Viewstamped **R**eplication Revisited

Liskov, B, and Cowling, J. (2012)

MIT-CSAIL-TR-2012-021, 14 p.

Barbara Liskov

2008 Turing award - For contributions to practical and theoretical foundations of programming language and system design, especially related to data abstraction, fault tolerance, and distributed computing.

Liskov Substitution Principle

Original paper

Oki, B., and Liskov, B. (1988). **Viewstamped replication**: A new primary copy method to support highly-available distributed systems. *Proceedings of the Seventh annual ACM Symposium on Principles of Distributed Computing* (PODC), pp. 8-17.

VR

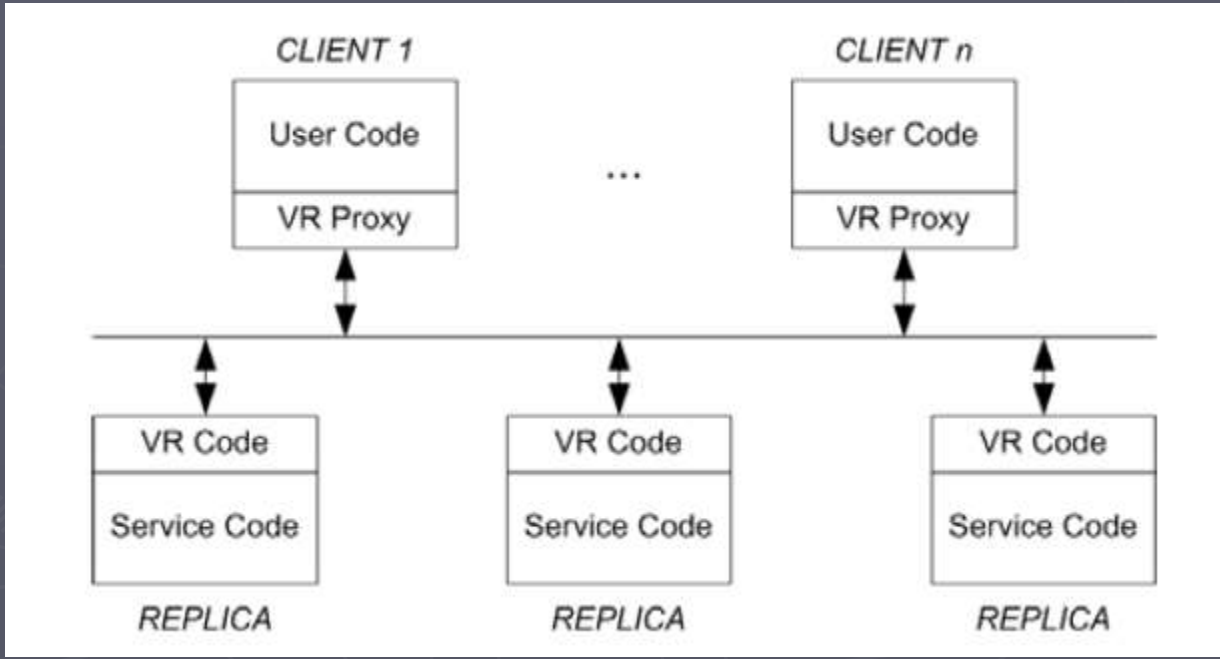
The goal was to support a replicated service, running on a number of replicas. The service maintains a state and makes that state accessible to a set of client machines.

It ensures reliability and availability when no more than a threshold of f replicas are faulty (crash failure model). It does this by using replica groups of size 2f+1.

VR

The rationale for needing this many replicas is as follows. We have to be able to carry out a request without f replicas participating, since those replicas might be crashed and unable to reply. However, it is possible that the f replicas we didn't hear from are merely slow to reply. In this case up to f of the replicas that processed the operation might fail after doing so. Therefore, we require that at least f+1 replicas participate in processing the operation, since this way we can guarantee that at least one replica both processed the request and didn't fail subsequently. Thus, the smallest replica group we can run with is of size 2f+1.

Architecture



Requeriments

1. Every operation executed by the replica group survives into the future in spite of up to f failures.
2. When client requests are made concurrently, they have to be executed in the same order by all replicas.

