases its gas consumption. Similarly, both satisfy strict Set Inclusion (Property 4): since transactions’ gas consumptions are strictly positive, a strict superset of a given set of transactions consumes strictly more gas. For the remaining three properties, the two mechanisms behave slightly differently:

Storage Key Monotonicity (Property 1). The current mechanism ignores the set of storage keys K that a transaction accesses, so replacing K with a strict superset of it does not change the transaction’s gas consumption. Therefore, the current mechanism satisfies Storage Key Monotonicity *with equality*. On the other hand, the Weighted Area mechanism adds an extra term of $t \cdot w_k > 0$ to the gas consumption for each additional storage key k added to K , so it satisfies strict Storage Key Monotonicity.

Storage Key-Time Monotonicity (Property 3). From the above results on whether our two mechanisms satisfy Storage Key Monotonicity and Time Monotonicity, Lemmas 4 and 5 allow us to conclude that the current mechanism satisfies Storage Key-Time Monotonicity, while the Weighted Area mechanism satisfies it strictly.

Transaction Bundling (Property 5). Concatenating two transactions with times t_1 and t_2 results in a transaction with time $t_1 + t_2$. Since the current mechanism equates gas consumption with time, the bundled transaction has the same gas consumption as the two individual transactions combined. This implies that the current mechanism satisfies Transaction Bundling *with equality*. Finally, the Weighted Area mechanism satisfies Transaction Bundling, as shown below. If we restrict ourselves to bundling transactions with different storage key sets, the property is strictly satisfied.

► **Lemma 16.** *The Weighted Area GCM satisfies Transaction Bundling (Property 5). If we only consider bundling transactions with different storage key sets, the property is strictly satisfied.*

Proof. Consider two transactions $tx_1 \simeq (t_1, K_1)$ and $tx_2 \simeq (t_2, K_2)$. Let $tx_3 \simeq (t_1 + t_2, K_1 \cup K_2)$ be a transaction consisting of the concatenation of tx_1 and tx_2 . We want to show that $\text{gas}^{WA}(tx_1) + \text{gas}^{WA}(tx_2) \leq \text{gas}^{WA}(tx_3)$. By definition, this amounts to:

$$t_1 \cdot \left(1 + \sum_{k \in K_1} w_k\right) + t_2 \cdot \left(1 + \sum_{k \in K_2} w_k\right) \leq (t_1 + t_2) \cdot \left(1 + \sum_{k \in K_1 \cup K_2} w_k\right) \quad (4)$$

Which is true because $\sum_{k \in K_i} w_k \leq \sum_{k \in K_1 \cup K_2} w_k$ for $i \in \{1, 2\}$. Because the weights are strictly positive, equality occurs if and only if $K_1 = K_1 \cup K_2$ and $K_2 = K_1 \cup K_2$; i.e., $K_1 = K_2$. Hence, the property is satisfied strictly if we restrict bundling transactions to cases where $K_1 \neq K_2$. \blacktriangleleft

5.2 Mechanisms *without* Easy Gas Estimation

In this section, we analyze the Shapley, Banzhaf, TPM, ESM, and XSM GCMs. By definition, all of them require knowledge of T to compute $\text{gas}_T(tx)$, so they do not satisfy Easy Gas Estimation (Property 8). Next, we examine each of the remaining properties individually and analyze whether they hold for our five mechanisms.

Poly-time Computable (Property 9). The TPM, ESM, and XSM GCMs are all Poly-time Computable whenever determining the makespan of a given set of transactions under the chosen scheduler is feasible in polynomial time, i.e., when $v(T)$ can be computed in polynomial time.¹⁷ In fact, the problems are all equally difficult to computing such makespans. In contrast, computing gas consumptions under the Shapley and Banzhaf GCMs hits another hurdle: it is no longer enough to be able to efficiently compute $v(T)$, but instead, one needs to be able to compute the Shapley or Banzhaf values of the transactions, involving aggregating over $(v(T'))_{T' \subseteq T}$. Hence, computing Shapley and Banzhaf values is, in general, NP-hard and #P-complete [19, 24, 41, 42, 52]. Moreover, no deterministic polynomial-time algorithm can approximate the Shapley values within a constant factor unless $P = NP$. Using randomization could, in principle, circumvent this: to approximate an average consisting of exponentially many terms, sample polynomially many uniformly at random, and take their average. One could imagine implementing this in a blockchain via a VRF or similar mechanism to ensure unbiased randomness. Still, this comes with a notable downside: the accuracy of the computed gas consumption may be hard to predict in advance and might vary wildly. Moreover, even with randomization, one could still run into issues computing the sampled terms, which is as hard as evaluating v a constant number of times. See [15] for an ampler discussion of computing Shapley and Banzhaf values. Note that the previous results mostly pertain to functions v of a different shape than ours (i.e., not related to scheduling). We have not attempted to show that hardness is retained in our context, but expect this to be true. Note still that because the Shapley GCM is Efficient, it can be used to compute $v(T)$ by adding up the gas consumptions of all transactions in T , meaning that the Shapley GCM is at least as difficult to compute as $v(T)$. However, this simple reduction no longer works for the Banzhaf GCM.

