Zyzzyva: Speculative Byzantine Fault Tolerance

Océan Gillaux

DISTRIBUTED SYSTEM MID110

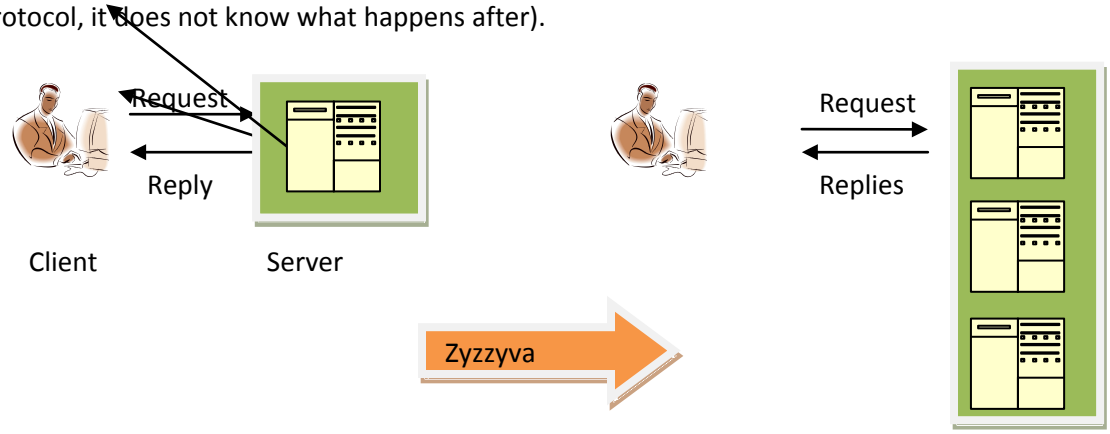
University of Stavanger April 2010

This paper is a summary about Zyzzyva protocol [1]. This protocol implements the design of Byzantine fault Tolerance (BFT). Before starting and explaining Zyzzyva, a short presentation of Byzantine [2,3].

Requirements

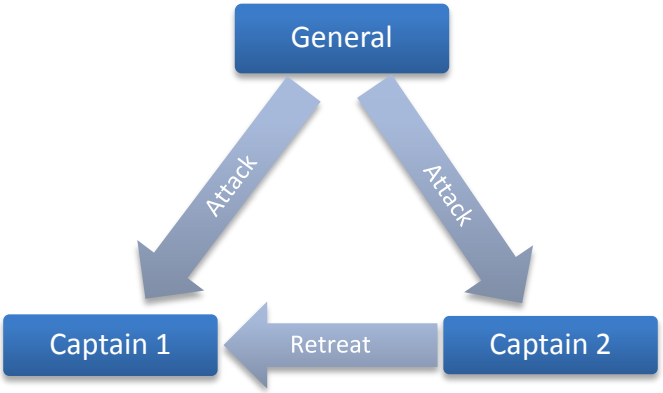
Why to use fault tolerance? Now especially with Internet, the users want access 24/7 for the data, the website, email, … But unfortunately no server is perfect. We must have system for managing if one or more server does not work or is faulty. The solution is to add more servers, but you need to check the integrity of the server response. For example if there are some errors in the data (database, file system corrupted) or worse the server is corrupted by adversary. We need system that manages the problem with server but also that checks if the response received by the client is correct.

Zyzzyva transforms services into high-performance and reliable services. The service is replicated to tolerate failure. Application has to see one centralized service (Application "sends" request to zyzzyva protocol, it does not know what happens after).



Byzantine fault tolerance or Byzantine General's problem

The name byzantine refers to the problem encountered by the generals of the Byzantine Empire Army [4]. The generals had to decide who attacked, but they could communicate only by messenger and eventually there were traitors among the generals.



In this example, the captain 2 is the liar. How the Captain 1 can choose the good response?

If you replace the captain 1 by the client and the general and the Captain 2 by the server, you have the Byzantine fault tolerance problem in computer science. For this example 2 servers with one liar have not solution, we need more server, but it's not the subject of this paper. To tolerate m traitorous generals it is required 2m + 1 loyal generals[4].

