Proceeding of the IEEE
International Conference on Robotics and Biomimetics
Dali, China, December 2019

A pipelined Single-Phase Paxos Extension without lease

1st Tongxing Ma
Interactive Entertainment
Tencent
Shenzhen, China
terrillma@tencent.com

2nd Haibin Wang
Jining Institute of Advanced Technology
Shandong Institute of Advanced Technology
Jinan, China
hb.wang@sdiat.ac.cn

***Abstract*—Variety of single-phase consensus protocols based on Paxos have made significant progress in low latency and high performance. However, these improvements are either limited to specific favorable conditions, such as most commands rarely dependencies on each other, or weaken system availability with bigger fast writes quorums requirement or designated leader. In this paper, Decree Promise Paxos (DP Paxos) is proposed and implemented. DP Paxos provides single-phase fast writes without commands dependency restriction by leveraging a no lease and dynamic Decree Promised leader. The single-phase fast writes only require quorums composed of a majority of legislator nodes, these quorums are the minimum requirement to solve consensus. In most leader and lease system, followers redirect all requests to the leader, leader failure causes the system stuck till the lease expired and new leader is elected. DP Paxos solved the stuck problem by allowing followers proposal commands via two-phase Paxos instances when it receives client requests. Followers and leader may compete to proposal different commands. Normally, too many collisions of competing proposals will result in expensive coordinate cost and poor performance. In our scenario, 3-node cluster provides remarkable available improvement and acceptable cost for large-scale deployment. In DP Paxos, we design a 'reserve' and 'odd-even gear' mechanism to reduce collisions, it is very efficient for coordinating two nodes competing proposals in 3-nodes cluster. Generally, each proposer node propose 50 percent of client requests will cause massive collision, but after the DP Paxos optimization, these collisions are decreased to unbelievable rarely. Moreover, a busiest follower proposer substitutes the previous leader in very simple and cheap way, just attaching few bytes of election info within a normal command. The busiest node play the leader role,ensure more than half of client requests are reach consensus in single-phase. Decree Promise Paxos is not only a theoretical algorithm, but also implemented and applied in some online games data cache service of Tencent, where it processes 3.6 billion writes per day.**

***Index Terms*—Paxos; Consistency; Fault tolerant; Availability; Reliability**

I. Introduction

Paxos is a well proven and widely used algorithms for solving the consensus problem. It provides strongly consistent and extreme fault tolerance with reasonable expense [1], [2], [12], [13]. A standard Paxos instance composed by phase 1 prepare-promise and phase 2 ballot-vote, to conduct a success ballot will at least cost 2*RTT (Round trip time) latency. Many research on Paxos based protocol contributes to low latency and fewer data to exchange. For example, the leader and lease mechanism [7], [15] define a designated node as leader which is responsible for ordering received commands, while all other nodes redirect client requests to the leader. In no-faulty executions, a client command reaches consensus within three communication delays. However, if the leader goes down, the whole system will be unavailable until the lease expires, and a new leader is elected. In potential, the leader and lease mechanism weaken the system availability.

In practical distributed system, it is not trivial to detect the leader failure immediately, the system unavailable gap is a non-negligible problem. Solve the stuck (leader failure leads to system unavailable) problem has great significance of high-reliable decentralized distributed system.

Several recent algorithms solve the designated leader defect by introduce collision recovery mechanism to allow multiple nodes to proposal commands concurrently. In case of commands rarely interfering with each other, commands are decided in two communication delays, these systems provide higher availability and good performance. But, these algorithms introduce high cost of handling contention among concurrently interfered commands from multiple proposers. Egalitarian Paxos, a recent multi-leader consensus algorithm, employs four communication delays for safely decide a proposed command in case of contention. In order to reduce collision, these algorithms choose to relax the consistency requirement by allowing non-interfering commands to apply disordered. The approach is called generalized consensus, a weaken version of traditional consensus. In some application scenarios, most of concurrent commands interfere with each other, for example online flash sales, stock trading, open world online games (massive players interact with each other) and so on. Commands operate some related data, there are dependency relations among commands, to achieve consistent result, all commands must apply in strictly same order on all replicas. Any commands applied disorder may result in inconsistent outcome, generalized consensus is no longer applicable, these generalized consensus algorithms are fallback to slow and expensive collision recovery procedures.