Efficiency (Property 7). We have already seen that the Shapley GCM is Efficient (Lemma 10) while the Banzhaf GCM is not (Lemma 12). Moreover, by adding up the gas consumption, one can immediately see from the definitions that TPM and ESM are

¹⁷ Notably, this is essentially never true for non-trivial makespan minimization problems. In particular, if we restrict our attention to the case of unit-length transactions (i.e., with $t = 1$) and infinitely many threads $n = \infty$, then checking for the existence of a schedule with makespan c corresponds to checking whether the intersection graph of the transactions (i.e., where edges correspond to transactions with intersecting storage key sets) is c -colorable. For $c = 3$, this is well-known to be NP-complete, but this result is for general graphs. However, there is a straightforward way to model any graph $G = (V, E)$ as an intersection graph of transactions: vertices $v \in V$ are transactions and for every edge $(u, v) \in E$, create a new storage key k and add it to the storage key sets of transactions u and v .

Efficient, while XSM is not. We take this occasion to also note that any GCM can be made Efficient by appropriately scaling its output. Most notably, applying this to the Banzhaf GCM yields a mechanism based on the well-known *normalized Banzhaf values*, though the resulting mechanism otherwise seems rather poorly behaved.

Scheduling Monotonicity (Property 6). The ESM and XSM GCMs can be easily seen to strictly satisfy Scheduling Monotonicity. This is because, under both mechanisms, given a set of transactions T , the gas consumption of any transaction in T is computed as $f(|T|) \cdot v(T)$, where f is either $\frac{1}{x}$ or $\frac{1}{3^x}$. Notably, the first factor depends only on $|T|$, so if a transaction in T were modified in a way that increases the makespan, this increase would also be reflected proportionally in its gas consumption. In contrast, the Shapley, Banzhaf, and TPM GCMs do *not* satisfy Scheduling Monotonicity (proven in Lemma 17 below). This may be particularly surprising for the Shapley and Banzhaf GCMs, as they were specifically designed to account for marginal increases in makespan. The catch is that Scheduling Monotonicity considers replacing a transaction tx with another one leading to an increase in makespan. However, some elements in $(v(S \cup \{tx\}) - v(S))_{S \subseteq T \setminus \{tx\}}$ may still decrease as a result (except for $S = T \setminus \{tx\}$, for which we assumed an increase). Because both the Shapley and Banzhaf values of tx take a weighted average over these values, the average might still decrease, which is what happens in the example in the lemma below.

► **Lemma 17.** *The Shapley, Banzhaf and TPM GCMs do not satisfy Scheduling Monotonicity (Property 6).*

Proof. Consider the set of transactions $T = \{tx_1, tx_2\}$ and two additional transactions tx_3 and tx_4 such that $tx_1 \simeq (1, \{k_1\})$, $tx_2 \simeq (3, \{k_2\})$, $tx_3 \simeq (2, \{k_1\})$, and $tx_4 \simeq (1, \{k_2\})$. Then, for any number of threads $n \geq 2$ and the optimal scheduler we have $v(T \cup \{tx_3\}) = 3 < 4 = v(T \cup \{tx_4\})$. However:

$$\begin{aligned} \text{gas}_{T \cup \{tx_3\}}^S(tx_3) &= \frac{2 + 2 + 2 + 0 + 0 + 0}{6} = 1 > \frac{5}{6} = \frac{1 + 1 + 1 + 1 + 1 + 0}{6} = \text{gas}_{T \cup \{tx_4\}}^S(tx_4) \\ \text{gas}_{T \cup \{tx_3\}}^B(tx_3) &= \frac{2 + 2 + 0 + 0}{4} = 1 > \frac{3}{4} = \frac{1 + 0 + 1 + 1}{4} = \text{gas}_{T \cup \{tx_4\}}^B(tx_4) \\ \text{gas}_{T \cup \{tx_3\}}^{\text{TPM}}(tx_3) &= \frac{2}{1 + 3 + 2} \cdot 3 = 1 > \frac{4}{5} = \frac{1}{1 + 3 + 1} \cdot 4 = \text{gas}_{T \cup \{tx_4\}}^{\text{TPM}}(tx_4). \end{aligned}$$

So, the Shapley, Banzhaf, and TPM GCMs all violate Scheduling Monotonicity in this case. ◀

Transaction Bundling (Property 5). We find that the Banzhaf, TPM and XSM GCMs satisfy Transaction Bundling, only XSM satisfying it strictly, while the Shapley and ESM GCMs do not satisfy the property. We prove these facts in the following 5 lemmas.