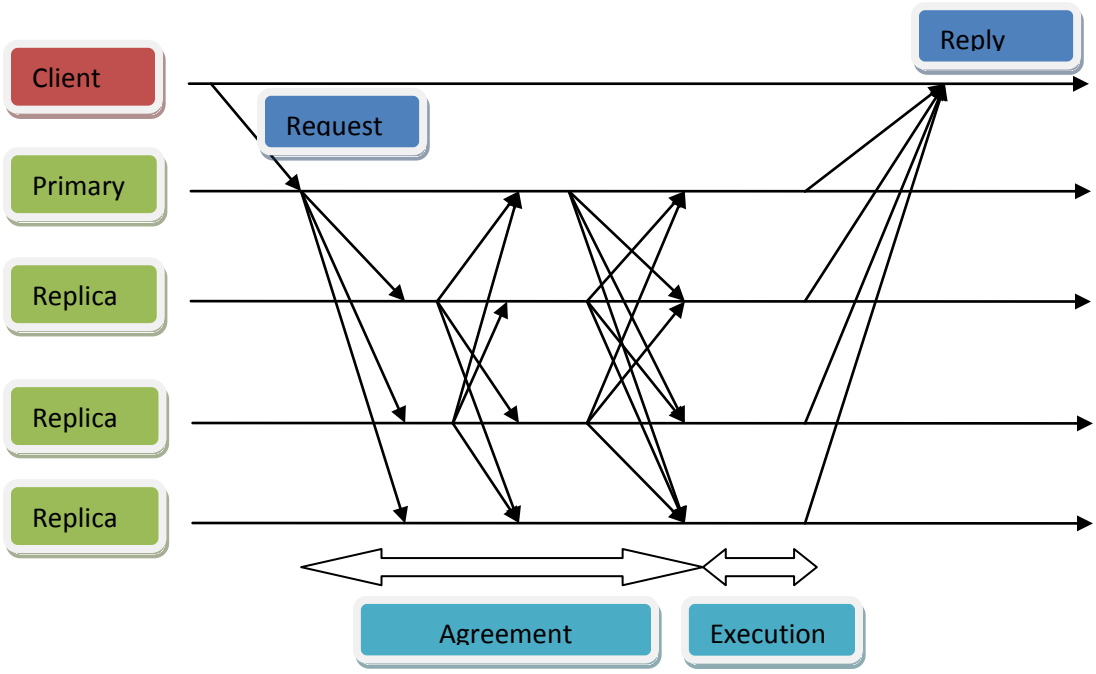
Security

We admit that the adversary cannot break cryptographic techniques like collision-resistant hashes, encryption, … Zyzzyva uses the concept of public/private key.

