Coral CDN & Stellar Consensus Protocol

Surya Sonti [012535523]

Srinivas Prasad Sunnapu [012526241]

Yudhajith Belagodu [012548354]

Sampath Lakkaraju [011818781]

Vrushali

Abraham

Under Prof. Sithu Aung

CMPE 273 Fall 2018

Presentation Report

Table of Contents

Abstract

Coral CDN:

CoralCDN is a free and open content distribution network based around peer-to-peer technologies, comprised of a world-wide network of web proxies and nameservers. It allows a user to run a web site that offers high performance and meets huge demand.

Publishing through CoralCDN is as simple as appending a short string to the hostname of objects' URLs; a peer-to-peer DNS layer transparently redirects browsers to participating caching proxies, which in turn cooperate to minimize load on the source web server. CoralCDN proxies automatically replicate content as a side effect of users accessing it, improving its availability. Using modern peer-to-peer indexing techniques, CoralCDN will efficiently find a cached object if it exists *anywhere* in the network, requiring that it use the origin server only to initially fetch the object once.[1]

SCP:

Keywords: CoralCDN, Proxy, Network, DNS Mapping.

Operation

Coral CDN:

Coral avoided high loads on individual nodes through an indexing abstraction called a distributed sloppy hash table (DSHT). DSHTs create self-organizing clusters of nodes which fetch information from each other to avoid communicating with more distant or heavily-loaded servers. DSHT is described in the paper Sloppy hashing and self-organizing clusters.[2]

The *sloppy* hash table refers to the fact that Coral was made up of concentric rings of distributed hash tables (DHTs), each ring representing a wider and wider geographic range (or rather, ping range). The DHTs are composed of nodes all within some latency of each other (for example, a ring of nodes within 20 milliseconds of each other). It avoids hot spots (the 'sloppy' part) by only continuing to query progressively larger sized rings if they are not overburdened. In other words, if the two top-most rings are experiencing too much traffic, a node will just ping closer ones: when a node that is overloaded is reached, upward progression stops. This minimizes the occurrence of hot spots, with the disadvantage that knowledge of the system as a whole is reduced.

System Overview

Coral CDN:

There is step wise process that travels when client issues a request to coral cdn network as shown below

Coral - Resolving DNS

A client resolves a “Coralized” domain name (*e.g.*, of the form example.com. nyud.net) using CoralCDN nameservers. A Coral- CDN nameserver probes the client to determine its round-trip-time and uses this information to determine appropriate nameservers and proxies to return.

Coral - Processing HTTP client requests

The client sends an HTTP request for a Coralized URL to one of the returned proxies. If the proxy is caching the web object locally, it returns the object and the client is finished. Otherwise, the proxy attempts to find the object on another CoralCDN proxy.

Coral - Discovering cooperative-cached content

The proxy looks up the object’s URL in the Coral indexing layer.

Coral - Retrieving content

If Coral returns the address of a node caching the object, the proxy fetches the object from this node. Otherwise, the proxy downloads the object from the origin server example.com.

Coral - Serving content to clients

The proxy stores the web object to disk and returns it to the client browser.

Announcing cached content

The proxy stores a reference to itself in Coral, recording the fact that is now caching the URL.