► **Lemma 18.** *The Shapley GCM does not satisfy Transaction Bundling (Property 5).*

► **Lemma 19.** *The Banzhaf GCM satisfies Transaction Bundling (Property 5).*

► **Lemma 20.** *The TPM GCM satisfies Transaction Bundling (Property 5).*

► **Lemma 21.** *The ESM GCM does not satisfy Transaction Bundling (Property 5).*

► **Lemma 22.** *The XSM GCM strictly satisfies Transaction Bundling (Property 5).*

Set Inclusion (Property 4). We find that the TPM and ESM GCMs satisfy Set Inclusion, while the Shapley, Banzhaf and XSM GCMs do not satisfy the property. We prove these facts in the following 5 lemmas.

► **Lemma 23.** *The Shapley GCM does not satisfy Set Inclusion (Property 4).*

► **Lemma 24.** *The Banzhaf GCM does not satisfy Set Inclusion (Property 4).*

► **Lemma 25.** *The TPM GCM satisfies Set Inclusion (Property 4).*

► **Lemma 26.** *The ESM GCM satisfies Set Inclusion (Property 4).*

► **Lemma 27.** *The XSM GCM does not satisfy Set Inclusion (Property 4).*

Storage Key-Time Monotonicity (Property 3). All five mechanisms satisfy this property. For the ESM and XSM GCMs, this is an immediate consequence of property (S3). For the Shapley and Banzhaf GCMs, one can see this by recalling that they compute the gas consumption of a transaction $tx \in T$ as a weighted average over $(v(S \cup \{tx\}) - v(S))_{S \subseteq T \setminus \{tx\}}$. Hence, by property (S3), when tx is replaced with some $tx' \succeq tx$, no term in the previous decreases, so their weighted average also does not decrease. Finally, for the TPM GCM, we show this in the lemma after the next paragraph*.

Storage Key Monotonicity (Property 1) and Time Monotonicity (Property 2). Because all five mechanisms satisfy Storage Key-Time Monotonicity, by Lemma 4, they also all satisfy Storage Key Monotonicity and Time Monotonicity. Out of the five mechanisms, the Shapley, Banzhaf, and TPM GCMs satisfy the property strictly. For the first two, this is because when $t(tx)$ increases, at least one term in $(v(S \cup \{tx\}) - v(S))_{S \subseteq T \setminus \{tx\}}$ strictly increases, namely the term for $S = \emptyset$, while no terms decrease by property (S3). Last, for the TPM GCM, we show this in the lemma below.

► **Lemma 28.** *The TPM GCM satisfies Storage Key-Time Monotonicity (Property 3) and strict Time Monotonicity (Property 2).*

6 Towards a Fee Market for Parallel Execution

Armed with an understanding of the trade-offs between desirable properties in a GCM, in particular the impossibility of satisfying all properties within a single mechanism, and informed by our analysis of various candidate mechanisms and their properties, we propose two mechanisms for practical implementations of a fee market supporting parallel execution. Each represents one side of the design spectrum: one drawn from the class of mechanisms with Easy Gas Estimation, and the other from the class without it.

The advantage of adopting a mechanism with Easy Gas Estimation is that satisfying this property ensures that each transaction consumes a fixed amount of gas, regardless of other transactions in the same block. From the perspective of currently deployed TFMs (e.g., EIP-1559), which process transactions with fixed sizes, nothing changes. Thus, the existing properties of these mechanisms remain intact. On a high level, the key properties we strive for in a TFM are incentive compatibility for both block producers and users, welfare optimality, and collusion resistance. However, no TFM can achieve all these properties simultaneously [18, 17, 27]. Importantly, when composing a GCM that satisfies Easy Gas Estimation with a TFM of choice, the level of sophistication required from users in their bidding strategy does not increase, unlike in currently deployed fee markets for parallel execution [40, 44].

On the other hand, foregoing Easy Gas Estimation makes it possible to price transactions according to the load they impose on the network relative to the other transactions in the block, rather than evaluating each one in isolation. Mechanisms in this class can align the total gas charged in a block with the actual execution cost of that block, enabling resource pricing at the block level rather than only at the transaction level. Recent discussions in Ethereum research have explored similar designs under the umbrella of block-level fee markets [11]. If the protocol is already moving toward block-level metering for other resources, such as data availability or storage contention, extending this approach to execution costs could enable direct efficiency gains from parallel execution that are not achievable with transaction-level pricing alone.

