Sidechains vs Plasma vs Sharding

2019 Jun 12 See all posts

Special thanks to Jinglan Wang for review and feedback

One question that often comes up is: how exactly is sharding different from sidechains or Plasma? All three architectures seem to involve a hub-and-spoke architecture with a central "main chain" that serves as the consensus backbone of the system, and a set of "child" chains containing actual user-level transactions. Hashes from the child chains are usually periodically published into the main chain (sharded chains with no hub are theoretically possible but haven't been done so far; this article will not focus on them, but the arguments are similar). Given this fundamental similarity, why go with one approach over the others?

Distinguishing sidechains from Plasma is simple. Plasma chains are sidechains that have a non-custodial property: if there is any error in the Plasma chain, then the error can be detected, and users can safely exit the Plasma chain and prevent the attacker from doing any lasting damage. The only cost that users suffer is that they must wait for a challenge period and pay some higher transaction fees on the (non-scalable) base chain. Regular sidechains do not have this safety property, so they are less secure. However, designing Plasma chains is in many cases much harder, and one could argue that for many low-value applications the security is not worth the added complexity.

So what about Plasma versus sharding? The key technical difference has to do with the notion of **tight coupling**. Tight coupling is a property of sharding, but NOT a property of sidechains or Plasma, that says that the validity of the main chain ("beacon chain" in ethereum 2.0) is inseparable from the validity of the child chains. That is, a child chain block that specifies an invalid main chain block as a dependency is by definition invalid, and more importantly a main chain block that includes an invalid child chain block is by definition invalid.

In non-sharded blockchains, this idea that the canonical chain (ie. the chain that everyone accepts as representing the "real" history) is *by definition* fully available and valid also applies; for example in the case of Bitcoin and Ethereum one typically says that the canonical chain is the "longest valid chain" (or, more pedantically, the "heaviest valid and available chain"). In sharded blockchains, this idea that the canonical chain is the heaviest valid and available chain *by definition* also applies, with the validity and availability requirement applying to both the main chain and shard chains. The new challenge that a sharded system has, however, is that users have no way of fully verifying the validity and availability of any given chain *directly*, because there is too much data. The challenge of engineering sharded chains is to get around this limitation by giving users a maximally trustless and practical *indirect* means to verify which chains are fully available and valid, so that they can still determine which chain is canonical. In practice, this includes techniques like committees, SNARKs/STARKs, fisherman schemes and fraud and data availability proofs.

If a chain structure does not have this tight-coupling property, then it is arguably not a layer-1 sharding scheme, but rather a layer-2 system sitting on top of a non-scalable layer-1 chain. Plasma is not a tightly-coupled system: an invalid Plasma block absolutely can have its header be committed into the main Ethereum chain, because the Ethereum base layer has no idea that it represents an invalid Plasma block, or even that it represents a Plasma block at all; all that it sees is a transaction containing a small piece of data. However, the consequences of a single Plasma chain failing are localized to within that Plasma chain.

Sharding Try really hard to ensure total validity/availability of every part of the system

Plasma Accept local faults but try to limit their consequences

However, if you try to analyze the process of *how* users perform the "indirect validation" procedure to determine if the chain they are looking at is fully valid and available without downloading and executing the whole thing, one can find more similarities with how Plasma works. For example, a common technique used to prevent availability issues is fishermen: if a node sees a given piece of a block as unavailable, it can publish a challenge claiming this, creating a time period within which anyone can publish that piece of data. If a block goes unchallenged for long enough, the blocks and

all blocks that cite it as a dependency can be reverted. This seems fundamentally similar to Plasma, where if a block is unavailable users can publish a message to the main chain to exit their state in response. Both techniques eventually buckle under pressure in the same way: if there are too many false challenges in a sharded system, then users cannot keep track of whether or not all of the availability challenges have been answered, and if there are too many availability challenges in a Plasma system then the main chain could get overwhelmed as the exits fill up the chain's block size limit. In both cases, it seems like there's a system that has nominally $O(C^2)$ scalability (where C is the computing power of one node) but where scalability falls to $O(C)$ in the event of an attack. However, sharding has more defenses against this.

First of all, modern sharded designs use randomly sampled committees, so one cannot easily dominate even one committee enough to produce a fake block unless one has a large portion (perhaps $>\frac{1}{3}$) of the entire validator set of the chain. Second, there are better strategies to handling data availability than fishermen: data availability proofs. In a scheme using data availability proofs, if a block is *unavailable*, then clients' data availability checks will fail and clients will see that block as unavailable. If the block is *invalid*, then even a single fraud proof will convince them of this fact for an entire block. An $O(1)$-sized fraud proof can convince a client of the invalidity of an $O(C)$-sized block, and so $O(C)$ data suffices to convince a client of the invalidity of $O(C^2)$ data (this is in the worst case where the client is dealing with N sister blocks all with the same parent of which only one is valid; in more likely cases, one single fraud proof suffices to prove invalidity of an entire invalid chain). Hence, sharded systems are theoretically less vulnerable to being overwhelmed by denial-of-service attacks than Plasma chains.

Second, sharded chains provide stronger guarantees in the face of large and majority attackers (with more than $\frac{1}{3}$ or even $\frac{1}{2}$ of the validator set). A Plasma chain can always be successfully attacked by a 51% attack on the main chain that censors exits; a sharded chain cannot. This is because data availability proofs and fraud proofs happen *inside the client*, rather than *inside the chain*, so they cannot be censored by 51% attacks. Third, the defenses provided by sharded chains are easier to generalize; Plasma's model of exits requires state to be separated into discrete pieces each of which is in the interest of any single actor to maintain, whereas sharded chains relying on data availability proofs, fraud proofs, fishermen and random sampling are theoretically universal.

So there really is a large difference between validity and availability guarantees that are provided at layer 2, which are limited and more complex as they require explicit reasoning about incentives and which party has an interest in which pieces of state, and guarantees that are provided by a layer 1 system that is committed to fully satisfying them.

But Plasma chains also have large advantages too. First, they can be iterated and new designs can be implemented more quickly, as each Plasma chain can be deployed separately without coordinating the rest of the ecosystem. Second, sharding is inherently more fragile, as it attempts to guarantee absolute and total availability and validity of some quantity of data, and this quantity must be set in the protocol; too little, and the system has less scalability than it could have had, too much, and the entire system risks breaking. The maximum safe level of scalability also depends on the number of users of the system, which is an unpredictable variable. Plasma chains, on the other hand, allow different users to make different tradeoffs in this regard, and allow users to adjust more flexibly to changes in circumstances.

Single-operator Plasma chains can also be used to offer more privacy than sharded systems, where all data is public. Even where privacy is not desired, they are potentially more efficient, because the total data availability requirement of sharded systems requires a large extra level of redundancy as a safety margin. In Plasma systems, on the other hand, data requirements for each piece of data can be minimized, to the point where in the long term each individual piece of data may only need to be replicated a few times, rather than a thousand times as is the case in sharded systems.

Hence, in the long term, a hybrid system where a sharded base layer exists, and Plasma chains exist on top of it to provide further scalability, seems like the most likely approach, more able to serve different groups' of users need than sole reliance on one strategy or the other. And it is unfortunately *not* the case that at a sufficient level of advancement Plasma and sharding collapse into the same design; the two are in some key ways irreducibly different (eg. the data availability checks made by clients in sharded systems *cannot* be moved to the main chain in Plasma because these checks only work if they are done subjectively and based on private information). But both scalability solutions (as well as state channels!) have a bright future ahead of them.