In this paper, Decree Promise Paxos (abbreviated as DP Paxos) was proposed. It is aimed at optimizing the consensus

Corresponding author: Haibin Wang. The research was supported by National key Research and Development Plan(2017YFB1302400), key Research and Development Plan in Shandong Province(2017CXGC0912)

978-1-7281-6321-5/19//$31.00 ©2019 IEEE

algorithm performance without weaken availability and relax consistency requirement. Based on our application scenarios (commercial online games), 3-node cluster provides remarkable available improvement and acceptable cost for large-scale deployment. So, DP Paxos focus on provide good performance in 3-nodes cluster. Briefly summarize the technique specification of the consensus problem solution: (1) Legislator nodes commit high proportion of concurrently commands in two communication delays (*pipelined single-phase fast write*); (2) The majority legislator nodes available, then the system available, leader failure not causes system stuck or performance degradation; (3) Applicable for scenarios that most commands interfering each other (traditional consensus), not only limited to favorable conditions of low inter-node contention. *Pipelined fast write* in this paper means the algorithm allows concurrent ballot-voting on different commands, commands commit out of order but applied in order. With this *pipelined* feature, the system keeps good throughput even when working with the long distance of WAN latencies.

DP Paxos exploit leader mechanism to implement the pipelined single-phase fast write. The leader has no lease, to changing the no lease leader, just execute a normal command choose procedure, no additional leader election mechanisms required. It solves the leader failure led to system unavailable defect in an unprecedented way. The new approach not trying to detect the previous leader failure or network partition but allow any nodes to proposal decrees (In the paper, decree refer to command.) with two-phase Paxos instances when follower nodes receive client requests, further, accompany with few decrees are chosen, the most prolific proposer node change to new leader and acquire single-phase authorization by utilize the Decree Promise mechanism, then easily change to single-phase fast writes. DP Paxos fast writes need only classic quorum of nodes to ballot, it is composed of $[\frac{N}{2}]+1$ nodes, the minimum requirement of consensus algorithm, where *N* is the total number of nodes. Any follower nodes, who believes it is the most prolific proposer, can easily change to be the newer leader. In the local view of follower node, multiple nodes may though they are the most prolific proposer, for sure, multiple nodes may execute single-phase pipelined fast writes, normally, conflicted fast writes with classic quorum ($[\frac{N}{2}]+1$) may cause inconsistency or stuck, whereas DP Paxos circumvent these issues by Decree Promise mechanism. It not relies on larger fast quorum to recovery from conflicted fast writes, but, try to guarantee the earlier (earlier means the leaders are elected with smaller decrees in the decrees sequence) leaders fast writes will be ignored by majority nodes, in other words, will not be chosen. That is the key contribution of the algorithm.

In Section 2, recall the Phase 1 prepare-promise in Paxos, it is the intuition of DP Paxos fast writes. Related works and similar solution are discussed too. Section 3 describe the design of DP Paxos in detail, leader election, fast writes and so on. Also, Simple mathematical proof is given in Section 3.4.

Section 4 focus on engineering implementation. In this part, all key designs are about solving conflict. In our scenario, a 3-nodes cluster is the most attractive choice, several optimization mechanisms are implemented for 3-nodes cluster. In practice, proposal conflict retry may cause re-conflict. Traffic control implementation prompt fast writes decrees are prioritized chosen and lower efficiency normal writes halt for timeout retry if there are some conflict. An 'Odd-even gear' mechanism reduce the conflict to a rarely low rate, moreover 'Reserve' mechanism coordinate the rarely few conflict without retry Paxos Prepare-Promise phase. Test and online data will be shown in Section 5.

To enable this paper to be most clearly understood, some key words should be introduced first.

Decree value: Refers to the information that needs to be synchronous between server nodes. In a distributed system, it could be values, commands or a raw buffer.

Decree ID: It is a continuously increasing number for each decree. Each Decree ID is assign to a Paxos instance. A Paxos instance choose a unique decree value. If there are more than one decree values are proposal by different legislator nodes in same Paxos instance (means same decree id), the Paxos algorithm guarantee all legislator nodes will reach consensus about which value is chosen. In a parallel working system, many commands could be processed by Paxos instances concurrently. If nodes missed some chosen decree value, they can identify the hole in the decree ID sequence and try to learn from other replicas.

Legislator: Legislator (In different paper, aka priest, acceptor) is the server node that can vote for the decree.

Proposer: If one node plays the role of proposer, when it receives a client request, it will start a Paxos instance and propose it.

Ballot number: In a Paxos instance, there may have more than one ballot round, and each ballot round is assigned a unique Ballot number. In DP Paxos, there is a designated smallest ballot number, it is 0.