Primary-backup model

- Primary runs the protocol (normal operation)
  - It orders the requests (and tells the backups what to do)
  - It executes the client request and returns the result to the client, but it does this only after at least f+1 replicas (including itself) know about the request
- Backups monitor the primary
  - And do a view change if it fails (to mask the failure)

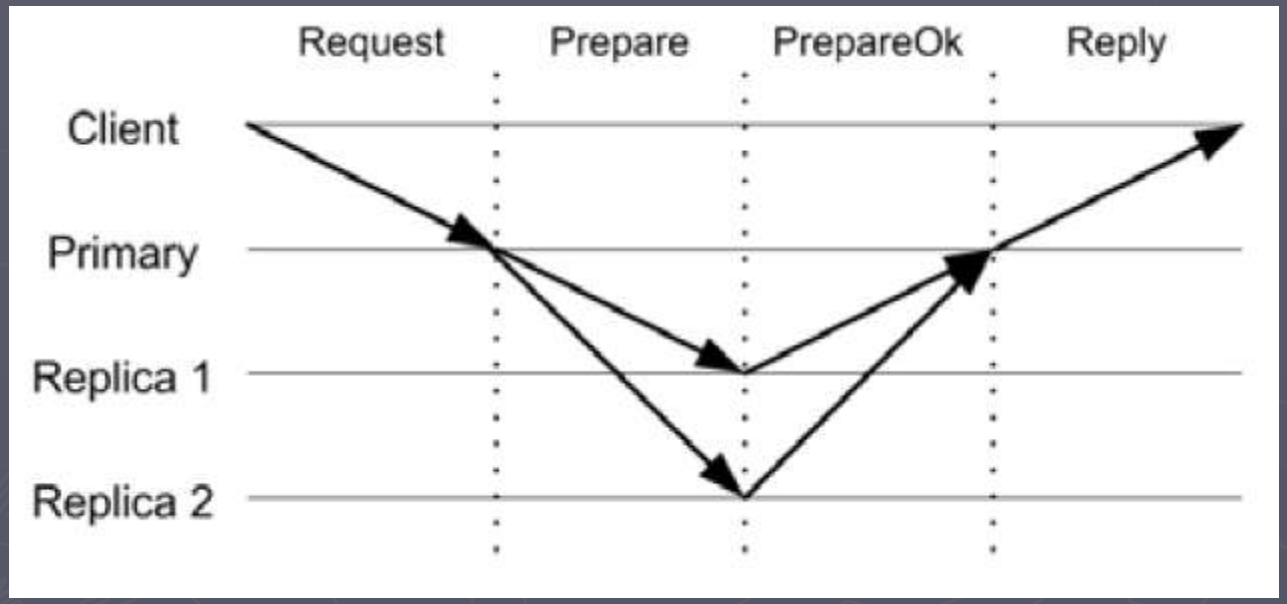
Primary-backup model

- Replicas move through a succession of views.
- The primary is chosen round-robin, starting with replica 1, as the system moves to new views. The other replicas are the backups.

State at each replica

- The *configuration*, i.e., the IP address and replica number for each of the $2f + 1$ replicas. The replicas are numbered 1 to $2f + 1$. Each replica also knows its own replica number.
- The current *view-number*, initially 0.
- The current *status*, either *normal*, *view-change*, or *recovering*.
- The *op-number* assigned to the most recently received request, initially 0.
- The *log*. This is an array containing *op-number* entries. The entries contain the requests that have been received so far in their assigned order.
- The *client-table*. This records for each client the number of its most recent request, plus, if the request has been executed, the result sent for that request.

Normal operation



Normal operation

1. The client sends a <REQUEST *op, c, s, v*> message to the primary, where *op* is the operation (with its arguments) the client wants to run, *c* is the *client-id*, *s* is the number assigned to the request, and *v* is the *view-number* known to the client.

Normal operation

2. When the primary receives the request, it advances *op-number* and adds the request to the end of the *log*. Then it sends a <PREPARE *m, v, n*> message to the other replicas, where *m* is the message it received from the client (*op,c,s*), *v* is the current *view-number*, and *n* is the *op-number* it assigned to the request.