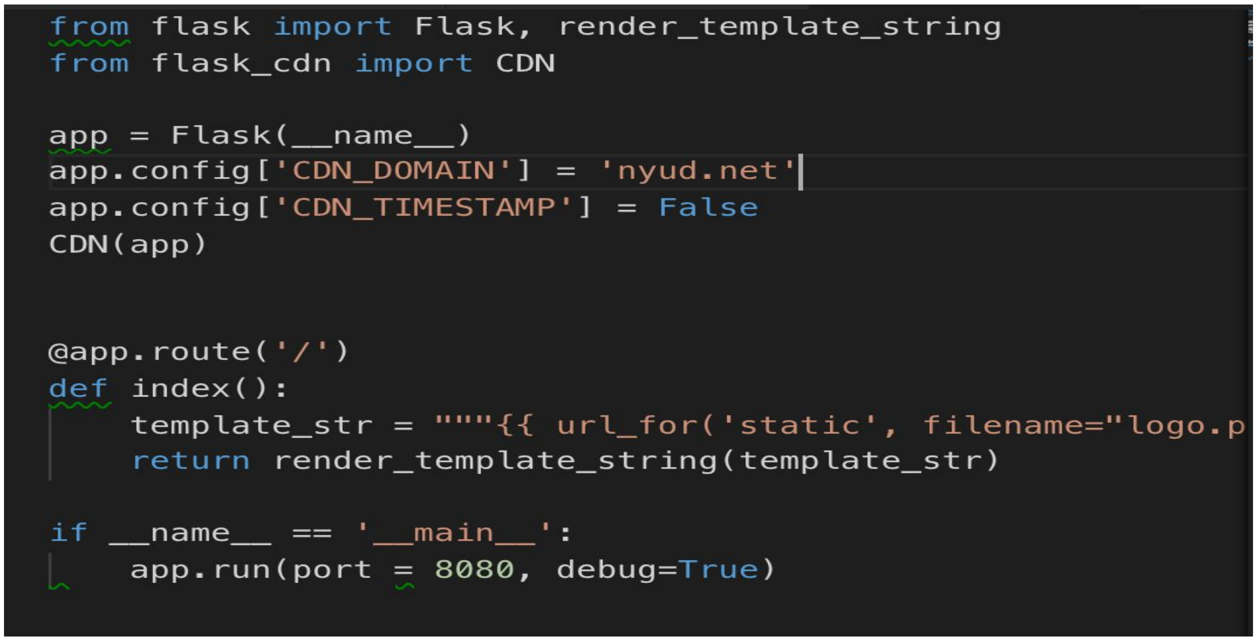
Programming Libraries Used:

Python with FlaskCDN

Alternatives to Existing CDN:

Amazon CDN-Cloud Front with Amazon S3

Screens:



```
from flask import Flask, render_template_string
from flask_cdn import CDN

app = Flask(__name__)
app.config['CDN_DOMAIN'] = 'nyud.net'
app.config['CDN_TIMESTAMP'] = False
CDN(app)


@app.route('/')
def index():
    template_str = """{{ url_for('static', filename="logo.p
    return render_template_string(template_str)

if __name__ == '__main__':
    app.run(port = 8080, debug=True)
```

INTRODUCTION

This paper is about the Stellar Consensus Protocol (SCP) which is a construction for the Federated Byzantine Agreement (FBA)- a model for consensus. FBA uses a concept known as quorum slices which are individual trust decisions made by nodes. These slices bind the system together similarly to the internet.

SCP, a construction for FBA is a model which allows open-membership promoting organic network growth. It lowers the barrier to entry thus opening up the financial systems to new participants, while also ensuring the security and integrity of the system.

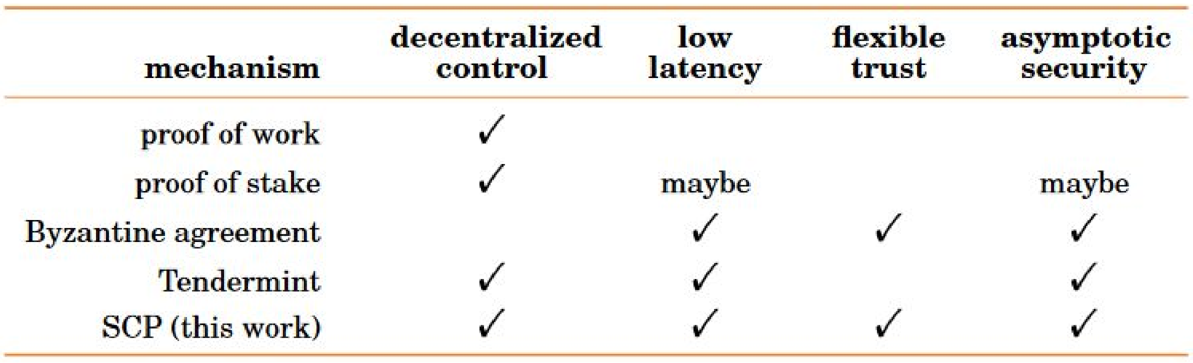
Financial transactions between organizations of different countries are very slow and high transaction costs. A worldwide financial network that supports organic growth and innovation is needed to solve these problems. The challenge for such a network is ensuring that the participants record transactions correctly. Having a low barrier to entry will cause users to lose trust. Also, for a worldwide network, a centralized authority won't be trusted by the providers to operate it. But using a decentralized system where the participants themselves ensure integrity of each other is a good alternative. This form of agreement hinges on a mechanism for worldwide consensus.