Earlier Round: The earlier round means the ballot round with smaller ballot number. It describe the logical order of ballot round, not timing order.

Fast round: In DP Paxos, the fast round use 0 as its ballot number. Naturally, fast round is the earliest round in all Paxos instances. In fast round, proposal could start Begin Ballot directly, skip Phase 1 (Prepare and Promise).

Fast Write: Some legislator nodes in the algorithm defined status, have authority to safely proposal commands via fast round in Paxos instances, skipping Phase 1.a and 1.b [2]. These proposals are called fast writes. Fast means less latency and fewer steps of communication.

Network partition: Communication failures fragment the network into isolated sub networks called Network partitions [14].

II. Comparison with Related Work

Paxos is a two-phase protocol, a classic single instance is composed by Phase 1 (prepare-promise) and Phase 2 (ballot-vote). By introduce leader and lease mechanism, A replica

becomes a leader by running Phase 1 once for all unused Paxos instances. The leader is the distinguished proposer, all commands will be send or redirect to the leader. Leader assigns a new command id for each command, then leader try to proposal the command. In steady status, no command conflict and all command could be commit in two or three messages delay. But, in a distributed system, it is hard to identify if the leader is available (alive in a network partition with majority nodes.). When the leader is unavailable, the system is stuck for some seconds to collect information and elect a new leader. In fact, these designs sacrifice the system availability to save communication cost. Spanner is one of the most large-scale, latest and successful Paxos based data storage system. In spanner, killing leader with no warning, the rate of completion drops almost to 0, the Paxos leader leases are set to 10 seconds, approximately 10 seconds after the kill time, the groups have new leader and throughput has recovered, spanner team tried to implement some mechanism to reduce the recovery time [5]. In Decree promise Paxos, killing leader with no warning, the system keeps stable throughput and takes no time to recovery. There are also have batch prepare, pre-prepare engineering method [16] to lower the command chosen cost, as two-phase implementation, command chosen latency is improvable.

In previous Paxos based projects, engineers designed some approaches to do fast write (without prepare) with reserved ballot number, like zero. These designs try to follow the basic rule that each round has a unique ballot number in a Paxos instance (each decree). So obviously, limitation occurs only after a previous value has been chosen. The next decree could run with fast writes to ensure that there is only one ballot using reserved number with same decree id. Fast write can not run concurrently and other parallel requests should run full Paxos procedure, just like fast write design in MegaStore [9], *fastpropose* design in Deconstructing Paxos [20]. DP fast round breaks the rule. Even when there are more than one decree values conflict with the same ballot number zero, DP Paxos could run fast writes rounds safely, safely means keep consistency.

DP fast write is different from Fast Paxos fast round also. It works well without having to redefine (enlarge) the fast write ballot round quorum set requirement. That means, when the majority of the legislator nodes are alive (failure nodes less than $\frac{N}{2}$), DP fast wirte can work. It has better fault-tolerance than other algorithms [3], [10]. The DP fast write quorum is always the majority, different from the fast quorum defined in Generalized consensus and Paxos [6]. Unlike the Cheap Paxos [4], the system requires no reconfiguration if majority of nodes are alive when the leader node fails.

As an extension of Paxos, review the basic Paxos procedure will helps to understand DP Paxos. A basic Paxos procedure includes the following 4 steps, consisted by Phase 1.a,1.b and Phase 2.a, 2.b:

(1). Proposer node gets a client request to conduct some operations. To make sure all replicas do the same job, the node will start a Paxos instance. Proposer chooses a new decree id *d* and new ballot number *b*, which should be different from any other ballot numbers in this decree instance. Proposer start the round by broadcasts the Prepare request with ballot number *b*. We describe the Prepare request as **Prepare**(*d, b*). This is Phase 1.a of Paxos.

(2). Once a node receives a Prepare request, it will check the local status of decree *d*.

Case I. local max ballot number $b_n < b$ and no value is voted in previous ballot. It will return a promise message, described as **Promise**(*d, b, lastvote*(*-1, null*)). Local b_n will be updated to *b* as well.

Case II. Local max ballot number $b_n < b$ and it votes decree value V_m in the previous ballot b_m ($b_m \leq b_n$). It responds with a promised message like this:

Promise(*d, b, lastvote*(b_m, V_m)).

b_m is the greatest local ballot number that is less than or equal to b_n and voted some value.

