Availability and validation for Polkadot parachains

Jeffrey Burdges

Web 3 Foundation

4 May 2020

Sharding: Convince n parties some state evolved correctly, while revealing each update to only $k < n$ checking parties.

These k checkers are trusted authorities in more centralised designs, but..

We keep "authority" more narrowly prescribed, ala slsahable stake, by treat everyone relatively adversarially.

Sharding: Convince n parties some state evolved correctly, while revealing each update to only $k < n$ checking parties.

These k checkers are trusted authorities in more centralised designs, but..

We keep "authority" more narrowly prescribed, ala slsahable stake, by treat everyone relatively adversarially.

Naive Answer: Select the k checkers randomly, but make k large ByzCoin ($k = 300$ or 500?), ETH2 ($k = 100$?)

Those k checkers know themselves in advance, which increases their authority. It still works for k large enough but..

Sharding: Convince n parties some state evolved correctly, while revealing each update to only $k < n$ checking parties.

Shared Randomness Problem:
Can anyone manipulate who becomes a checker?

Adaptivity Problem:
Can anyone interfear with the checkers operation?

We cannot observe when the network behaves adversarially, so one critical case is:
Availability Problem:
Can honest checkers obtain the update/block?

Sharding: Convince n parties some state evolved correctly, while revealing each update to only $k < n$ checking parties.

Shared Randomness Problem:
Can anyone manipulate who becomes a checker?

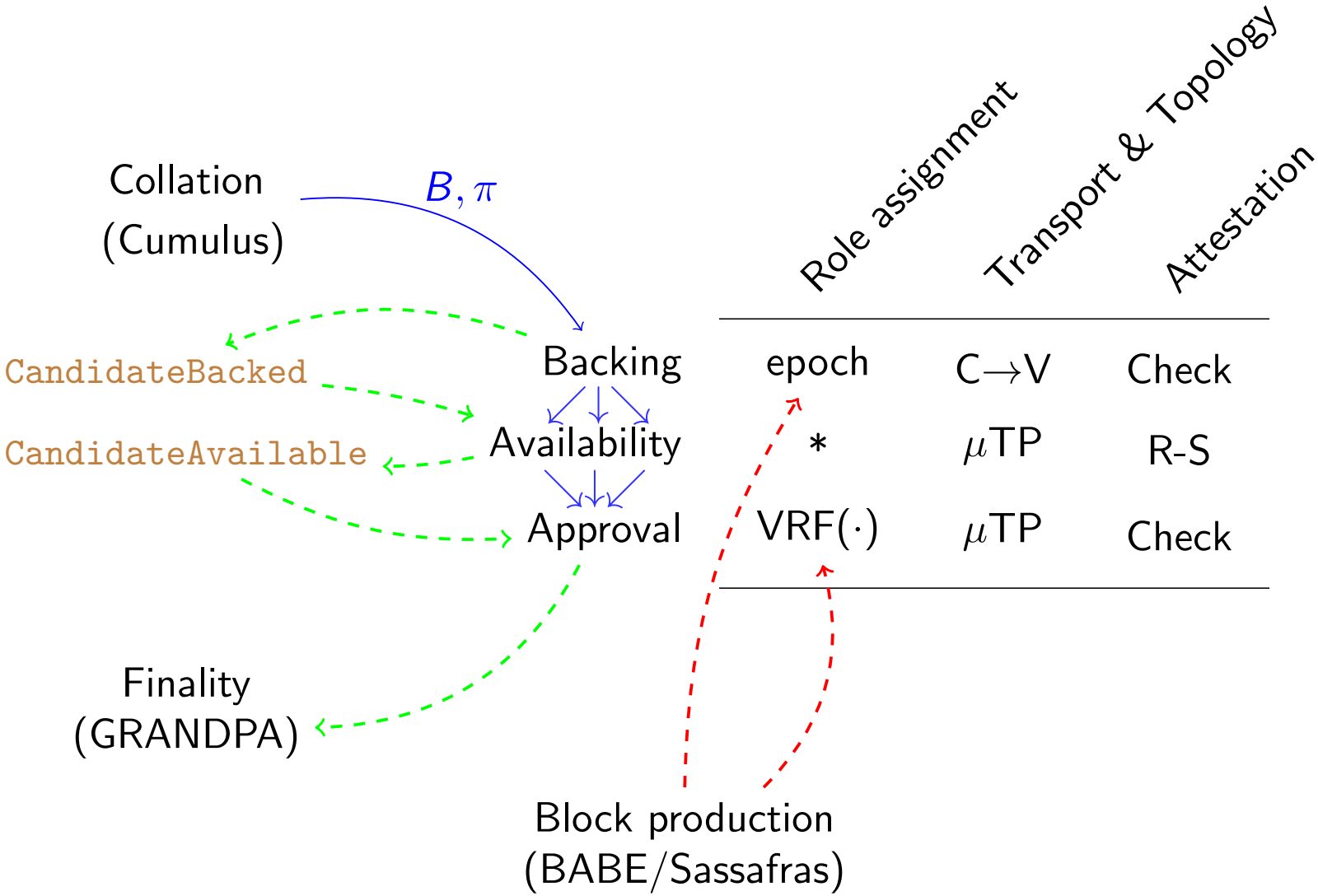
Adaptivity Problem:
Can anyone interfear with the checkers operation?