A viewstamp is a pair <*view-number, op-number*>

Normal operation

3. Non-primary replicas process PREPARE messages in order: a replica won't accept a prepare with *op-number n* until it has entries for all earlier requests in its *log*. When a non-primary replica *i* receives a PREPARE message, it waits until it has entries in its *log* for all earlier requests (doing state transfer if necessary, to get the missing information). Then it adds the request to the end of its *log* and sends a <PREPAREOK *v, n, i*> message to the primary.

Normal operation

4. The primary waits for *f* PREPAREOK messages from different replicas; at this point it considers the operation to be ***committed***. Then, after it has executed all earlier operations (those assigned smaller *op-numbers*), the primary executes the operation by making an up-call to the service code, and sends a <REPLY *v, s, x*> message to the client; here *v* is the *view-number*, *s* is the number the client provided in the request, and *x* is the result of the up-call.

Quorums of at least *f*+1 replicas

Normal operation

5. At some point after the operation has committed, the primary informs the other replicas about the ***commit***. This need not be done immediately. A good time to send this information is on the next PREPARE message, as piggy-backed information; only the *op-number* of the most recent committed operation needs to be sent.

6. When a non-primary replica learns of a commit, it waits until it has executed all earlier operations and until it has the request in its *log*. Then it executes the operation by performing the up-call to the service code but does not send the reply to the client.

View change

1. A replica *i* that suspects the primary is faulty advances its *view-number*, sets its *status* to *view-change*, and sends a <DOVIEWCHANGE *v, l, k, i*> to the node that will be the primary of the next view. Here *v* is its *view-number*, *l* is the replica's log, and *k* is the *op-number* of the latest committed request known to the replica.

Viewstamps are used in the *view change* protocol

View change

2. When the new primary receives *f*+1 of these messages from different replicas, including itself, it selects as the new *log* the *most recent* of those it received in these messages: this is the one whose topmost entry has the largest **viewstamp**. It sets the *op-number* to that of the latest entry in the new *log*, changes its *status* to *normal*, and informs the other replicas of the completion of the view change by sending a <STARTVIEW *v, l, k*> message, where *l* is the new log and *k* is the *op-number* of the latest committed request it heard about in the responses.

View change

3. The new primary executes (in order) any committed operations that it hadn't executed previously, sends the replies to the clients, and starts accepting client requests.

View change

4. When other replicas receive the STARTVIEW message, they replace their *log* with that in the message, set their *op-number* to that of the latest entry in the log, set their *view-number* to the view number in the message, and change their *status* to *normal*. Then they continue the protocol for all operations not yet known to be committed by sending PREPAREOK messages for these operations.

Recovery

1. The recovering replica, *r*, sends a <RECOVERY *v, r*> message to all other replicas, where *v* is its starting *view-number*.

2. A replica *i* replies to a RECOVERY message only when its status is *normal*, its *view-number* ≥ *v*, and it is the primary of its view. In this case the replica sends a <RECOVERYRESPONSE *v, l, k, i*> message to the recovering replica, where *v* is its *view-number*, *l* is its *log*, and *k* is the latest committed request.

Recovery

3. The recovering replica waits to receive a RECOVERYRESPONSE message. Then it updates its state using the information in the message. It writes the new *view-number* to disk if it is larger than what was stored on disk previously, changes its status to *normal*, and the recovery protocol is complete. The replica then sends PREPAREOK messages for all uncommitted requests.

Reconfiguration

A reconfiguration is triggered by a special client request.

This request is run through the normal case protocol by the old group.

When the request commits, the system moves to a new epoch, in which responsibility for processing client requests shifts to the new group.

Reconfiguration

However, the new group cannot process client requests until its replicas are up to date: the new replicas must know all operations that committed in the previous epoch.

To get up to date they transfer state from the old replicas, which do not shut down until the state transfer is complete.

VR and 2PC

Clearly VR was heavily influenced by the earlier work on two-phase commit.

Our primary is like the coordinator, and the other replicas are like the participants.

Furthermore, the steps of the protocol are similar to those in 2PC and even have names (prepare, commit) that come from 2PC.

However, unlike in 2PC, there is no window of vulnerability in VR: there is never a time when a failure of the primary prevents the system from moving forward (provided there are no more than f simultaneous failures).

In fact, VR can be used to replicate the coordinator of two-phase commit in order to avoid this problem.