For example:

Node N local info about decree *d*:

Promised Ballot number *3*

Voted for Decree value V_8 with ballot number *8*

Promised Ballot number *12*

If a new prepare request **Prepare**(*d, 13*) arrives, Node N will send **Promise**(*d, 13, lastvote(8,*V_8*)*) and update promised ballot number to *13* locally.

Case III. Local Max Ballot number $b_n >= b$. It will not do anything. It means Node N has already promised for b_n.

Step 2 is the implementation of Paxos Phase 1.b. Before a proposer start Phase 1.a, other proposer may already propose some different decree value and maybe has been chosen in some earlier ballot round (earlier means ballots with smaller ballot number). Propose unsafe value will break consensus. So, proposer should run Prepare-Promise (Phase 1) to collect these earlier ballots, and pick up the safe value (if there is one). Phase 1 makes sure after collection that these earlier ballots will not change again. Because of the quorum nodes after promised *b*, they will never accept or promise any ballot number smaller than *b*. Thats the meaning of **Case III**. A successful prepare with ballot number *b* will get a quorum nodes promise. Obviously, any two quorums must have intersection, so that any later arrived Ballot or Prepare with ballot number smaller than *b* could not succeed. Besides, the collected *lastvotes* must contain the value that has been chosen in the earlier ballot round (if there is one).

(3). The node gets a successful prepare with ballot number *b*. To start Phase 2.a, it could send Begin Ballot message to try to pass a decree, like: **Ballot**(*d, vote(b, V)*). *V* is selected by the rules below:

If majority nodes are in **Case I**, *V* could be any value.

Otherwise, *V* should be the Max *lastvote*(b_m,V_m) in the promised quorum. Max means the largest b_m.

(4). If a node receives a Ballot message and *b* is larger than or equal to the largest local ballot number, the Ballot could be accepted and the node will respond with a *Voted(d, b)* message. If the proposer can't collect majority votes, it start next round via increase the ballot number and retry Phase 1.a. Step 4 is Phase 2.b.

Similarly, once proposer collects a quorum voted for ballot number *b*, it means that the value is chosen. Any node trying to propose a different value, within the same Paxos instance, it will find the selected *V* in step (3) will always be the chosen value. If the proposer finds that *V* has been chosen, it can send a **Commit**(*d, b*) to acknowledge other nodes.

As we described, phase 1 find a safe decree value from earlier ballots. If the system define an "earliest" ballot with the designated smallest ballot number, then there have not any smaller ballot number and earlier ballots. The round with the smallest ballot number is called fast round. Obviously, in a Paxos instance if only one legislator node utilize fast round to skip phase 1, it is compatible with ordinary Paxos algorithm. Usually, if more than one legislator nodes begin ballot with fast round, the Paxos algorithm can not identify which fast round decree value is the logically *lastvote*, the Paxos instance can not guarantee consensus anymore. We design a new approach to allow a legislator node utilize fast round safely, while some unreachable nodes are execute fast round in same Paxos instance.

III. Decree Promise Fast Writes

Multiple nodes fast writes will result in different decree values be voted in different nodes fast round. If in later normal round with greater ballot number, the proposer collect more than one *lastvote* with different decree values but same smallest ballot number 0, it can not pick up the which decree value is safe. Our intuition is simple, if a proposer node *L* intend to utilize the fast round, it try to invalidate other nodes fast round authority first.It request a majority legislator nodes promise to not vote for the other node's fast round for all later Paxos instances with larger decree id, then, other node's fast round will not acquire enough vote to success for all later Paxos instances. After successful invalidate other nodes fast round authority, in later Paxos instances node *L* can utilize the fast round safely. In normal ballot round, we need a 'tag' to identify which fast round decree value is valid and which one is invalidated, then the algorithm can ignore these invalid *lastvote* and pick the valid one. Naturally, we use leader election mechanism to build the 'tag'.

A. DP Paxos Leader

At the very beginning, the system initial without leader. A legislator node process client request via ordinary Paxos instance. Unbounded and distinct normal ballot number set assign to each legislator node. The fast round ballot number 0 is not authorized to any one. For example:

Max Node id is 3
Node 1 ballot number set: {1, 4, 7, ..., 1+3*(t)}
Node 2 ballot number set: {2, 5, 8, ..., 2+3*(t)}
Node 3 ballot number set: {3, 6, 9, ..., 3+3*(t)}
$\{t \in (0, \infty]\}$

t is the number of times the Proposer retry to prepare in a Paxos instance.