Introduction

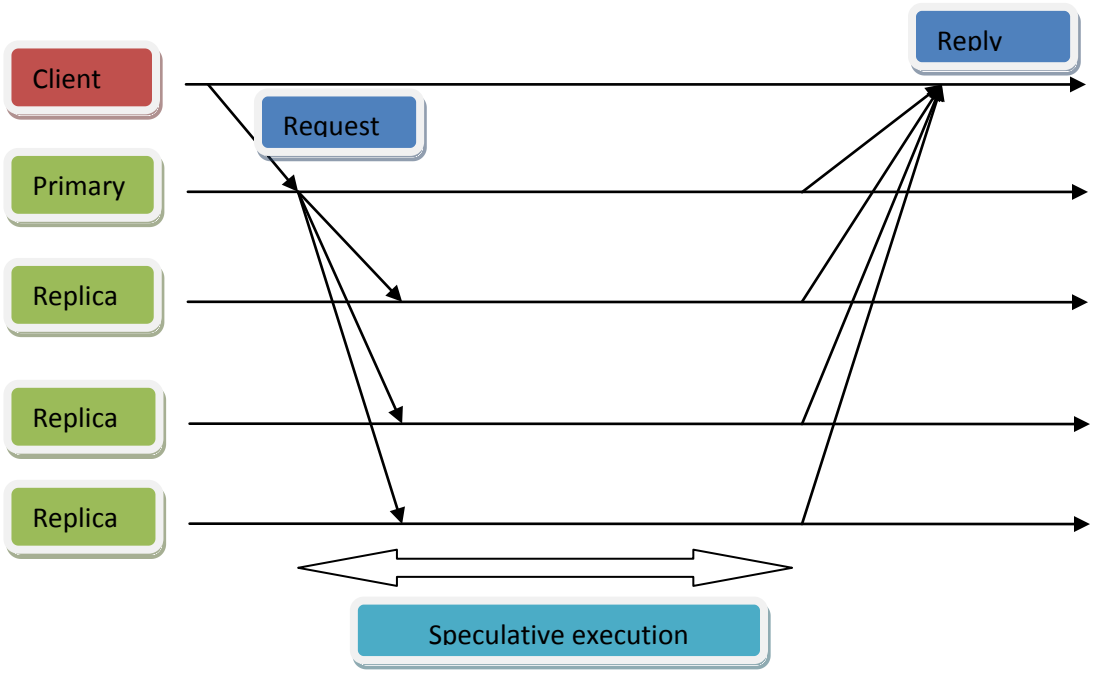
Different protocols implement this solution, the most famous is Practical Byzantine Fault Tolerance (PBFT)[3]. But these protocols need a long phase of agreement consensus. It is not easy to choose the correct implementation because it depends on your architecture and requirements: High request contention, low latency, Replication cost, … Zyzzyva tries to optimize this phase and simplify the structure of BFT. The main idea in Zyzzyva to reduce the consensus phase is speculation.

Traditional BFT state machine replication:



The replicas are to agree on the execution order before answering the request. This system has a cost: Many messages and time for agreement phase.

Zyzzyva state machine replication:



Replicas execute the request without agreement.

The replicas do not need to know if the system is consistent, only the client verifies if the reply is stable before committing to the application. If is not stable, the client commit message to ask the primary and replicas to converge on a good view.

In the replication services fault tolerance, “f” represents the number of tolerating fault.

Zyzzyva Protocol

The Zyzzyva protocol is based on three sub-protocols:

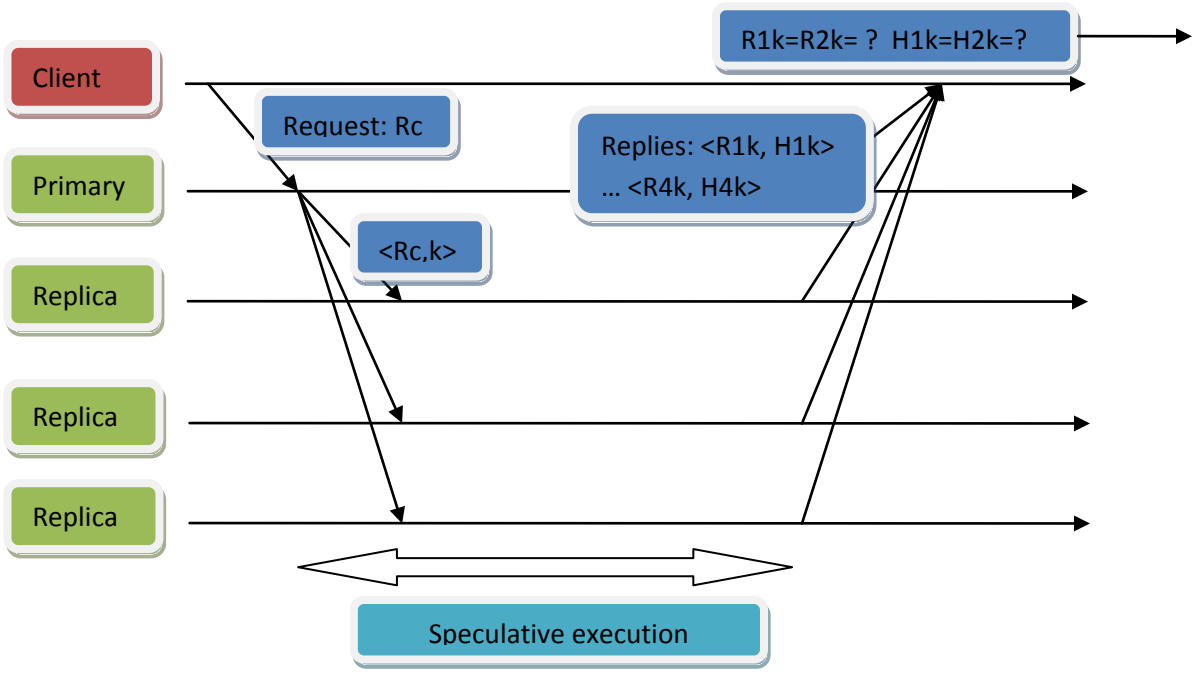
- Agreement protocol (Orders the execution)
- View-change protocol (Manages the change of the view)
- Checkpoint protocol (Manages the state stored by the replicas)