For the class of mechanisms with Easy Gas Estimation, the natural choice is the Weighted Area GCM. Given that it satisfies Easy Gas Estimation, it achieves the most additional properties one could hope for in a (non-constant) GCM (see Theorem 7).

For the class of mechanisms without Easy Gas Estimation, we propose the Time-Proportional Makespan GCM as a concrete candidate. By allocating gas in proportion to each transaction’s execution time relative to the block’s total makespan, it provides a simple way to align gas consumption with how transactions constrain parallel execution within a block.

The decision between these approaches ultimately depends on protocol-level priorities: maintaining user-side simplicity and compatibility with existing TFMs through Easy Gas Estimation, which also makes implementation simpler by reusing current transaction-level pricing infrastructure, or pursuing block-level efficiency and more accurate resource pricing at the cost of giving up the previous benefits.

7 Related Work

7.1 Transaction Fee Mechanisms

There is extensive research on blockchain fee markets, with a particular focus on Ethereum and Bitcoin. Early studies primarily examined Bitcoin, exploring monopolistic pricing mechanisms [37, 64]. More recent contributions to this field include [48, 26, 50]. Unlike these works, our study concerns measuring storage key usage on a blockchain with client-side parallel execution, rather than focusing on pricing.

The TFM design framework was introduced by Roughgarden [54, 55]. Roughgarden’s analysis of the EIP-1559 mechanism [12] initiated an active line of research on TFMs. Chung and Shi [18] demonstrated that no TFM can be ideal – meaning it cannot simultaneously be incentive-compatible for users and block producers while also being resistant to collusion between the two. This conclusion holds even for weaker definitions of collusion resilience, as shown by Chung et al. [17] and Gafni and Yaish [27]. Finally, attempts to address these limitations using cryptographic techniques [59, 62] have made progress in overcoming certain impossibilities, while other attempts relax the desiderata [28]. However, designing an ideal TFM still remains out of reach. While these studies examine the limitations of TFMs, our focus is on GCMs for parallel execution and how to integrate them with a TFM.

A related body of work examines the dynamics of TFMs over multiple blocks, particularly focusing on the base fee in EIP-1559. Leonardos et al. [38, 39] demonstrate that the stability of the base fee depends on the adjustment parameter, with short-term volatility but long-term block size stability. Reijnders et al. [53] suggest using an adaptive adjustment parameter to mitigate block size fluctuations, while Ferreira et al. [25] highlight user experience issues caused by bounded base fee oscillations. Additionally, Hougaard and Pourpouneh [32] and Azouvi et al. [6] reveal that the base fee can be manipulated by non-myopic miners.

Given the discussion surrounding multi-dimensional fees in Ethereum [9, 10] and the deployment of EIP-4844 [13] (a first step towards a multi-dimensional fee market on Ethereum), a recent line of work explores multi-dimensional fee markets, focusing on efficient pricing mechanisms and their optimality. This work is further refined by Diamandis et al. [21], who design and analyze multi-dimensional blockchain fee markets to align incentives and improve network performance. Building on this, Angeris et al. [3] prove that such fee markets are nearly optimal, with efficiency improving over time even under adversarial conditions. Multidimensional fee markets are closely related to fee markets designed for parallel execution. In particular, in the weighted area GCM, the weights can be interpreted as fees within a multidimensional fee market. Unlike previous literature on multidimensional fee markets, we focus on parallelization, introduce desirable properties, and evaluate how various mechanisms perform.

Further extensions of TFMs have emerged. Bahrani et al. [8] consider TFMs in the presence of maximal extractable value (MEV), i.e., value extractable by the block producer. Further, Wang et al. [61] design a fee mechanism for proof networks, whereas Bahrani et al. [7] introduce a transaction fee mechanism for heterogeneous computation. Our work most closely relates to the latter, but, in contrast, our chosen approach is closer to multidimensional fee markets, trading complexity for the block producer for stronger incentive compatibility for the user.

Local fee markets have recently been a topic of discussion in the blockchain space [23, 20, 40, 34]. The core idea is that transactions interacting with highly contested states incur higher fees, while those involving non-contested states pay lower fees. However, discussions on local fee markets have largely remained high-level, without a precise characterization of the desired properties beyond this general goal. Moreover, currently implemented local fee markets [40] require significant user sophistication to set fees appropriately. In this work, we formalize the desiderata for fee markets in the context of parallel execution and identify the weighted area GCM as a promising candidate. One key advantage is its compatibility with a TFM, enabling simple fee estimation for users.