The FBA model is suited for worldwide consensus. Here each participant considers some others important. It waits until the majority of those to agree on any transaction before it agrees. Similarly, the participants it considers important will also have other participants they consider important and they will wait for their approval and so on. Eventually once enough of the network have agreed then the transaction will be considered settled.

SCP, a construction for FBA has four key properties:

1. Decentralized Control: Anyone can participate and no central authority.
2. Low Latency: Nodes can reach consensus at timescales similar to web payments i.e few seconds.
3. Flexible trust: Users will have the freedom to trust any combination of parties they see fit.
4. Asymptotic Security: Safety is ensured with digital signatures and hash families that require a vast amount of computing powers to crack.

SCP compared with other models:



mechanism	decentralized control	low latency	flexible trust	asymptotic security
proof of work	✓			
proof of stake	✓	maybe		maybe
Byzantine agreement		✓	✓	✓
Tendermint	✓	✓		✓
SCP (this work)	✓	✓	✓	✓

Federated Byzantine Agreement Systems (FBAS):

A Federated Byzantine Agreement (FBA) is similar to non-Federated Byzantine Agreements address the problems involved in updating a replicated state such as Certificate trees, Transaction Ledgers. Each update is identified as a SLOT, these slots are modified by nodes in the system. The FBA helps the nodes to agree on the updates to slots, which avoids contradictory or irreversible changes to the slots in the system. An FBA system achieves this by running a consensus protocol that ensures the nodes to agree on the slots and their current updates. The majority quorum-based agreements do not work on the FBA since malicious parties can out number the honest nodes and make the system independent. So, FBA implements a system known as quorum slices. We will define this in the next section.

For the basic overview of the working of an FBAS let us consider that a node **v** which needs to update **x** on the slot **i**. In an FBA system the node **v** can apply this update safely when it updates the respective slots which are dependent on **i** and believes that all the correctly functioning nodes in the system agree on the updated-on **i**. Once this is done, we can say that **v** has successfully externalized **x** on slot **i**. The node **v** does not consider all the nodes for agreement but the nodes which believes are correctly functioning.

Quorum slices:

As mentioned in an FBA the node which tries to externalize for a slot will need a sufficient set of nodes assertion for deeming the externalization safe. This set of sufficient nodes is called as a quorum slice or being precise a slice.

The following are the basic definition provided in [1]

Definition (FBAS):

" A federated Byzantine agreement system, or FBAS is a pair <V;Q> comprising a set of nodes V and a quorum function $Q : V \rightarrow 2^{2^{V}}$ *specifying one or more quorum slices for each node, where a node belongs to all of its own quorum slices—i.e.,* ∈

$\forall v \in V,\ \forall q \forall Q(v),\ v \in q$ *" [1]*

Definition (quorum):

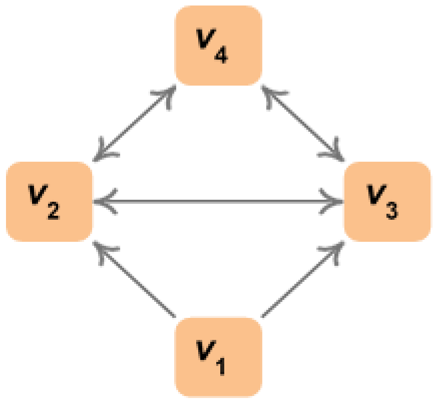
"A set of nodes U ⊆ V in FBAS <V;Q> is a quorum iff U =ø and U contains a slice for each member—i.e., (∀v ∈ U, ∃q ∈ Q(v) such that q ⊆ U)." [1]

A quorum is set of nodes that are required for reaching an agreement. A quorum slice is a set of nodes that are required for a node to reach an agreement. A quorum slice belongs to one node, whereas the quorum belongs to all the nodes that are present in the system. A node usually has more than one slice to overcome the issue of node failure.