Agreement

How the clients check stable reply? The client uses the history included in the message.

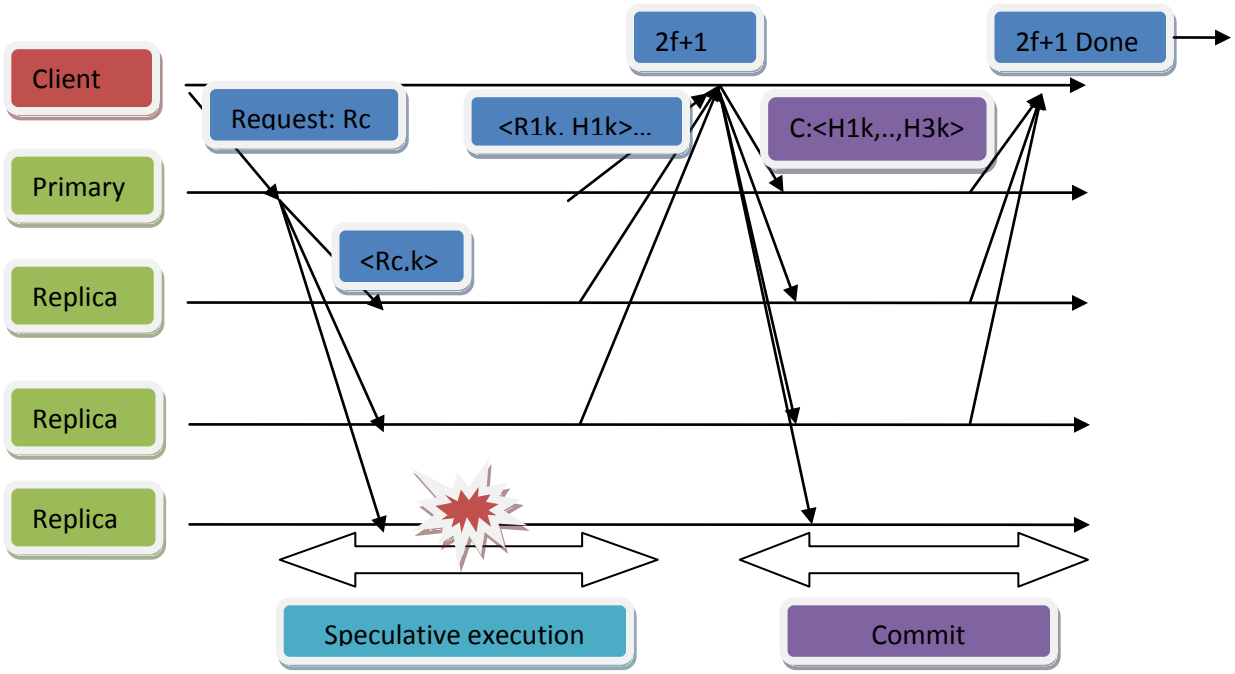
The request history contains the order of the request executed. All the replies contain application response and request history. History: $<R_{ik},H_{ik}>$ Reply from a replica i after executing request k. This part is the main idea of Zyzzyva protocol: speculation.