As we know, in ordinary Paxos, each Paxos instance is assigned a unique decree id, and chosen a unique decree value. After few decrees commit, some legislator node will find it commit more commands than others. For example, in the last 128 committed decrees, 60 percent are proposed by Node 1. Node 1 is the busiest legislator node in the cluster. The best practice of reducing system cost is let the busiest legislator node acquire the fast writes authority. The approach of election is similar as Megastore [9], The leader is the busiest node chosen alongside the ordinary Paxos instance consensus decree value. In other words, the candidate node append its election command into a normal decree value, once the decree value is chosen in a normal Paxos instance, means the candidate win the election. The different from Megastore's leader is, DP Paxos leader serves for all later decrees till some newer leader elected. Each Paxos instance assigned a unique decree id, so as the one alongside election, the decree id of the leader election Paxos instance is the 'tag', we call it *ld* for short. So, every leader has a unique *ld*, the logically earlier leader have smaller *ld*. Earlier here similar as the earlier in earlier ballot number, its a logical order. Using the *ld* as a 'tag' for fast round, we can identify the leader of every fast round ballot. The ballot number 0 like normal ballot number, it is unique now, it is not a single number anymore, but a unique two-dimension array *[0, ld]*. For example:

Ballot(*2341, vote([0,128], Vn)*)
Ballot(*2341, vote([0,1672], Vq)*)

In a Paxos instance, its decree id is 2341, the earlier leader with $ld = 128$ proposal a value V_n and the newer leader with $ld = 1672$ proposal a value V_q. In the example, some legislators may voted V_n in fast round from earlier leader $ld = 128$, with the classic quorum available requirement, in the view of a single legislator node, it can not identify V_n is chosen or not, so vote for the newer leader's fast round ballot is not safe. In concurrent system, a leader *ld* can not start fast writes from *ld+1*, because earlier leaders maybe concurrently run many Paxos instance for fast writes, and some of them have larger decree id (ex. $2341 > 1672 + 1$), already been chosen. New leader fast writes from *ld+1* is not safe.

B. Decree Prepare-Promise

Modern system needs high throughput, and require many Paxos instances to run concurrently. In the previous example, Node 1 ($ld = 128$) may have broadcast fast write voting for Decree *p*, $p > 1672$, before it realizes Node 2 became leader in Decree 1672. At the same time, more than one node may believe they are leaders. Fast write is single-phase ballot, it is impossible to collect these existed earlier (*vote([0,128], Vn)* is logical earlier than *vote([0,1672], Vq)*) ballots. Decree Prepare-Promise is designed to address this problem. Our basic idea is to guarantee majority nodes do not vote for these earlier fast round ballots.

After a leader wins the election, it requests (**DPrepare**) legislator nodes to promise that they will not vote for all earlier leader's fast round ballot in new Paxos instances any more. Legislator node will promise (**DPromise**) the newer leader (leader with larger *ld*) and respond the max voted decree id (abbreviated as ***dm***) for the earlier leaders (*ldn*) fast

round ballot. Once the new leader collects a majoritys decree promise, in the decrees after the max ***dm***, these earlier ballots from earlier leaders fast writes will not be voted by majority, in other words, will not be chosen. Stated as follows:

DPrepare(*ld*)

DPromise(*ld, lastfastvote(ldn, dm)*)

For example:

N1 wins the leader election in Decree *ldn*

N2 wins the leader election in Decree *ldp*

$ldn < ldp < lds$ *and* $[0, ldn] < [0, ldp]$

let *ldq* = max(***dm***).

Majority nodes not and will never vote for fast round ballots in Paxos instances with decree id larger than *ldq*. New leader *ldp* can safely starts its fast writes from decree id *ldq+1*. We mark the leader's smallest safe fast writes decree id as *lds* for short. In the example, *lds=ldq+1*.

C. Simple mathematical proof

Formally, a fast ballot round FB consists:

FB_d The Paxos instance decree id.

FB_{ld} The decree id of the leader elected.

FB_{lds} The decree id of the leader's *lds*.

FB_V The fast round decree value.

FB_{vot} The nodes set voted for the fast round decree value.

In fast round ballot FB and $FB^{'}$ with same FB_d, and $FB_d \geq FB_{lds}$.

If $FB_{ld} > FB^{'}_{ld}$

Then $FB^{'}_V$ from leader $FB^{'}_{ld}$ is not chosen in fast ballot round $FB^{'}$.

Proof:

Assume, $FB^{'}_V$ from leader $FB^{'}_{ld}$ has been chosen in fast ballot round.