7.2 Parallel Execution

Blockchain concurrency has been a focal point in an active line of research. In particular, numerous efforts have aimed to enable parallel transaction processing through speculative execution [58, 65, 2, 22, 4, 29, 16, 56]. Note that speculative execution is already deployed by multiple blockchains [5, 57, 43]. Static analysis has also been employed to identify parallelizable transactions, though it cannot completely eliminate inherent dependencies [51, 45]. Similarly, Neiheiser et al. [47] demonstrate how parallel execution can assist struggling nodes in catching up. While these works are orthogonal to ours, they highlight the overhead of parallel execution when there is no advance knowledge about a transaction’s state accesses.

Further, Saraph and Herlihy [56] and Heimbach et al. [30] have evaluated the parallelization potential of the Ethereum workload. The latter demonstrates that a speedup of approximately fivefold is achievable, assuming state accesses are known in advance. Additionally, Solana [60] and Sui [46] already perform parallel execution with advance knowledge of state accesses. However, in practice, state accesses are not known beforehand on many blockchains such as Ethereum. There, less than 2% of transactions disclose them proactively, as shown by Heimbach et al. [31], due to a lack of incentives. In this work, we aim to take a step toward unlocking the parallelization potential by designing a TFM that supports parallel execution. This mechanism relies on the disclosure of state accesses as done in Solana [60] and Sui [46].

7.3 Cloud Computing Pricing and Parallel Resource Allocation

While our primary focus is on transaction fee mechanisms for blockchain networks, parallels can be drawn to resource allocation and pricing in cloud computing. Several papers study pricing schemes for cloud computing resources in a monopolistic setting, where a single provider sells access to multiple resources and sets prices to optimize revenue [63]. In these models, the provider posts prices for resource usage, users select resource bundles to maximize their individual surplus, and the provider earns revenue from the resulting allocation. Mechanisms such as CloudPack [33] extend this approach by incorporating workload flexibility, using concepts like Shapley values for fair cost distribution among colocated jobs. Similarly, works such as [49] address resource contention through dynamic pricing strategies that adjust costs based on runtime slowdowns.

While structurally similar, the blockchain context introduces an additional constraint. In addition to objectives like maximizing provider revenue, it emphasizes maximizing throughput by ensuring transactions are processed efficiently, minimizing their collective execution time within a block. Moreover, blockchain transactions explicitly declare access to distinct storage keys, introducing contention through overlapping access patterns. This adds complexity compared to cloud models, which typically abstract resource demands as scalar quantities and address contention at a more aggregated level.

Capturing the effects of storage key-level conflicts on parallel execution in blockchains requires fee computation mechanisms that are sensitive to these fine-grained interactions. Thus, while conceptually addressing similar challenges, the blockchain setting imposes unique constraints that motivate the need for specialized GCMs as explored in this work.

8 Conclusion

In this work, we took a step towards creating a fee market that meets the demands of parallel execution environments while also upholding the properties we want from a TFM.

Recently, the idea of local fee markets has been proposed for blockchains that support parallel execution. However, to the best of our knowledge, before this work, the demands on these fee markets have only been outlined at a very high level, and the markets that have been implemented are not ideal yet, e.g., they require high levels of sophistication from users when bidding.

In this work, we addressed this gap by introducing a framework with two key components: a GCM, which measures the execution-related load a transaction imposes on the network in units of gas, and a TFM, which determines the cost associated with each unit of gas. We then formalized the desired properties for the GCM in such a fee market. After outlining these desiderata, we evaluated various mechanisms against them and identified two strong candidates through this analysis: the *weighted area* GCM for the class of mechanisms with Easy Gas Estimation, and the *time-proportional makespan* GCM for the class without it.

Setting the right incentives in fee markets for parallel execution is crucial to unlocking the full potential of execution layer parallelization, and we hope that our work contributes to the development of fee markets capable of meeting the demands of such environments.

References

- 1 Bahar Acilan, Andrei Constantinescu, Lioba Heimbach, and Roger Wattenhofer. Transaction Fee Market Design for Parallel Execution, 2025. doi:10.48550/arXiv.2502.11964.
- 2 Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. Parblockchain: Leveraging Transaction Parallelism in Permissioned Blockchain Systems. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1337–1347. IEEE, 2019. doi:10.1109/ICDCS.2019.00134.