From **V**iewstamped **R**eplication to **P**ractical **B**izantine **F**ault **T**olerance

Liskov, B.(2010)

Replication - Lecture Notes in Computer Science, Vol. 5959, Chapter 7, pp. 121-149.

Original paper

Castro, M. and Liskov, B. (1999). **Practical byzantine fault tolerance**, *Proceedings of the Third Symposium on Operating Systems Design and Implementation* (OSDI).

https://youtu.be/Q0xYCN-rvUs

VR and PBFT

- VR allows a group of replicas to continue to provide service in spite of a certain number of crashes among them.
- PBFT handles Byzantine failures.
- Both protocols allow users to execute general operations (thus they provide state machine replication).

PBFT

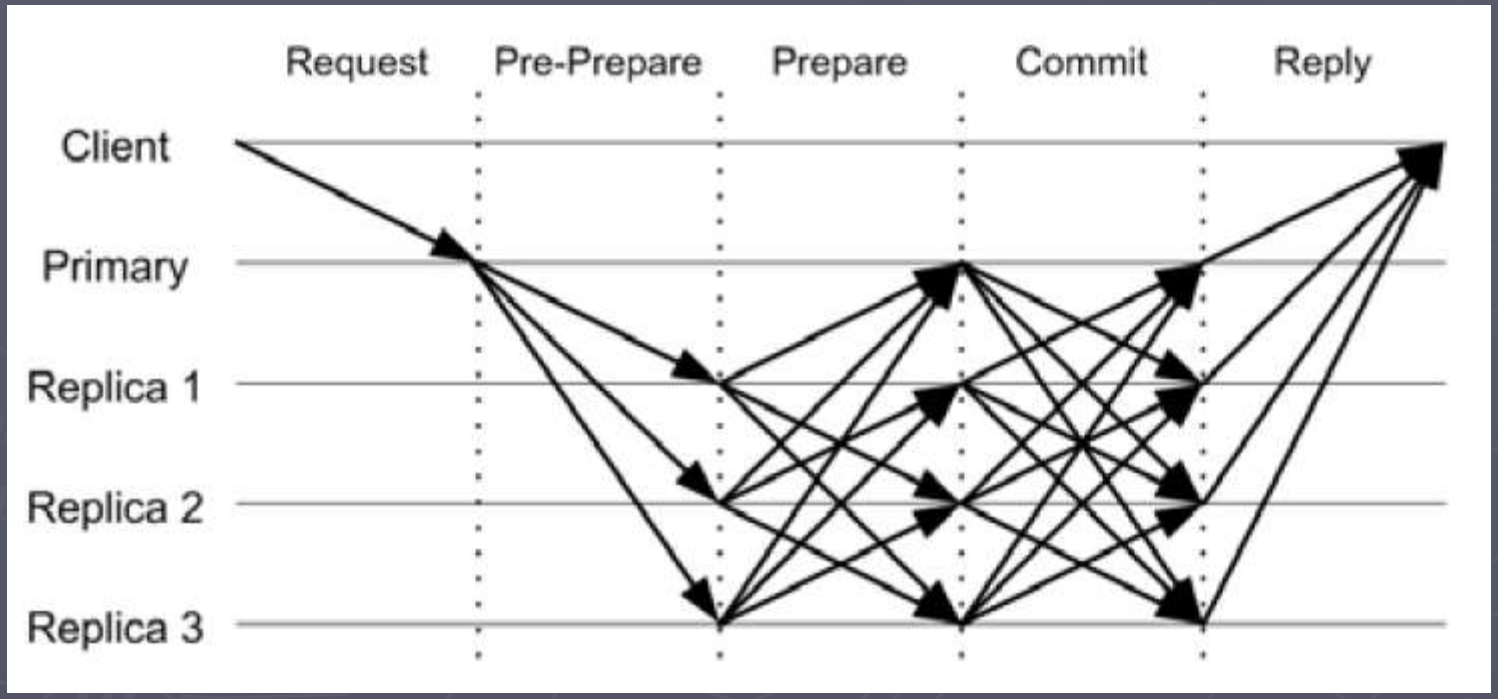
PBFT uses 3f+1 replicas to tolerate up to f faulty nodes (bizantine failure mode).