(1) Based on Paxos value chosen rule, $FB^{'}_{vot}$ is a majority set $Q^{'}_v$. All nodes in the majority set $Q^{'}_v$ voted $FB^{'}_V$, in leader's fast ballot round, which leader is elected in decree id $FB^{'}_{ld}$.

(2) $FB_d \geq FB_{lds}$. There exist a majority set Q_{dp}, with each node promised not to accept fast write round greater equal than FB_{lds} from earlier leaders.

(3) Obvious, $FB_{ld} > FB^{'}_{ld}$ All nodes in the majority set Q_{dp} do not accept $FB^{'}_V$ in the earlier leaders fast ballot round.

(4) $Q_{v'} \cap Q_{dp} \neq \emptyset$

There is a node that accepted $FB^{'}_{dec}$ and not accepted $FB^{'}_{dec}$ in the earlie leaders fast round ballot, which leader is elected in decree id $FB^{'}_{ld}$.

Obviously, we reach a contradiction.

IV. Engineering Implementation

In engineering, it is a non-trivial task [17] to implement high performance decree promise Paxos. To fulfill the availability requirement, DP Paxos has to allow multi-nodes write, in worst case, collision is not negligible. DP Paxos focus on 3-node cluster optimization.

A. Odd-even Gear and reserve mechanism

In Decree Promise Paxos implementation, leader node can do fast rounds while other nodes can do normal rounds. Normally, default router rule can define that client requests are routed to the leader, leader commit all commands in two communication delay. During network partition or node failure, multi-node write will happen. After receiving a client request, the follower node has several choices.

(1) Redirect the request to the leader and the leader process it with the new decree id and fast round. It looks good, but if the leader is down or not reachable, the system will totally unavailable for a short period of time. Besides, new leader election relays on some election lease time-outs. Like Raft [7].

(2) Follower tries to propose the requests (decree) with Normal Ballot round. If the leader is down and majority of the nodes are alive, the system keeps working. Follower as the busiest node candidate to be the new leader and keep the system available.

Decree Promise Paxos uses the second strategy. In 3-Node cluster N1, N2, N3. Normally, two of then, for example: N1 and N3, are configured to receive client requests, N2 configure as a pure follower legislator node, client requests will not send to N2, so N2 will never be a leader candidate. By default, clients send request to N1, if N1 is unreachable for some clients, they can send to N3. The busiest legislator node will elect to be leader, so the worst case is half proposal from leader and another half requests from a follower node. Each Paxos instance chosen a unique consensus decree value, heavy collision will cause lots of client requests fail. We try to pre-allocate the decree id to reduce collision. The odd-even gear mechanism run different decree id pick up strategy on leader and follower. Leader node occupy the smallest idle decree id to run Paxos instance. Idle means the leader not detect other legislator node occupy the decree id locally. While, the follower legislator node take the smallest idle even decree id, if it receive client request. If the leader node and the follower node process same amount of client requests, the system works just like a gear. Decree id increase on both node in same speed. In half to half status, odd-even gear reduce the collision (two node proposal with same decree id) to unbelievable rarely. Normally, follower process much less client requests than leader, follower proposal very easy to conflict with leader. Reserve mechanism is introduced.

Once conflicted with another proposer, the winner node could reserve (pre-prepare) a decree id for the loser node. Winner node pick up another idle decree id as the odd-even gear mechanism and locally execute phase 1, prepared for loser, then reply the promise message with a new reserved decree id. Once the loser node receive the reserve prepare, it could start a vote or prepare with the newly reserved decree id for the conflicted and failed client requests.

In our practice, 3 server nodes form a server cluster which allows the system to tolerate one failure. Client requests only needs to route to 2 nodes. That means during network partition

TABLE I
RESERVE DECREE MECHANISM

Decree ID	Node1 Leader	Node2 Follower	Node3 Follower
16	*Voted(16,0,V_w^{16})*	*Voted(16,0,V_w^{16})*	*Preparing(16,3,V_l^{16})*
	Chosen V_w^{16}	*Chosen V_w^{16}*	*Chosen V_w^{16}*
	Reserve 19 Promised(19,3)		*Reserve 19 for V_l^{16} Promised(19,3)*
17	*Voted(17,0,V_w^{17})*	*Voted(17,0,V_w^{17})*	*Voted(17,0,V_w^{17})*
	Chosen V_w^{17}	*Chosen V_w^{17}*	*Chosen V_w^{17}*
18	*Voted(18,0,V_w^{18})*	...	...
19	*Promised(19,3)*		*Promised(19,3)*
	Voted(19,3,V_l^{16})	*Voted(19,3,V_l^{16})*	*Voted(19,3,V_l^{16})*
	Chosen V_l^{16}	*Chosen V_l^{16}*	*Chosen V_l^{16}*