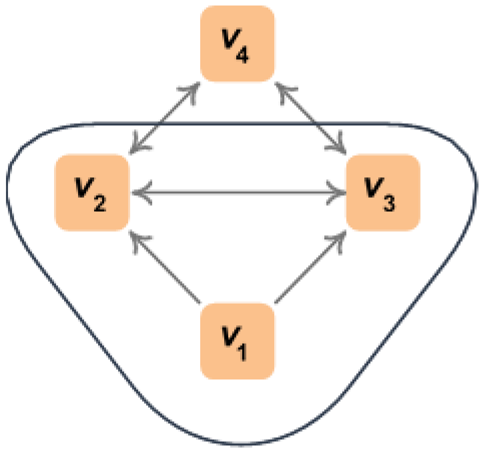
This quorum slice (or slices) of a particular node is left to the node itself. This is the main feature of the FBA that is different from the non-Federated systems. Since the quorum slice selection is left to the nodes itself, the resulting system is truly decentralized.

Examples:

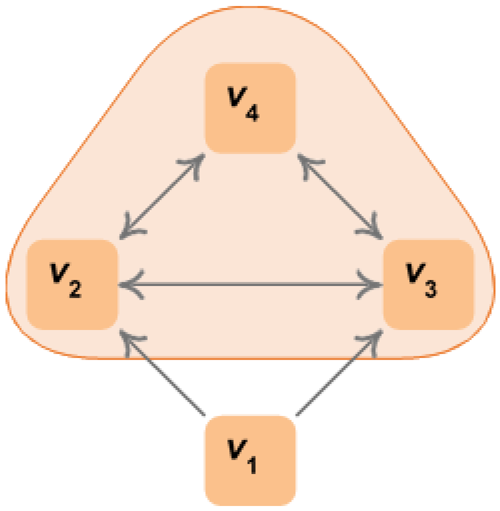
1) Consider the following system which represents an FBS. Each node in the system has a single slice, the slice of the node is represented by the arrows originating from the node.



In the case of the node v_1 its slice consists of v_1, v_2 ,v_3. So if the v_1 has to externalize a slot it needs the agreement from the nodes v_2 & v_3.

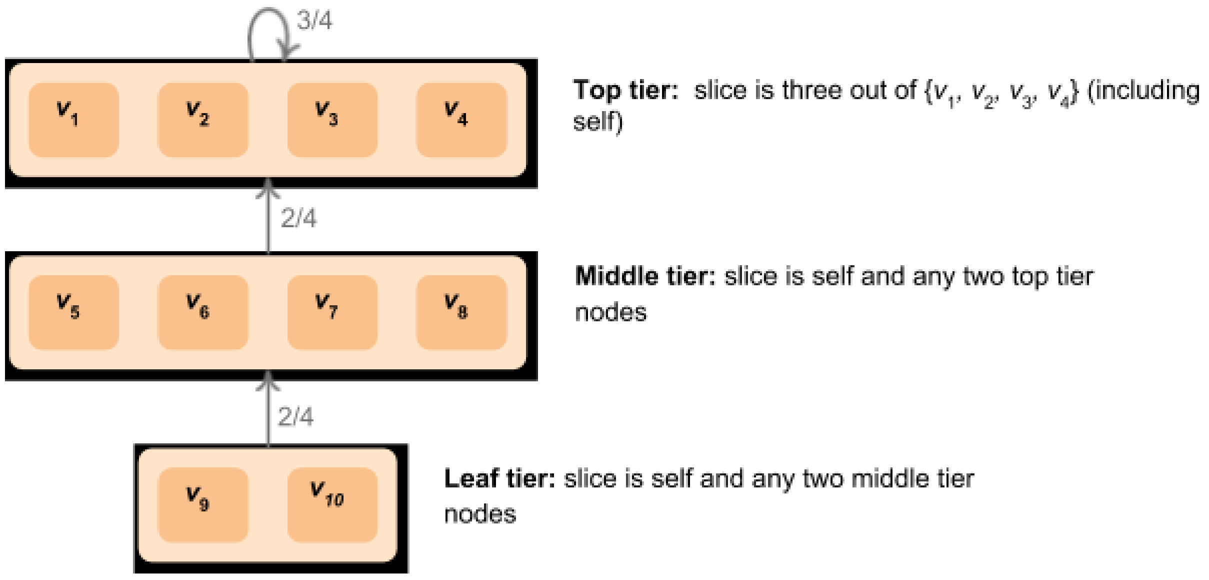


But v_2 , v_3 are both interdependent and are also dependent on v_4. So for v_2 , v_3 to agree the externalization v_4 needs to agree. In short if v_1 needs to externalize all nodes in the example v_1, v_2 , v_3 and v_4 needs to come to a consensus. This implies the quorum for v_1 is all the nodes in the example



While consider if v_2 needs to externalize a slot only v_2 ,v_3 and v_4 needs to come to a consensus, since no node is dependent on v_1. This group of nodes v_2, v_3 and v_4 can be considered as a quorum for v_2.

