In general, a “phase 4” tightly coupled sharding chain works in the following way:

1. Collations get produced (via some mechanism)
2. Some set (“committee”) of M

randomly sampled notaries verifies the data availability of the collations, and somehow votes that the collations are available.

1. The collation header plus committee votes backing it up get included in the main chain (the main chain validator verifies the votes, and may also verify a data availability proof of the collation itself), at which point the main chain and that collation are “tightly coupled”: if, for some reason, the collation is unavailable despite the votes being there, then that entire main chain block must be rejected by clients.

So the pipeline is:

In phases 1-3, we lack tight coupling, but instead we can replace this with any kind of in-protocol mechanism where a majority of validators can roll back a chain that contains a block that they deem to be unavailable.

Consider in the abstract the following variables:

- N

: the number of shards

- M

: the size of a random sample

- T

: time between collations on each shard (ie. block time in the shards)

- C

: the size of a collation

- R

: the number of distinct collations that a notary message votes for

- F

: time to internal finality (ie. tight coupling)

In the sharding 1.1 spec, these variables are as follows:

- N

: 100

- M

: ~25

- T

: 70 seconds

- C

: 1 MB

- R

: ~25

- F

: 70 seconds * ~25 ~= 30 minutes

Note that the sharding 1.1 spec does not have an explicit “committee size”, but it is there implicitly, as it’s expected that when a collator gets assigned to a shard, they only check the last M collations in the chain, with an attempt to cryptoeconomically enforce a hard minimum of 25. Hence, if a collation gets 25 confirmations, it does not really get checked anymore, so $M = 25$

. $R = 25$

because forking is voting; the act of making a collation on top of a head signifies approval of the head and its 25 most recent ancestors. Finality is subjective, but since there is no further verification after 25 collations, saying that it comes after 25 periods is as reasonable as anything else.

But there are forms of voting other than forking; for example, collations could all be independent of each other, except that each collation could contain a bit field expressing which of the 25 most recent collations that collator thinks are available. We could have a scheme where there is no forking of any kind; for a collation to even go into the main chain, it must already come with 25 votes supporting it, and at that point it's finalized. One goal of this post will be to try to analyze all schemes of this kind.

We can calculate the main chain overhead as:

$\frac{N * M}{T * R}$

This is justified as follows:

- N is the number of shards, and M

is the committee size per shard. For every collation header there would thus be M

signatures, so with N

collation headers that's $N * M$

on-chain overhead.

- The above happens once every T

seconds.

- The above can be cut down if one signature represents multiple things, hence we can add R

into the denominator.

Another important kind of overhead is notary burst overhead

: when a notary gets called to vote on one or more collations, they have some deadline within which they have to submit a signature, and they need to download all the data to check availability within that time. We can calculate notary burst overhead as:

$\frac{C * R}{F}$

This can be justified as follows:

- C

is the size of each collation. R

is the number of collations they have to check.

- Internal finality (ie. tight coupling) requires seeing the notary signatures, so whatever the length of time notaries are given to download the data, internal finality requires waiting for at least that.

Within this framework, we can calculate the efficiency of a few schemes:

- The 1.1 forkful model: main chain overhead = $\frac{100 * 25}{70 * 25} \approx 1.41$

, collator burst overhead = $\frac{10^6 * 25}{1750} \approx 14286$

bytes per second. Notice that in reality, the forkful model is actually much less efficient than this ideal because the lookahead is only 3 periods and not 25, so collator burst overhead is $\frac{10^6 * 25}{210} \approx 119047$

bytes per second.

- Increase collation block time to 1750 seconds, collation size to 25 MB, require collations to be ratified by committee or else rejected immediately. Main chain overhead = $\frac{100 * 25}{1750} \approx 1.41$

, collator burst overhead = $\frac{25 * 10^6}{1750} \approx 14286$

bytes per second.

Notice that the two models seem to give exactly the same costs, and exactly the same finality time (1750 seconds), and the second is much simpler. So why even consider the first? The answer is: a model based on smaller collation sizes gives faster indication of partial confirmation. 25 confirmations is as final as it gets, but even 3 confirmations usually gives a lot of confidence that a transaction will be included for good.

If we look at the equations for calculating overhead, we see a few clear tradeoffs:

- The 1:1 tradeoff for main chain overhead versus notary overhead. You can halve main chain overhead at the cost of doubling notary overhead by doubling R

.

- The 1:1 tradeoff between notary burst overhead and finality time. You can halve notary burst overhead by doubling finality time (or alternatively, you can shift the savings to main chain overhead by also doubling R

).

- You can always reduce both C

and T

by an equal proportion, and cancel out the impact on the main chain and notary burst overhead by increasing R

by the same proportion.

This more generally shows that we can use techniques to make one signature mean multiple things to reduce T

, and try to provide faster partial information. But are there techniques other than forking to do it, and are they better?

First of all, let us switch from forking to a model where collation proposal and notarization are separate functions; that is, every period, a collation header gets proposed, but then a separate set of notaries engages in voting to finalize them. Collation proposers could, for instance, be randomly sampled from a set that exists within a particular shard, thereby allowing them to hold the state of that shard, and removing the need for collator-proposer games. Let us also switch from a forking-as-voting model to an explicit model: every period, k

notaries are selected per shard, and these notaries must attempt to download the last \frac{M}{k}

collations and deliver a verdict on the availability of each through a bitfield (ie. a string like 1111011100111111011101). After a collation is created, after p

periods, there would be p * k

votes on the collation, delivering evidence of partial confirmation. This allows us to move away from one

unit of voting per period per shard, and instead opens up the full parameter space between k = 1

(similar to the 1.1 forkful model) and k = M

(a collation gets ratified within one round, and ratification is done before the next collation is created).

Notarizations can be cryptoeconomic: you get more rewards by voting 1 on more collations that ultimately get accepted, but if you vote for a collation you are responsible for providing any specific chunk of the collation body is challenged, and if you can’t do that you lose your deposit.

To achieve optimally low notary burst overhead, the lookahead needs to be equal to \frac{M}{k}

, so notaries can start downloading and checking collations as they come. However, there is a clever way to improve security of this scheme against semi-adaptive adversaries: instead of checking availability vertically

across a range of period numbers on one shard, notaries check diagonally

, checking a collation from a different shard in each period. The randomness that determines which shard a notary checks during each period can be delayed until that period, achieving zero lookahead transparency, making the scheme secure under honest majority except for all but the most adaptive adversaries who can launch attacks on specific nodes within a single period.

To summarize, in this kind of diagonal scheme, a notary would be called upon during period p

, and required to verify a collation from some shard during every period from p

to p+\frac{M}{k}

. During the last period, the notary would submit a signature that contains that notary’s opinion on every collation within that range as a bitfield. This allows us to:

- Remove proposer/collator separation, as its function is now replaced by proposer/notary separation/
- Achieve even higher notary burst overhead efficiency (or alternatively, shift some of the gains into increasing M

from 25 to a safer 100-200)

- Make it known which notaries are active in general \frac{M}{k}

shards in advance, but not make it known which

shard a given notary will be notarizing in some period until that period begins.

- Avoid all concerns about using transaction fees or bids to bribe collators to make collations off the main chain, as the voting function is done by notaries
- Remove the complexity of having a “forking chain inside a forking chain”
- Still have basically the same main chain overhead as before
- Still have the same level of information about partial confirmations as before