Execution with 3f+1:



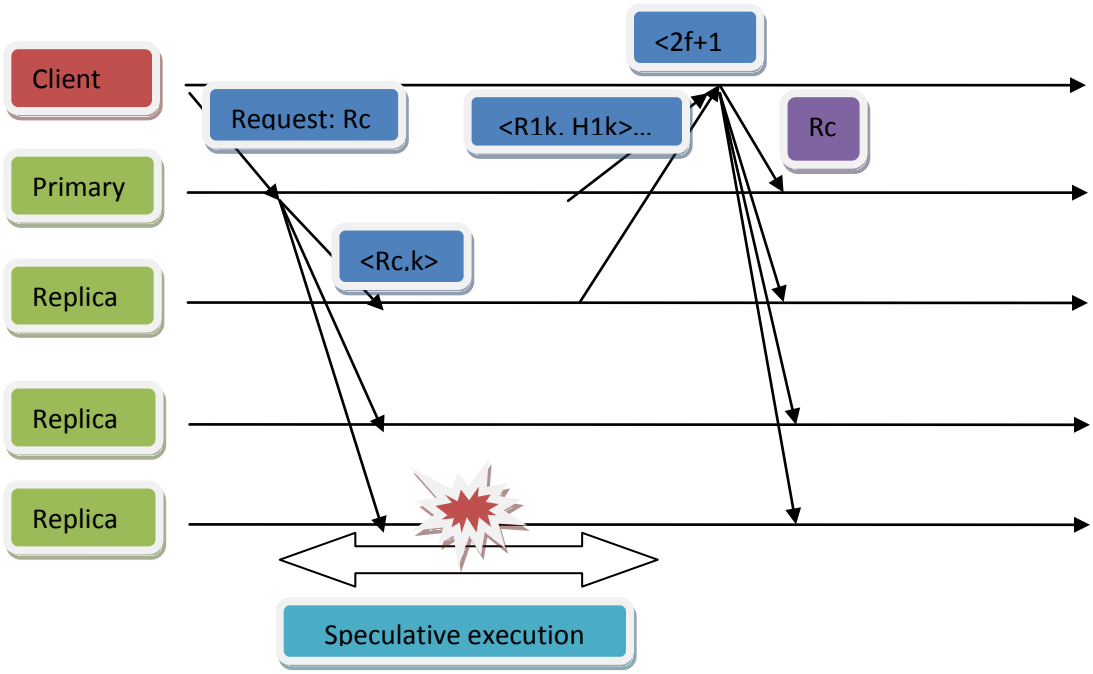
1. Client sends a request to the primary Rc
2. Primary receive request and forwards ordered request to replicas <Rc, k>
3. Replica receives ordered request, speculatively executes it and responds to the client <R1k, H1k> … <R4k, H4k>
4. Client receives matching responses, i.e. all the application response are all equal R1k=R2k=??, all the request histories are equal H1k=H2k, and the client received 3f+1 responses. If everything is ok, zyzzyva completes the request to the application else see next part.

One replica faulty: 2f+1 replies:



1. Client sends a request to the primary Rc
2. Primary receives request and forwards ordered request to replicas <Rc, k>
3. Replica receives ordered request, speculatively executes it and responds to the client <R1k, H1k> … <R3k, H3k>
4. Client receives matching responses, but the client receives only 2f+1 response. The client sends a commit message with different information, list of replica has responded.
5. The replica receives commit from client and send a Local-commit to client
6. The client completes the request.

The client receives less 2f+1 response:



1. Client sends a request to the primary Rc
2. Primary receives request and forwards ordered request to replicas <Rc, k>
3. Replica receives ordered request, speculatively executes it and responds to the client <R1k, H1k> … <R2k, H2k>
4. Client receives matching responses, but the client receives less than 2f+1 responses. The client sends a request Rc to all replicas (the client needs more than two replies before sending to application)
5. The replica receives Rc from client, if there is some inconsistency, the replica initiates a view change

Node State and Checkpoint protocol

Each replica stores histories of execution request and certificates, for maintaining consistent history and optimizing it, zyzzyva uses checkpoint protocol. Each CP_INTERVAL, replicas try to create checkpoint and send after creating it to all replica a CHECKPOINT message. Replica maintains only one checkpoint, we did not need to store everything, only the last information could be necessary.

View Change

The view change manages the new primary election AND guarantees the history has not changed for the correct request for the correct client.