- 3 Guillermo Angeris, Theo Diamandis, and Ciamac Moallemi. Multidimensional Blockchain Fees are (Essentially) Optimal. *arXiv preprint arXiv:2402.08661*, 2024. doi:10.48550/arXiv.2402.08661.
- 4 Parwat Singh Anjana, Sweta Kumari, Sathya Peri, Sachin Rathor, and Archit Somani. Optsmart: A space efficient optimistic concurrent execution of smart contracts. *Distributed and Parallel Databases*, pages 1–53, 2022.
- 5 Aptos Labs. Aptos, 2023. Accessed: 2025-01-23. URL: <https://www.aptoslabs.com>.
- 6 Sarah Azouvi, Guy Goren, Lioba Heimbach, and Alexander Hicks. Base Fee Manipulation in Ethereum’s EIP-1559 Transaction Fee Mechanism. In *37th International Symposium on Distributed Computing*, 2023.
- 7 Maryam Bahrani and Naveen Durvasula. Resonance: Transaction Fees for Heterogeneous Computation. *arXiv preprint arXiv:2411.11789*, 2024. doi:10.48550/arXiv.2411.11789.
- 8 Maryam Bahrani, Pranav Garimidi, and Tim Roughgarden. Transaction Fee Mechanism Design with Active Block Producers. In *International Conference on Financial Cryptography and Data Security*, pages 85–90. Springer, 2024. doi:10.1007/978-3-031-69231-4_6.
- 9 Vitalik Buterin. Multidimensional EIP-1559. <https://ethresear.ch/t/multidimensional-eip-1559/11651>, 2022. Accessed: 2025-01-23.
- 10 Vitalik Buterin. Multidimensional Pricing for EIP-1559. <https://vitalik.eth.limo/general/2024/05/09/multidim.html>, May 2024. Accessed: 2025-01-23.
- 11 Vitalik Buterin. Block-level fee markets: Four easy pieces. <https://ethresear.ch/t/block-level-fee-markets-four-easy-pieces/21448>, 2025. Accessed: 2025-08-04.
- 12 Vitalik Buterin, Eric Conner, Rick Dudley, Matthew Slipper, Ian Norden, and Abdelhamid Bakhta. Eip-1559: Fee market change for eth 1.0 chain. <https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1559.md>, 2024. Accessed: 2025-01-23.
- 13 Vitalik Buterin, Dankrad Feist, Diederik Loerakker, George Kadianakis, Matt Garnett, Mofi Taiwo, and Ansgar Dietrichs. EIP-4844: Shard Blob Transactions. <https://eips.ethereum.org/EIPS/eip-4844>, 2022. Accessed: 2025-01-23.
- 14 Vitalik Buterin and Martin Swende. Eip-2930: Optional access lists, October 2020. Accessed: 2025-01-22. URL: <https://eips.ethereum.org/EIPS/eip-2930>.
- 15 Georgios Chalkiadakis, Edith Elkind, and Michael Wooldridge. *Weighted Voting Games*, pages 49–70. Springer International Publishing, Cham, 2012. doi:10.1007/978-3-031-01558-8_5.
- 16 Yang Chen, Zhongxin Guo, Runhuai Li, Shuo Chen, Lidong Zhou, Yajin Zhou, and Xian Zhang. Forerunner: Constraint-Based Speculative Transaction Execution for Ethereum. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 570–587, 2021. doi:10.1145/3477132.3483564.
- 17 Hao Chung, Tim Roughgarden, and Elaine Shi. Collusion-Resilience in Transaction Fee Mechanism Design. In *Proceedings of the 25th ACM Conference on Economics and Computation*, pages 1045–1073, 2024. doi:10.1145/3670865.3673550.
- 18 Hao Chung and Elaine Shi. Foundations of Transaction Fee Mechanism Design. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3856–3899. SIAM, 2023. doi:10.1137/1.9781611977554.CH150.
- 19 Xiaotie Deng and Christos H. Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19(2):257–266, 1994. doi:10.1287/moor.19.2.257.
- 20 Theo Diamandis, Tarun Chitra, and 0xShitTrader. Toward multidimensional solana fees. *Umbra Research*, 2024. URL: <https://www.umbraresearch.xyz/writings/toward-multidimensional-solana-fees>.
- 21 Theo Diamandis, Alex Evans, Tarun Chitra, and Guillermo Angeris. Designing Multidimensional Blockchain Fee Markets. In *5th Conference on Advances in Financial Technologies (AFT 2023)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.AFT.2023.4.