2) Consider a tiered which has different slices for individual nodes. This kind of system is only possible in a federated system.


3/4
v_1
v_2
v_3
v_4
Top tier: slice is three out of {v_1, v_2, v_3, v_4} (including self)
2/4
v_5
v_6
v_7
v_8
Middle tier: slice is self and any two top tier nodes
2/4
v_9
v_{10}
Leaf tier: slice is self and any two middle tier nodes

The top tier structure has nodes v_1 to v_4. This tier is similar to PBFT with the fault tolerance of f =1, which implies that the system can only tolerate a failure of one node. As mentioned before this implies a node from top tier need an acceptance from the other two nodes in the tier for reaching a consensus.

While the middle tier node has a slice of self and any two of the top tiers agreement and self for reaching a consensus. This is the slice for the node in the middle tier.

Lastly the same applies to the leaf tier nodes, which need two node's agreement and self to form a consensus. Consider that v_9 needs to externalize a slot. It requires two other nodes from the middle tier to agree before to reach a consensus. While the two nodes from the middle tier rely on the top tier node for consensus. So for a quorum consisting of the last node involves all the tiers.

Safety and liveness:

Not all nodes in a system behave properly, some may not respond properly, and some may behave maliciously. To make a distinction between these nodes the system categorizes nodes as either well-behaved or ill-behaved.

Well-behaved node:

- Chooses sensible quorum slices for consensus.
- obeys the protocol for every operation.
- Also, eventually responding to all requests.

An ill-behaved node:

- May be compromised, provided incorrect statements from the actual.
- May not respond to request due to failure.

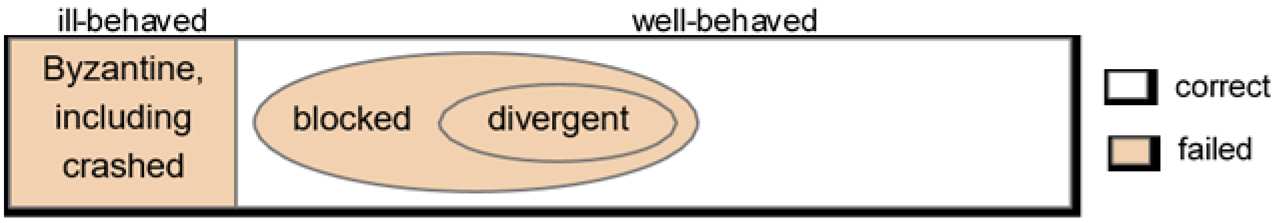
The main goal of Byzantine agreement is to ensure that well-behaved nodes externalize the same values despite the presence of such ill-behaved nodes. In order to achieve this, we need to make sure the nodes do not externalize incorrect or contradictory updates to a same slot. Secondly make sure that nodes can externalize values, which means that the nodes need not wait for consensus from any other nodes which are either in blocking or failed states. We define the following properties to achieve this state.

Definition (safety):

"A set of nodes in an FBAS enjoy safety if no two of them ever externalize different values for the same slot." [1]

Definition (liveness):

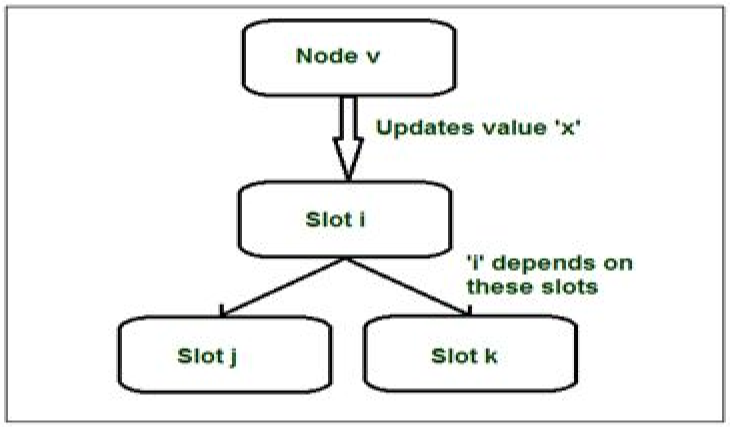
"A node in an FBAS enjoys liveness if it can externalize new values without the participation of any failed (including ill-behaved) nodes." [1]



If a node satisfies both safety and liveness, it is called a correct node. While all other nodes can be considered as failed. All the ill-behaved nodes are failed nodes. While most of the well-behaved nodes are correct nodes, some nodes suffer from either one of the properties from due to the presence of the ill-behaved nodes in their slices.