TABLE II
ASYMMETRIC PRIORITY

Priority	Leader Proposer	Follower	Follower Proposer
0	N/A	Vote(0) Promise(id)	Promising(id)
1	Vote(0)	First arrive higher priority	Vote(0)
2	N/A	N/A	Promise(id)
3	Promising(id+M)	N/A	Promising(id+M)
4	Promise(id+M)	Promise(id+M)	Promise(id+M)
5	Promising(id+2M)	N/A	Promising(id+2M)
6	Promise(id+2M)	Promise(id+2M)	Promise(id+2M)
7	Promising(id+nM)	N/A	Promising(id+nM)
8	Promise(id+nM)	Promise(id+nM)	Promise(id+nM)
higher	...	...	...

id: Proposer node id, M:Max Proposer node id, 0:lowest priority

or node failure, at most 2 nodes may receive client requests. Once conflict occurs, the winner node reserves and promises a decree id for the loser node. Once the loser node receives the winners commit contain reserved decree id, it can modify the Paxos instance to the reserved decree id, promise locally and start vote immediately, solve the conflict not cost more communication, because winner and loser form a majority set in 3-nodes cluster. If in 5 server nodes or greater clusters, 2 nodes promise does not constitute a majority, loser node could run a new Paxos instance with the reserved decree id.

Table (I) shows the client requests route to node 1 or node 3. Node 2 never propose any client value.

Node 1 and node 3 proposal different values in a Paxos instance with decree id 16. The Paxos instance of decree id 16 chosen V_w^{16} , Node 1 is winner. Node 1 reserve and promise decree id 19 for loser Node 3 to proposal the conflicted value V_l^{16}. Node 3 skips prepare in Decree 19, due to Node 1 and Node 3, is majority set already. The winner node pick up a reserve decree id with the odd-even gear mechanism to reduce re-confilct.

B. Traffic Control

As we describe, leader has the authority to fast writes, it is more effective than ordinary Paxos instance. So, when a fast round conflicts with a normal round, it is better to halt the normal round and let the fast round go first. Asymmetric priority is defined to make sure that only one node keeps going and the others halt for time-out retry when there is a conflict. Besides, fast round uses 0 as ballot number which is smaller than any ballot number in the normal round. Promising status is defined to provide lower priority than ballot number 0.

Promising: If a node plays the legislator and proposer role, when starting a normal round with ballot number n, before it receives remote nodes promise, the local status is promising n. In our project, the minimum promising n for each node is its node id. After receiving enough remote promise responses, local prepare can be done. The *Promising n=nodeid*, is defined as the lowest priority.

Normal round starts with local Promising node id. After receiving remote promise response, local prepare is executed.

Fast round starts with local *vote(0)* directly. Then it broadcasts begin-ballot request.

If a normal round and fast round start at the same time, with the same decree id, fast round proposer starts with local **Vote**(0). Normal round proposer starts with local promising id. When it receives higher priority operation in promising status, like fast round **Ballot**(*0,v*), it will accept and respond. Normal round procedure will halt for time-out retry, then the fast round keeps going.

If the normal round prepares successfully, before fast round **Ballot**(*0,v*) arrives, normal round will keep going. The fast round will wait for time-out until commit is received. Usually lower priority will be ignored without any response. Lazy strategy could save cost and avoid active collision.

If the fast round proposer **Ballot**(*0,v*) times out, the next attempt at prepare will go with the ballot number that is at least (id+M), because some other node id may be larger than the leaders node id. When no commit message is received after the fast round time out, some failure might have taken place. Retrying with node id, may cause double time-out due to priority set.

As the busiest legislator node is work as leader, most clients requests are processed with fast writes. Odd-even Gear mechanism reduce the worst conflict, once collision happens, the traffic control improve fast round priority to keep the lower-cost fast writes going and halt the normal round. The failed normal round is transfer to another reserved decree id by the winner, in 3-node cluster the reserved Paxos instance skip Phase 1, so reserve mechanism do not spend more communication.