We cannot observe when the network behaves adversarially, so one critical case is:
Availability Problem:
Can honest checkers obtain the update/block?

Could we avoid the k checkers even knowing themelves in advance?

Problem:
How would you convince n people that you would give them data without actually sending them the data?

Answer:
Reed-Solomon erasure codes represent your data by the coefficients of a polynomial $p(x)$ of degree d over some finite field and produces code symbols by evaluating $p(0), p(1), \ldots, p(dn/f)$. You give symbols $p(id/f), \ldots, p((i+1)d/f - 1)$ to the ith person. Any f people could now recompute $p(x)$, and hence your data, by doing Lagrange interpolation with their combined pieces, so they can merely vote.


Collation
(Cumulus)
B, π
CandidateBacked
CandidateAvailable
Backing
Availability
Approval
Finality
(GRANDPA)
Block production
(BABE/Sassafras)
Role assignment
Transport & Topology
Attestation
epoch
C→V
Check
*
μTP
R-S
VRF(·)
μTP
Check

Backing Checks

Roughly like ByzCoin or ETH2, except small groups.

Group assignment:

```
use rand::{SeedableRng, seq::SliceRandom};
let seed = ... epoch.randomness ...;
let mut indexes = (0..num_validators).collect::<Vec<_>>();
indexes.shuffle(rand_chacha::ChaChaRng::from_seed(seed));
indexes.chunks(num_backing_per_parachain)
```

Transport & Topology:

Collator selects relay chain parent, makes the PoV block B by attaching witness data, and sends it to assigned validators.

Backing Checks

Assume $3f + 1$ validators. Reed-Solomon erasure code the PoV block B into $3f + 1$ pieces aka "symbols" so $f + 1$ pieces suffices for reconstruction. Compute the Merklee root m_B of the pieces and make authenticated pieces with attached witness for m_B.

Attestation:

Assigned validators check. If okay, they create and sign the candidate receipt, which contains m_B, and gossip everything among themselves.

If enough backers, they gossip *only* the candidate receipt to all validators.

We'd love the relay chain to avoid working on conflicting candidates, so any further work depends on the attested candidate receipt being included in a `CandidateBacked` transaction.

Availability

Assignment:

Send for blocks you backed. Receive for everything else.

Transport & Topology:

We've one unique erasure coded piece for each validator, so send the ith piece to the ith validator, assuming a `CandidateBacked`.

Reinvent BitTorrent but sans tracker!

Attestation:

Attest on-chain with `HaveCandidatePiece` for pieces you receive: Sign $(H(R), X)$ where X is a bitfield over `CandidateBacked` in R.

We declare `CandidateAvailable` once $2f + 1$ validators say so. In fact, parachain candidates "run" in the block R where they reach this goal, so they receive and send XCMP messages here.

VRFs

Verifiable random functions

(VRFs) are pseudo-random functions (PRFs) that provide publicly verifiable proof of outputs' correctness.

VRF protocols consist of

- a commit that certifies some public key with stake, and
- unlimited "reveals" aka outputs under arbitrary inputs.

VRFs are signatures with uniqueness, but you maximize messages for signatures.

You want relative uniqueness from "reveals", for which you minimize VRF inputs.

Approval

Assignment:

All validators assign themselves with *VRFs conditions*, which only they know until they announce themselves.

Transport & Topology:

If assigned, you fetch erasure coded pieces of $f$ish other validators.

Increases our BitTorrent resemblance, but still without tracker.

Attestation:

If block passes then gossip attestation for inclusion on on-chain.
Grandpa only votes for forks with *enough* attestations in children.

Rule: If validators disagree on validity then all validators check it.

Approval: VRF conditions

Consider the `CandidateAvailable` $\mathcal{B} = \{B_1, \ldots, B_n\}$ announced in a relay chain block R. After this, each validator V gossips VRF signatures $\sigma := \mathrm{VRF}_V(\texttt{input})$ that determine which B_i that V checks.

Minimal: $i \leftarrow \mathrm{VRF}_V\left(R.\texttt{praos\_vrf.out}() +\!\!+ \texttt{no\_shows\_tranche}\right)$
Very few choices, which minimizes adversaries control.
Very efficient with one VRF determining all assignments.
But leaks under relay chain equivocations!

Unleaked: $\texttt{delay\_tranche}_i \leftarrow \mathrm{VRF}_V\left(H(B_i)\right)$
Adversaries choose $H(B_i)$ and their revealed VRFs.
Less efficient with one VRF per assignment.
But revealed only for $B_i \neq B_i'$ in relay chain equivocations.