There are two types of failures, first is well behaved nodes which suffer liveness due to a node which is in failed or crashed state in its slice. This type of nodes is termed as *"blocked".* While the second type is where a well-behaved node is provided with wrong consensus by a node in its slice and suffer safety property. This type of nodes is termed as "*divergent"*. The entire state of nodes can be seen in the above figure. How the nodes enjoy safety and liveness will be explained in the next chapter.

Slots and externalisation



Slots:

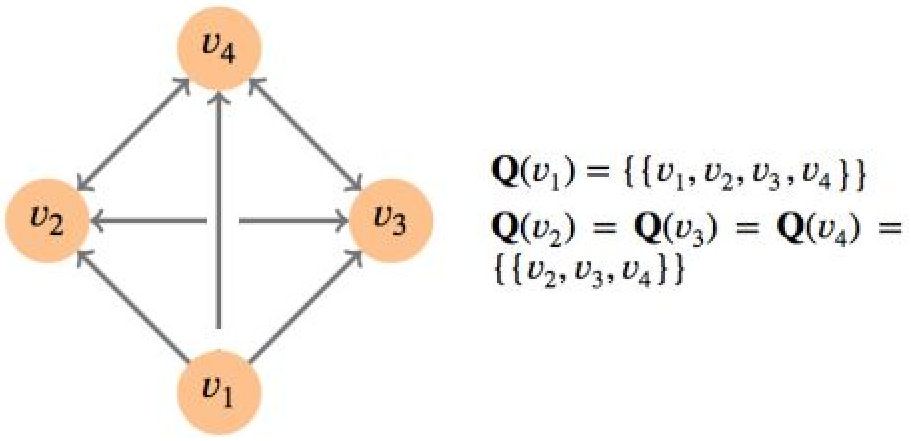
Every update that the nodes have to come on an agreement on is represented by a slot.

Externalisation:

Nodes coming on an consensus on a slot value and making it available for the world to use.

Quorum intersection :-

There has to be a node common between all the quorum slices for the system to guarantee that an agreement will be achieved.


v_4
v_2
v_3
v_1
$\mathbf{Q}(v_1) = \{\{v_1, v_2, v_3, v_4\}\}$
$\mathbf{Q}(v_2) = \mathbf{Q}(v_3) = \mathbf{Q}(v_4) = \{\{v_2, v_3, v_4\}\}$

V4: Is the intersection

Following system might not be able to achieve an agreement:

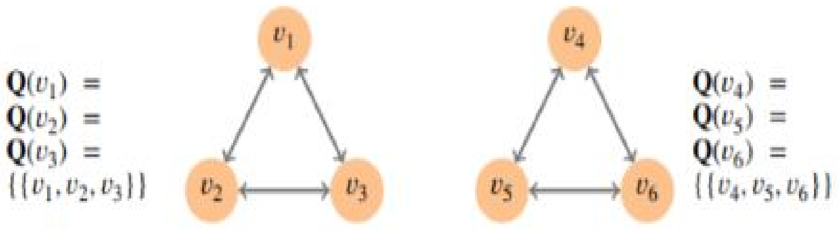

$\mathbf{Q}(v_1) = \mathbf{Q}(v_2) = \mathbf{Q}(v_3) = \{\{v_1, v_2, v_3\}\}$
v_1
v_2
v_3
v_4
v_5
v_6
$\mathbf{Q}(v_4) = \mathbf{Q}(v_5) = \mathbf{Q}(v_6) = \{\{v_4, v_5, v_6\}\}$

Fig. 6. FBAS lacking quorum intersection

As there is no agreement.

A faulty node as intersection between Quorums is as good as 2 different FBAS.

