Cardano Disaster Recovery Plan

Deliverable CDR-D1

Christian Badertscher, Nicholas Clarke, Jared Corduan, Duncan Coutts, Peter Gaži, Philipp Kant, Aggelos Kiayias, Alexander Russell

Project: Cardano

Type: Deliverable
Due Date: 5$^{\text{th}}$ May 2021

Responsible team: IOHK Research and Engineering
Editor: Philipp Kant, IOHK
Team Leader: Aggelos Kiayias, IOHK

Version 1.0
5$^{\text{th}}$ May 2021

Dissemination Level		
PU	Public	√
CO	Confidential, only for company distribution	
DR	Draft, not for general circulation	

Contents

Change Log

Rev.	Date	Who	Team	What
1	2021-03-10	PK	IOHK	Transcribed draft to LaTeX
2	2021-05-03	MW	IOHK	Incorporated AK's changes
3	2021-05-05	PK	IOHK	Set dissemination level to public

Cardano Disaster Recovery Plan

An IOHK technical report

May 5, 2021

The version of the Ouroboros protocol used in the current implementation of Cardano operates as expected (i.e., provides network-wide consistency and liveness) as long as two "disaster" events do not occur. These events are extremely unlikely to occur in the actual protocol deployment, but they may occur in standalone test deployments or under extremely unlikely real world circumstances (some examples are given below). This document describes these events in detail and the corresponding mitigating actions that need to be undertaken should they occur.

1 Long-lived network-wide partitions

1.1 Scenario

We have a network partition, where a cluster of nodes are not connected to the main network. The partition persists long enough that the cluster produces more than k blocks, and nodes in the cluster add them to their chain. After the partition ends, those nodes will not automatically revert to the main chain, since that would require them to switch to a fork that is deeper than k blocks and the current implementation of the Ouroboros protocol does not accommodate such re-organization. We note that for the Cardano mainnet parameterization, we have $k = 2160$, 1-second slots and $f = 1/20$, hence producing k blocks by a minority cluster would take at least one day. We want to reconcile the nodes in the cluster with the network.

1.2 Manual mitigation

We shall add functionality to the node that allows manually switching to a deep fork. Specifically, we will implement a new mode, where an operator can specify a block header hash/slot pair, `lastGood`, when starting the node. The node shall then keep its own chain only until it reaches the block with header hash `lastGood`, and subsequently try to sync with the network.

Operators of nodes in the cluster can then, independently of one another, start their nodes in this mode, selecting peers from the main network, in order to recover. Node operators need to make sure to select peers from the main network, to avoid re-syncing to the minority partition.

The parameter `lastGood` can be determined by observing the chains of the main network and the cluster (for example, via an explorer).

1.2.1 Additional Measures: Recovering Transactions from the Minority Partition

Discarding a long fork will potentially lose a lot of transactions. Some of those might not be compatible with the main chain, but if the partition also partitions users of the network, it is likely that many of the transactions from the discarded fork can also be inserted in the main chain.

In order to preserve those transactions from the minority partition that are also valid on the main chain, we shall produce a tool to read the transactions from the discarded blocks, and resubmit them to the network after the partition has been resolved.

1.2.2 Limitations

The solution naturally requires that the operators of nodes in the cluster *want* to rejoin the main chain. In case of a network split in two roughly equal parts, it can be hard to agree on which part is considered to be the main chain. How to reach such a social consensus is beyond the scope of this technical disaster recovery plan.

1.2.3 Workaround

In case the solution above is not (yet) implemented, it is also a viable alternative for operators of nodes in the cluster to wipe their nodes' state, and restart them, connecting them to the main network. In that case, the nodes would have to resync the whole chain. Again, operators should explicitly pick peers from the main network, given that during a transition period there may still be many nodes in the network that are following the other fork.

1.3 Automated Mitigation

A new version of Ouroboros is currently under development that automatically recovers from network-wide partitions of indefinite length.

2 Long-lived Global Outage

2.1 Scenario

Under normal conditions, we have about one block per $1/f$ slots (one block per 20 seconds). The current Ouroboros implementation requires to have at least one block in $3k/f$ slots which is a 36-hour window under current Cardano mainnet parametrization. A violation of this requirement is extremely unlikely -– block production would have to stop for 36 hours in the whole network. Events under which block production could stop for such a long time are extremely unlikely and include:

1. There is a software fault in the node implementation that affects all nodes simultaneously and prevents them from producing blocks. Monitoring would reveal this immediately, and developers would work on fixing this fault, but if producing the fix and deploying it on a subset of block producing nodes takes longer than 36 hours, the network would not automatically resume block production.

2. Carrington scale solar flares could take down the global internet, for much longer than 36 hours.

2.2 Manual mitigation

We shall add functionality that allows to start a node with a virtual clock, that starts at given time `tFail`, and is faster by a factor of `timeAcceleration` than the actual wall clock. In addition to those parameters, an operator will also provide a time `tRecover`, the wall clock start time of the recovery process.

Started in this mode, the node will

1. wait until the system wall clock is at `tRecover`

2. set an internal clock to `tFail`

3. discard any blocks that have been produced after `tFail`

4. set the speed of the internal clock to `timeAcceleration` times the speed of real time

5. produce, send, and accept empty blocks according to the slot leader schedule, but using the internal clock instead of the system clock

6. once the internal clock is no longer behind the system clock, resume operation normally, according to the system clock (perhaps via a shutdown and restart in the normal mode).

With that functionality implemented, repairing the chain would require manual coordination between a group of stake pool operators. The group of stake pool operators would agree on `tFail` (from historical data before the event), and `tRecover` and `timeAcceleration` (`tRecover` should be a time in the near future, so that all nodes can start at the same time, and `timeAcceleration` should be large enough that the recovery process does not take too long, but low enough that the system can still send blocks quickly enough. Should the latter be problematic, the stake pool operators can decide to use a more localised network than in normal operation, such as running all nodes on the same continent. Note that using empty blocks helps here as well: they are smaller, which helps propagating them through the network quickly). They would then all start their nodes with those parameters. The nodes would produce blocks for the time of the outage at a higher cadence, until they "catch up" with real time, and resume normally from there on.

When starting their node in recovery mode, stake pool operators will not be able to use their current operational certificate and KES key, as that key will already have evolved to the present, and so will not be considered valid when the clock is set back to `tFail`. Instead, they will discard their current operational certificate and KES key, and issue a new operational certificate and KES key, which is valid at `tFail`.

2.3 Automated mitigation

A new version of Ouroboros is currently under development that automatically recovers from global outages of indefinite length.