The system must be able to execute a request using responses from 2f+1 replicas. It can't require more than this many replies because the other f replicas might be faulty and not replying. However, the f replicas we do not hear from might merely be slow to reply, and therefore up to f of the replies might be from faulty nodes. These replicas might later deny processing the request, or otherwise act erroneously. We can mask such bad behavior, however, since we have replies from at least 2f+1 replicas, and therefore we can be sure that at least f+1 honest replicas know about the request. Since every request will execute with 2f+1 replicas, we can guarantee that at least one honest replica that knows of this request will also participate in the processing of the next request. Therefore, we have a basis for ensuring ordered execution of requests.

PBFT

- Quorums of at least 2f+1 replicas
- All messages are signed by the sender, since replicas might lie
- Extra phase in the protocol
  - Pre-Prepare (to cope with a malicious primary)

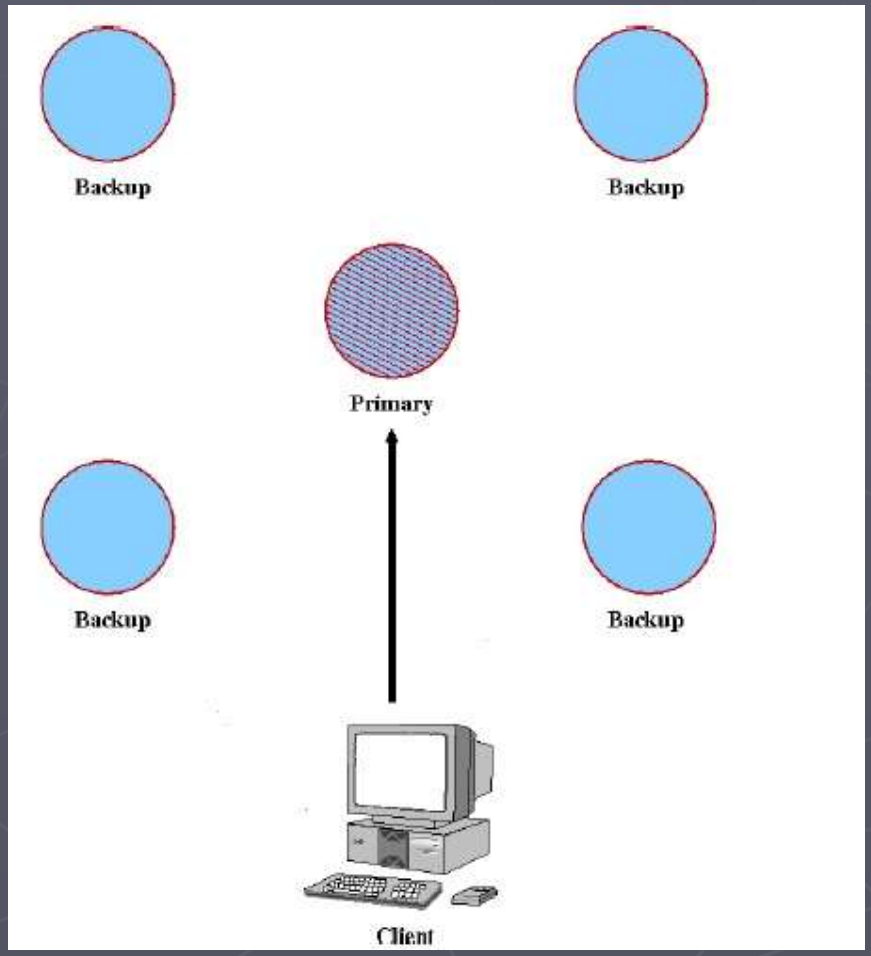
Normal operation



https://youtu.be/IafgKJN3nwU

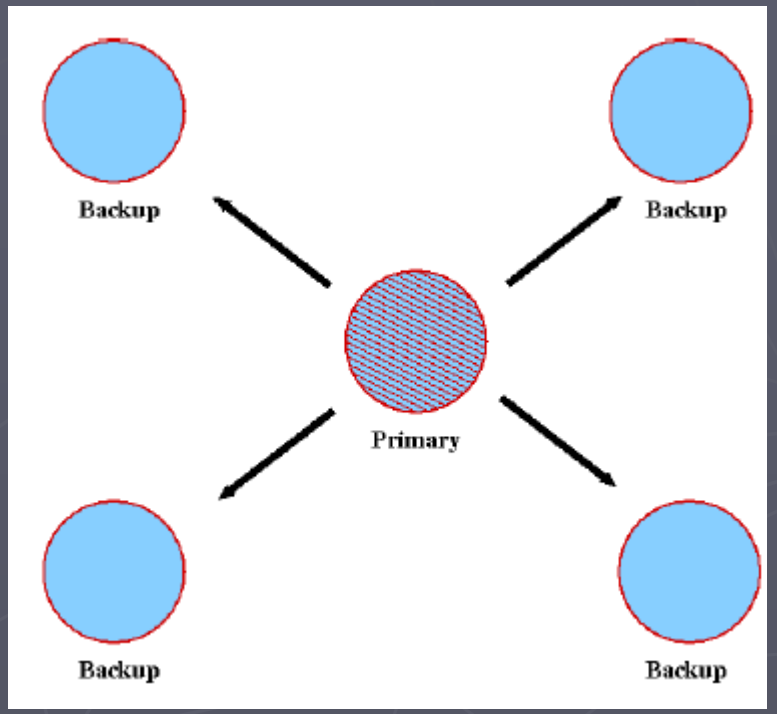
Client

Client sends a request to the primary



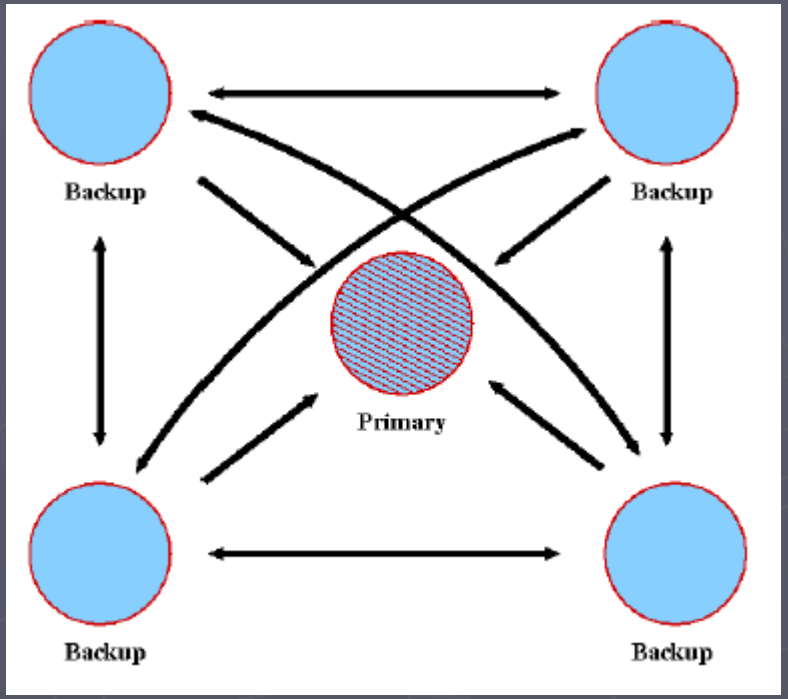
Pre-Prepare

Primary multicasts a PREPREPARE message containing current viewstamp



Prepare

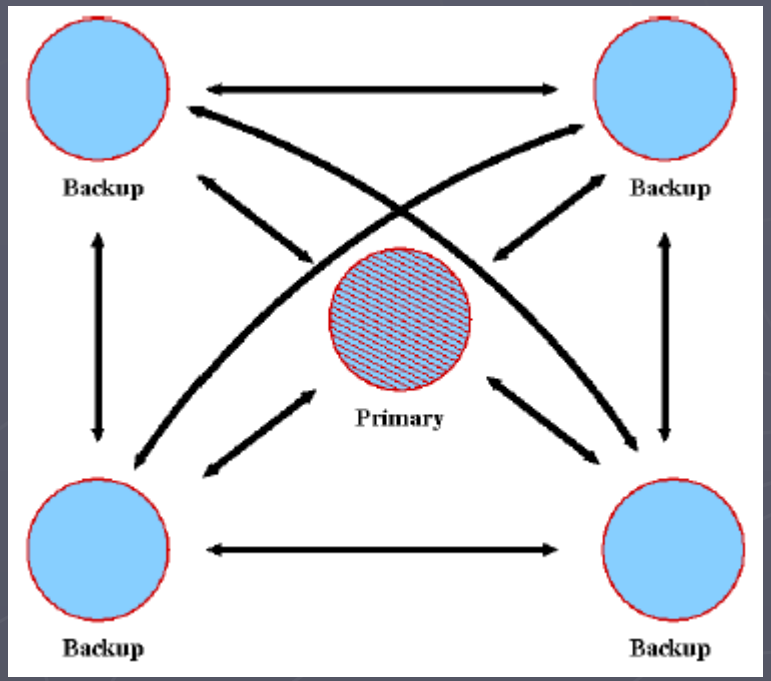
Backup “acknowledges” by multicasting a PREPARE message if it has not accepted a different PREPREPARE at that viewstamp



Commit

Request is *prepared* if replica has collected 2f+1 PREPREPARE and PREPARE messages