- 22 Thomas Dickerson, Paul Gazzillo, Maurice Herlihy, and Eric Koskinen. Adding Concurrency to Smart Contracts. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 303–312, 2017. doi:10.1145/3087801.3087835.
- 23 Eclipse Labs. Local fee markets are necessary to scale ethereum. *Eclipse*, 2024. URL: <https://www.eclipse.xyz/articles/local-fee-markets-are-necessary-to-scale-ethereum>.
- 24 Piotr Faliszewski and Lane Hemaspaandra. The complexity of power-index comparison. *Theoretical Computer Science*, 410(1):101–107, 2009. doi:10.1016/j.tcs.2008.09.034.
- 25 Matheus VX Ferreira, Daniel J Moroz, David C Parkes, and Mitchell Stern. Dynamic Posted-Price Mechanisms for the Blockchain Transaction-Fee Market. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 86–99, 2021.
- 26 Yotam Gafni and Aviv Yaish. Greedy Transaction Fee Mechanisms for (Non-) Myopic Miners. *arXiv preprint arXiv:2210.07793*, 5, 2022. doi:10.48550/arXiv.2210.07793.
- 27 Yotam Gafni and Aviv Yaish. Barriers to Collusion-Resistant Transaction Fee Mechanisms. In *Proceedings of the 25th ACM Conference on Economics and Computation*, pages 1074–1096, 2024. doi:10.1145/3670865.3673469.
- 28 Yotam Gafni and Aviv Yaish. Discrete and Bayesian Transaction Fee Mechanisms. In *The International Conference on Mathematical Research for Blockchain Economy*, pages 145–171. Springer, 2024. doi:10.1007/978-3-031-68974-1_8.
- 29 Rati Gelashvili, Alexander Spiegelman, Zhuolun Xiang, George Danezis, Zekun Li, Yu Xia, Runtian Zhou, and Dahlia Malkhi. Block-STM: Scaling Blockchain Execution by Turning Ordering Curse to a Performance Blessing. *arXiv preprint arXiv:2203.06871*, 2022. doi:10.48550/arXiv.2203.06871.
- 30 Lioba Heimbach, Quentin Knip, Yann Vonlanthen, and Roger Wattenhofer. Defi and nfts hinder blockchain scalability. In *International Conference on Financial Cryptography and Data Security*, pages 291–309. Springer, 2023. doi:10.1007/978-3-031-47751-5_17.
- 31 Lioba Heimbach, Quentin Knip, Yann Vonlanthen, Roger Wattenhofer, and Patrick Züst. Dissecting the EIP-2930 Optional Access Lists. *arXiv preprint arXiv:2312.06574*, 2023. doi:10.48550/arXiv.2312.06574.
- 32 Jens Leth Hougaard and Mohsen Pourpouneh. Farsighted Miners Under Transaction Fee Mechanism EIP-1559. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE, 2023. doi:10.1109/ICBC56567.2023.10174974.
- 33 Vatche Isahagian, Raymond Sweha, Azer Bestavros, and Jonathan Appavoo. Cloudpack: Exploiting workload flexibility through rational pricing. In *Proceedings of the 13th ACM/I-FIP/USENIX Middleware Conference*, December 2012.
- 34 keyneom. Local fee markets in ethereum. *Ethereum Research*, 2024. URL: <https://ethresear.ch/t/local-fee-markets-in-ethereum/20754>.
- 35 Mysten Labs. Sui object model documentation, 2025. Accessed: 2025-01-22. URL: <https://docs.sui.io/concepts/object-model>.
- 36 Solana Labs. Solana transactions documentation, 2025. Accessed: 2025-01-22. URL: <https://solana.com/de/docs/core/transactions>.
- 37 Ron Lavi, Or Sattath, and Aviv Zohar. Redesigning Bitcoin’s Fee Market. *ACM Transactions on Economics and Computation*, 10(1):1–31, 2022. doi:10.1145/3530799.
- 38 Stefanos Leonardos, Barnabé Monnot, Daniël Reijbergen, Efstratios Skoulakis, and Georgios Piliouras. Dynamical Analysis of the EIP-1559 Ethereum Fee Market. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 114–126, 2021. doi:10.1145/3479722.3480993.
- 39 Stefanos Leonardos, Daniël Reijbergen, Barnabé Monnot, and Georgios Piliouras. Optimality Despite Chaos in Fee Markets. In *International Conference on Financial Cryptography and Data Security*, pages 346–362. Springer, 2023. doi:10.1007/978-3-031-47751-5_20.
- 40 Lostin. The truth about solana local fee markets. *Helius*, 2025. URL: <https://www.helius.dev/blog/solana-local-fee-markets>.