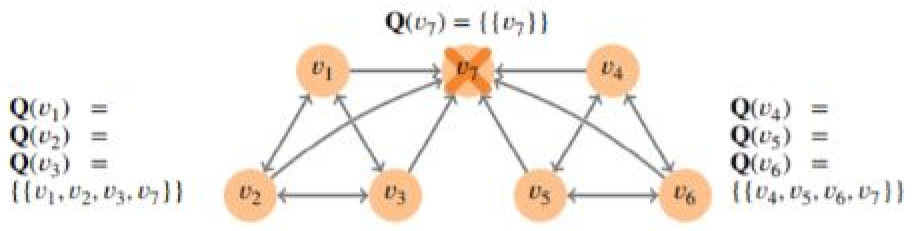

$\mathbf{Q}(v_7) = \{\{v_7\}\}$
$\mathbf{Q}(v_1) = \mathbf{Q}(v_2) = \mathbf{Q}(v_3) = \{\{v_1, v_2, v_3, v_7\}\}$
$\mathbf{Q}(v_4) = \mathbf{Q}(v_5) = \mathbf{Q}(v_6) = \{\{v_4, v_5, v_6, v_7\}\}$
v_1
v_2
v_3
v_4
v_5
v_6
v_7

Fig. 7. Ill-behaved node v_7 can undermine quorum intersection.

In short, FBAS (V,Q) can survive Byzantine failure by a set of nodes B ⊆ V iff (V,Q) enjoys quorum intersection after deleting the nodes in B from V and from all slices in Q.

V: Set of nodes

B: Faulty nodes that are a subset of V

Q: Quorum function

Dispensable sets:
Set of failed nodes in spite of which it is possible to guarantee both safety and liveness.

To prevent a misbehaving DSet from affecting the correctness of other nodes, two properties must hold:

1) Deleting the DSet cannot undermine quorum intersection
2) The DSet cannot deny other nodes a functioning quorum.

Federated voting :-

The federated voting technique that FBAS nodes can use to agree on a statement. At a high level, the process for agreeing on some statement involves nodes exchanging two sets of messages.From each node’s perspective, the two rounds of messages divide agreement on a statement into three phases: unknown, accepted, and confirmed.Initially status is completely unknown to a node could end up true, false, or even stuck in a permanently indeterminate state. If the first vote succeeds, may come to accept . No two intact nodes ever accept contradictory statements, so if is intact and accepts , then cannot be false. For two reasons, however, accepting does not suffice for to act on . First, the fact that accepted does not mean all intact nodes can; could be stuck for other nodes. Second, if is befouled, then accepting means nothing—may be false at intact nodes. Yet even if is befouled—which does not know—the system may still enjoy quorum intersection of well-behaved nodes, in which case, for optimal safety, needs greater assurance of 픺 Holding a second vote addresses both problems. If the second vote succeeds, moves to the confirmed phase in which it can finally deem true and act on it.

Ratify:-

In Ratify each node v can vote for a statement a - a must be consistent with past votes & accepted statements .

Definition (ratify):- A quorum U ratifies a statement a if every member of U votes for a. A node v ratifies a if v is a member of a quorum U that ratifies a.

Theorem: if well-behaved nodes enjoy quorum intersection despite ill-behaved nodes, won't ratify contradictory statements Problem: intact node v may be unable to ratify a after others do - v or nodes in v's slices might have voted against a, or - Some nodes that voted for a may subsequently have failed .

Accepting :-

Idea: say a v-blocking set claims system is a-valent - Either whole set lying (so v is not intact), or system in f act a-yalent .

Definition (accept) :-Node v accepts a statement a consistent with history iff either: 1. A quorum containing v each either voted for or accepted a, or 2. Each member of a v-blocking set claims to accept a.
#2 allows a node to accept a statement after voting against it.

Theorem: intact nodes cannot accept contradictory statements But two problems remain: 1. Still no guarantee all intact nodes can accept a statement 2. Suboptimal safety for non-intact nodes enjoying quorum intersection (v-blocking analogous to 11 - 1 in centralized system, not fs 1)

Confirmation:-

Idea: Hold a second vote on the fact that the first vote succeeded
Definition (confirm) :A quorum confirms a statement a by ratifying the statement "We accepted a:' A node confirms a iff it is in such a quorum.
Solves problem 2 (suboptimal safety) - Straight-up ratification provides optimal safety Solves problem 1 (intact nodes unable to accept) - Intact nodes may vote against accepted statements - Won't vote against the fact that those statements were accepted - Hence, the fact of acceptance is irrefutable Theorem: Once a single intact node confirms a statement, all intact nodes will eventually confirm it .