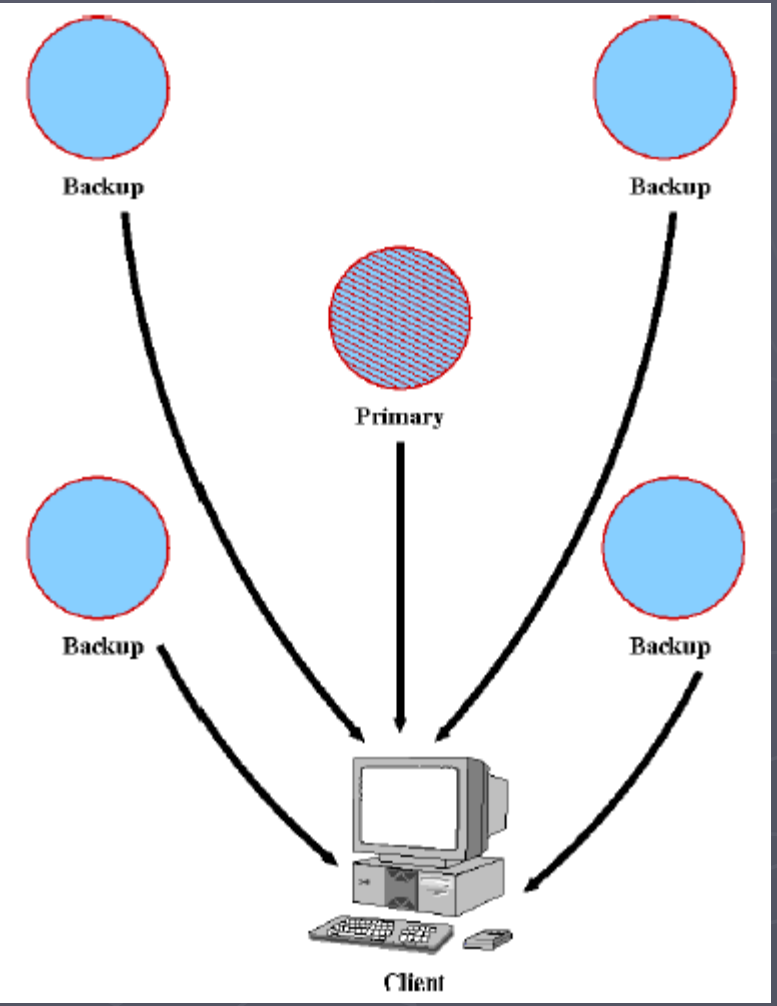
Backup multicast COMMIT message



Reply

Each replica execute the request when it receives 2f+1 COMMIT messages and send the result

Client waits for f+1 replies



PBFT

- VR uses a primary process to choose.
- PBFT uses a primary to propose, a quorum to choose.
  - A principle at work in a Byzantine setting: we can trust the group but not the individuals

View Change

Challenge

- convince backups of view change

Solution

- use (quorum) certificates to ensure that all committed operations make it into the next view in spite of whatever faulty replicas might attempt to do.
  - A certificate is a collection of matching valid signed messages from 2f+1 different replicas
  - A certificate represents a proof that a certain thing has happened