- 41 Tomomi Matsui and Yasuko Matsui. A survey of algorithms for calculating power indices of weighted majority games. *Journal of the Operations Research Society of Japan*, 43:71–86, November 2000. doi:10.15807/jorsj.43.71.
- 42 Yasuko Matsui and Tomomi Matsui. Np-completeness for calculating power indices of weighted majority games. *Theoretical Computer Science*, 263(1):305–310, 2001. Combinatorics and Computer Science. doi:10.1016/S0304-3975(00)00251-6.
- 43 Monad Labs. Monad, 2023. Accessed: 2025-01-23. URL: <https://www.monad.xyz>.
- 44 Sebastian Mueller. X post by Sebastian Mueller. <https://x.com/NaitsabesMue/status/1862519048069959893>, 2025. Accessed: 2025-02-08.
- 45 Maurizio Murgia, Letterio Galletta, and Massimo Bartoletti. A Theory of Transaction Parallelism in Blockchains. *Logical Methods in Computer Science*, 17, 2021. doi:10.46298/LMCS-17(4:10)2021.
- 46 Mysten Labs. Sui, 2023. Accessed: 2025-01-23. URL: <https://sui.io>.
- 47 Ray Neiheiser, Arman Babaei, Ioannis Alexopoulos, Marios Kogias, and Eleftherios Kokoris Kogias. Pythia: Supercharging Parallel Smart Contract Execution to Guide Stragglers and Full Nodes to Safety. https://aftsib.com/papers/SIB24_paper_4.pdf, 2024. Accessed: 2025-01-23.
- 48 Noam Nisan. Serial Monopoly on Blockchains. *arXiv preprint arXiv:2311.12731*, 2023. doi:10.48550/arXiv.2311.12731.
- 49 Qi Pei, Yipeng Wang, and Seunghee Shin. Litmus: Fair pricing for serverless computing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, ASPLOS '24, pages 155–169. ACM, April 2024. doi:10.1145/3622781.3674181.
- 50 Paolo Penna and Manvir Schneider. Serial Monopoly on Blockchains with Quasi-patient Users. *arXiv preprint arXiv:2405.17334*, 2024. doi:10.48550/arXiv.2405.17334.
- 51 George Pirlea, Amrit Kumar, and Ilya Sergey. Practical Smart Contract Sharding with Ownership and Commutativity Analysis. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 1327–1341, 2021. doi:10.1145/3453483.3454112.
- 52 K. Prasad and J. S. Kelly. Np-completeness of some problems concerning voting games. *International Journal of Game Theory*, 19(1):1–9, March 1990. doi:10.1007/BF01753703.
- 53 Daniël Reijnders, Shyam Sridhar, Barnabé Monnot, Stefanos Leonardos, Stratis Skoulakis, and Georgios Piliouras. Transaction Fees on a Honeymoon: Ethereum’s EIP-1559 One Month Later. In *2021 IEEE International Conference on Blockchain (Blockchain)*, pages 196–204. IEEE, 2021. doi:10.1109/BLOCKCHAIN53845.2021.00034.
- 54 Tim Roughgarden. Transaction Fee Mechanism Design for the Ethereum Blockchain: An Economic Analysis of EIP-1559. *arXiv preprint arXiv:2012.00854*, 2020. arXiv:2012.00854.
- 55 Tim Roughgarden. Transaction Fee Mechanism Design. *Journal of the ACM*, 71(4):1–25, 2024. doi:10.1145/3674143.
- 56 Vikram Saraph and Maurice Herlihy. An Empirical Study of Speculative Concurrency in Ethereum Smart Contracts. *arXiv preprint arXiv:1901.01376*, 2019. arXiv:1901.01376.
- 57 Sei Labs. Sei protocol, 2023. Accessed: 2025-01-23. URL: <https://www.sei.io>.
- 58 Ilya Sergey and Aquinas Hobor. A Concurrent Perspective on Smart Contracts. In *International Conference on Financial Cryptography and Data Security*, pages 478–493. Springer, 2017. doi:10.1007/978-3-319-70278-0_30.
- 59 Elaine Shi, Hao Chung, and Ke Wu. What can Cryptography do for Decentralized Mechanism Design. *arXiv preprint*, 2022. arXiv:2209.14462.
- 60 Solana Labs. Solana, 2023. Accessed: 2025-01-23. URL: <https://solana.com>.
- 61 Wenhao Wang, Lulu Zhou, Aviv Yaish, Fan Zhang, Ben Fisch, and Benjamin Livshits. Mechanism Design for ZK-Rollup Prover Markets. *arXiv preprint arXiv:2404.06495*, 2024. doi:10.48550/arXiv.2404.06495.

- 62 Ke Wu, Elaine Shi, and Hao Chung. Maximizing Miner Revenue in Transaction Fee Mechanism Design. *arXiv preprint*, 2023. [arXiv:2302.12895](#).
- 63 Hong Xu and Baochun Li. A study of pricing for cloud resources. *SIGMETRICS Perform. Eval. Rev.*, 40(4):3–12, April 2013. [doi:10.1145/2479942.2479944](#).
- 64 Andrew Chi-Chih Yao. An Incentive Analysis of some Bitcoin Fee Designs. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2020. [doi:10.4230/LIPIcs.ICALP.2020.1](#).
- 65 An Zhang and Kunlong Zhang. Enabling Concurrency on Smart Contracts Using Multiversion Ordering. In *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint International Conference on Web and Big Data*, pages 425–439. Springer, 2018. [doi:10.1007/978-3-319-96893-3_32](#).