The replica i having suspicions about the primary, does not stop working in the view but ask the other replica to vote “no confidence” in the primary, if f+1 replicas say yes, the replica i sends VIEW-CHANGE message to all replicas. The correct replica that receiving this message, joins the mutiny. With the

history and the commit from client, the new primary can know the correct history with checkpoint protocol and sends to the other replica the right history.

Client

We can see that the client role is important in Zyzzyva. There is an interesting question: Can a faulty client block zyzzyva?

The faulty client did not depose the commit certificate. This cannot block the other clients and they send commit certificate. Correct client ensure system progress because the system uses cumulative history of request and the correct client commit all previous requests. Faulty client cannot block the others. It can only affect its own process.

The faulty client cannot block the others, but could it compromise the safety by committing inconsistent history? The protocol uses encryption with private/public key, the faulty client cannot forge a bad history. Two valid certificates cannot have different prefixes, the encryption protect this.

Optimization

Replacing signatures with MACs

Many implementations of BFT use cryptographic operations to protect message, the main optimization used to reduce this is to replace signature by MACs.

Separating agreement from execution

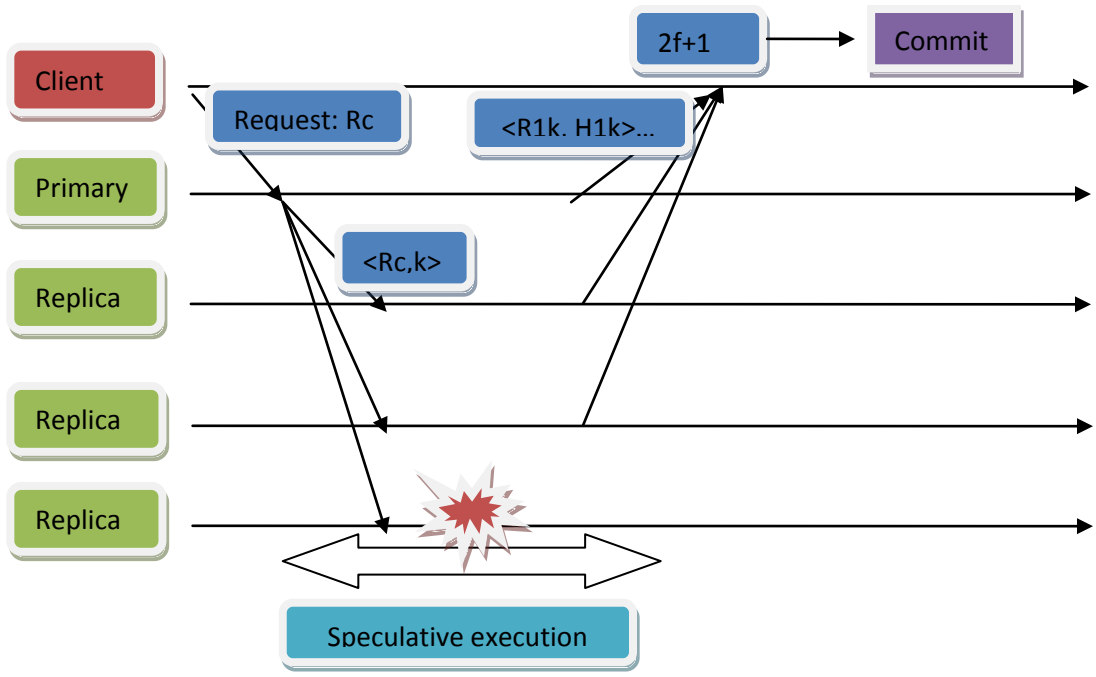
For optimizing application, state is replicated only 2f+1 replica, the other replicas are used like witnesses.