Federated voting outcomes :-

Federated voting has same possible outcomes as regular voting Apply the same reasoning as in centralized voting? - Premise was whole system couldn't fail; now failure is per node - Cannot assume correctness of quorums you don't belong to First-hand ratification now the only way to know system a-valent - How to agree on statement a even after voting against it? - How to know system has agreed on a

Stellar Consensus Protocol (SCP)

SCP consists of two sub-protocols: *nomination protocol* and *ballot protocol*.

Nomination Protocol

The nomination protocol produces candidate values for a slot. If run long enough, it eventually produces the same set of candidate values at every intact node, which means nodes can combine the candidate values in a deterministic way to produce a single composite value for the slot. There are two huge caveats, however. First, nodes have no way of knowing when the nomination protocol has reached the point of convergence. Second, even after convergence, ill-behaved nodes may be able to reset the nomination process a finite number of times. When nodes guess that the nomination protocol has converged, they execute the ballot protocol, which employs federated voting to commit and abort ballots associated with composite values.

The nomination protocol works by converging on a set of candidate values for a slot. Nodes then deterministically combine these candidates into a single composite value for the slot. Stellar network uses SCP to choose a set of transactions and a ledger timestamp for each slot. To combine candidate values, Stellar takes the union of their transaction sets and the maximum of their timestamps.

Properties

- Intact nodes can produce at least one candidate value
- set of possible candidate values stop growing

- if node considers x as candidate value, eventually every intact node will consider x to be a candidate value.

Caveats

- no-way of knowing when the nomication protocol has reached the point of convergence
- Even after convergence, ill-behaved nodes may be able to reset the nomination process a finite number of times.

Ballot Protocol

Once nodes have a composite value, they engage in the ballot protocol, though nomination may continue to update the composite value in parallel. A ballot *b* is a pair of the form $b = \langle n, x \rangle$, where $x \neq \perp$ is a value and *b* is a referendum on externalizing *x* for the slot in question. The value $n \geq 1$ is a counter to ensure higher ballot numbers are always available. Because at most one value can be chosen for a given slot, all committed and stuck ballots must contain the same value. Roughly speaking, this means commit statements are invalid if they conflict with lower-numbered unaborted ballots.

Properties

- at most one value chosen for a given slot (safety)
- all committed and stuck ballots must have same value (liveness)

States

prepared - commit *b* is valid to vote for only if *b* is confirmed prepared, which nodes ensure through federated voting on the corresponding abort statements.

externalized - To commit a ballot *b* and externalize its value *b.x*, SCP nodes first accept and confirm *b* is prepared, then accept and confirm commit *b*. Before the first intact node votes for commit *b*, the prepare step, through federated voting, ensures all intact nodes can eventually confirm *b* is prepared. When an intact node *v* accepts commit *b*, it means *b.x* will eventually be chosen.

LIMITATIONS

Even though SCP has good features it has some limitations. It can only guarantee safety when nodes choose adequate quorum slices. Since security depends on user choice, there is always a possibility of human error.

Even if the proper quorum slices are chosen and SCP guarantees safety, it cannot safeguard against some other security issues that may arise like a widely trusted node leveraging its position in the network to obtain information that may be used for unethical purposes.

SCP requires the participants to continuously be active in the network. If somehow all the nodes simultaneously and permanently leave then restarting consensus would require central co-ordination or human level agreement. However, if nodes return it is possible for the system to recover from the outage.

SUMMARY

Byzantine agreement has enabled distributed systems to achieve consensus with efficiency, security and flexibility. Bitcoin introduced the idea of decentralized consensus leading to many new systems. The FBA achieves decentralized consensus while preserving the traditional benefits of Byzantine agreement. The SCP is a construction for FBA that achieves optimal safety against ill-behaved participants.

References

http://www.coralcdn.org

https://en.wikipedia.org/wiki/Coral_Content_Distribution_Network

http://www.scs.stanford.edu/14au-cs244b/sched/readings/coral_experience.pdf

https://aws.amazon.com/cloudfront/

[1]http://delivery.acm.org/10.1145/950000/945470/p253-yin.pdf?ip=130.65.254.6&id=945470&acc=ACTIVE%20SERVICE&key=F26C2ADAC1542D74%2ED0BD0A8C52906328%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1544148842_18a27296359a7338132d3a6ac86238b2

https://www.coralcdn.org/docs/coral-iptps03.pdf