Request Batching

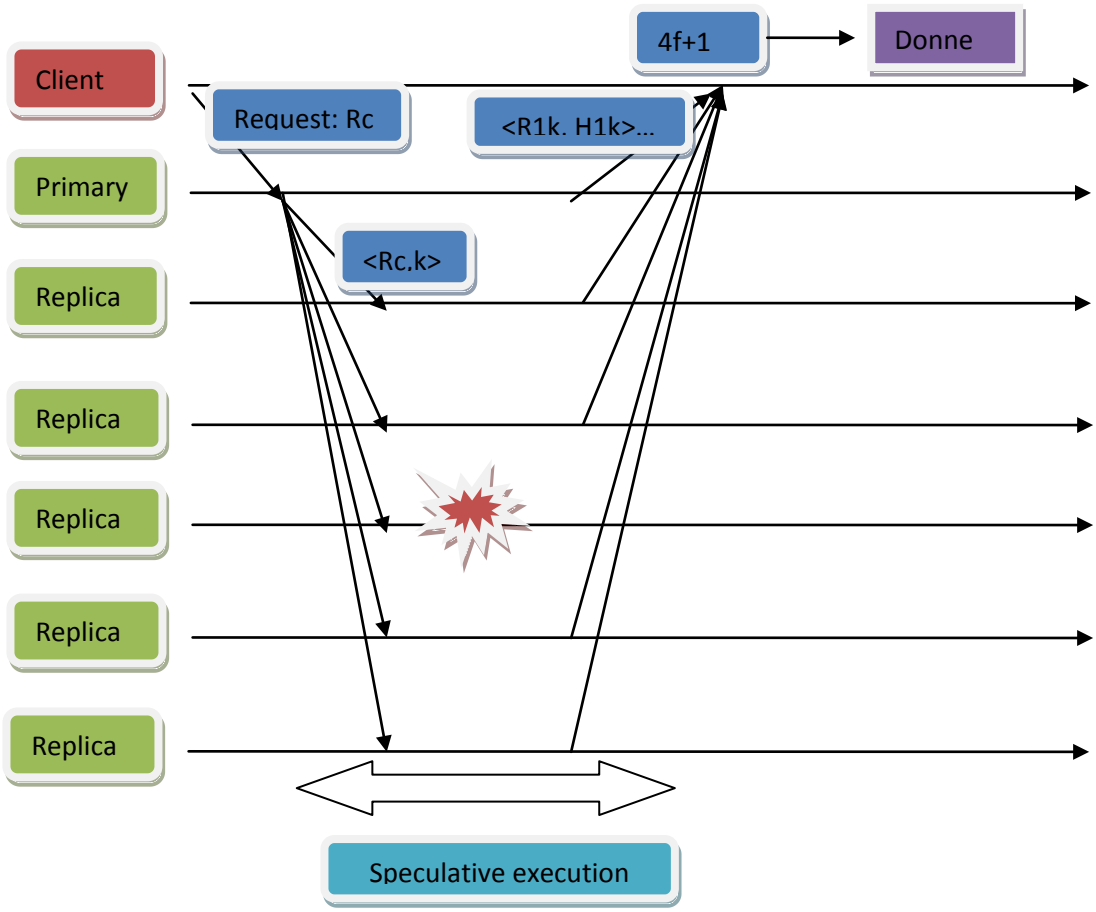
This technique is not specific to zyzzyva, a lot of systems use batching to optimize. For example if you see 10 times google.com in the day, your navigator asks once time pictures from google.com and after uses cache.

Zyzzyva5

We saw, if we use 3f+1 replica and the client receives 2f+1 responses (one server faulty), the client needs to send a commit before completing the request to application. Zyzzyva5 does not use 3f+1 replica but 5f+1 replicas. In this system if one server does not work, the client receive 4f+1 responses, there is no problem, and it can directly complete the request to the application.

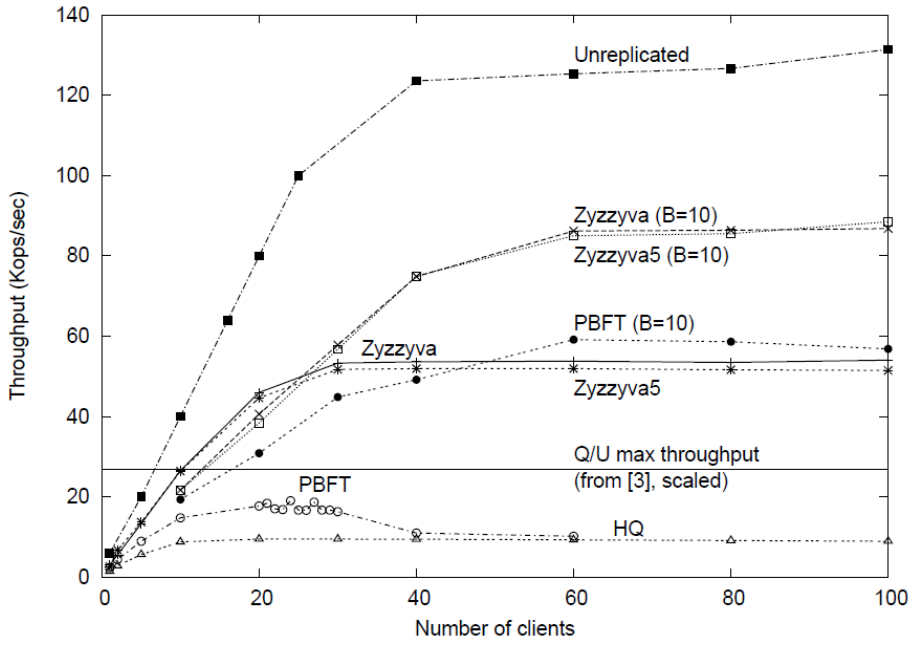


Zyzzyva5 with 5f+1 completes in a single phase with f faulty replicas.



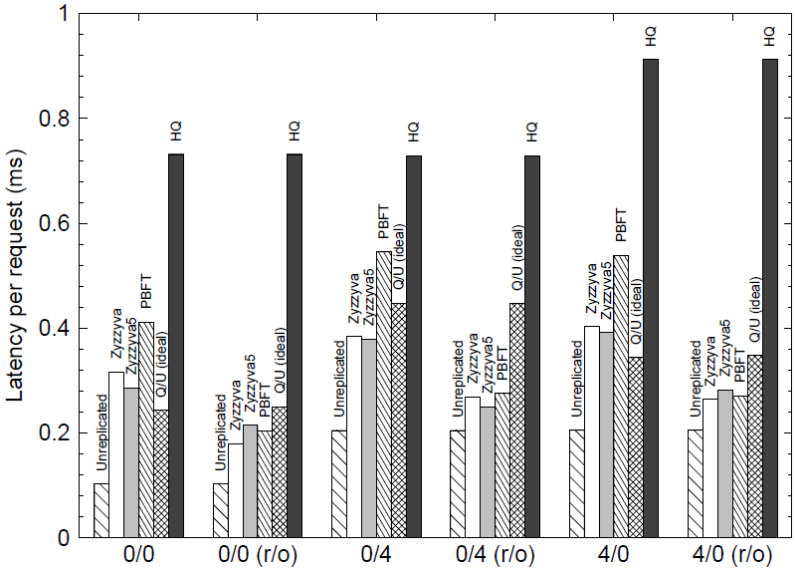
Evaluation

This evaluation uses benchmark (client sends 4KB request and receives null response, the client sends null request and receives 4KB response). We can know the number of operation over the number of clients.



B represents the size of the Batch. We can see Zyzzyva is the most performing BFT.

But it is not only the unique evaluation; we would like to know the latency between the request and the response.



0/0, 0/0(r/o) are different configuration of request and response.

Conclusion

In exploiting speculation, Zyzzyva has a good performance over existing BFT services. Zyzzyva approaches the theoretical lower bounds for any BFT.

References

[1] Zyzzyva: Speculative Byzantine Fault Tolerance, Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong Dept. of Computer Sciences University of Texas at Austin, 2007

[2] Byzantine Fault Tolerance Wikipedia, http://en.wikipedia.org/wiki/Byzantine_fault_tolerance

[3] M. Castro and B. Liskov. Practical byzantine fault tolerance. In Proc. OSDI, February 1999.

[4] In French, Algorithme des Généraux Byzantins Yann CEZARD, DESS TNI - Université de Montpellier II, décembre 2001 http://www.lirmm.fr/~ajm/Cours/01-02/DESS_TNI/TER21/