V. PERFORMANCE DATA

A Decree Promise Paxos project has been developed. It uses Co-routine to manage the Paxos procedure context. So, it is named DP CoPaxos. Tests below use a 3-nodes cluster, and run on tlinux server with E5-2620v3 processor. In fact, the test cases only run with single process and single workload thread. More instances being deployed will bring much greater performance. Testing Decree data length is about 60 bytes.

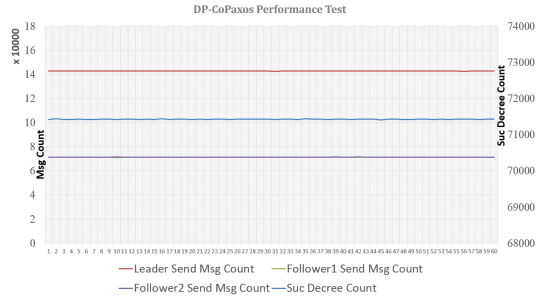

Fig. 1. Three Nodes Cluster Normal Working Fast Write

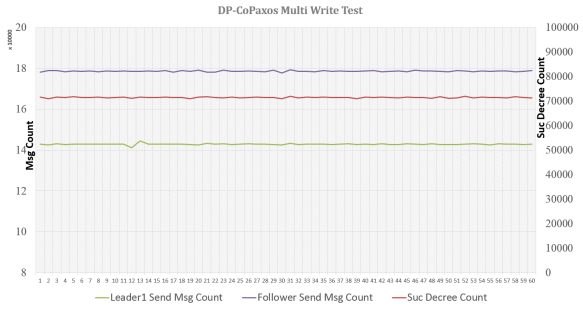

Fig. 2. Three Nodes Cluster Multi Write

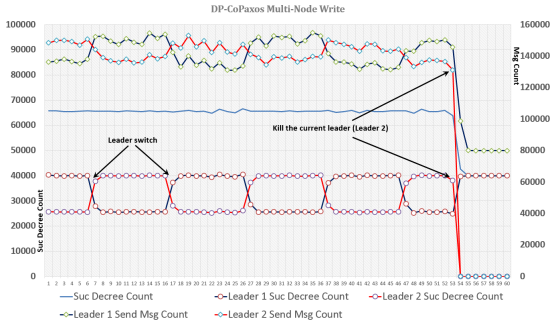

Fig. 3. Leader switch with Multi-Nodes Write.

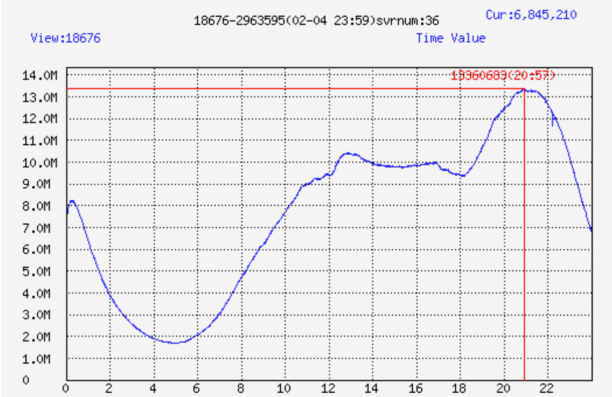

Fig. 4. PaxosMirror Writes Track (Tencent Monitor System).

A. Normal working fast write test

Figure (1) shows, DP CoPaxos basic running mode. The leader can process 71,000 requests per second, with each running a DP fast round. More or less, 142,000 messages are sent by the leader per second. Pass a decree needs ballot and commit to other 2 follower nodes. In the system, batch commit information was attached with next ballot message. With the combine trick, messages from leader are reduced to 142,000 per second. The test case run with 2 legislators as followers. Follower respond vote message only. Each decree costs 2 messages sent by the leader.

B. Multi Write test

Figure (2) is DP CoPaxos multi write test, leader node proposes about 35,700 decrees per second, a follower node proposes 35,700 decrees per second too. In the heavy conflict scenario, system keeps working stable. Leaders sent message count is 142,800 per second. It contain 35,700 fast round ballot broadcast to other 2 nodes (71,400 messages in all), 35,700 promise response to follower proposer and 35,700 vote response to follower proposer. It meaning that almost all fast write decrees succeed with fast round. The follower proposer sent 178,500 messages per second. It contain 35,700 vote response for leader proposer's fast round and 35,700 normal round prepare broadcast and 35,700 normal round ballot broadcast, almost all normal write decrees succeed without retry. For each fast write, the leader send 2 messages (broadcast ballot to other 2 nodes). For each normal write, the follower send 4 messages (broadcast prepare and ballot to other 2 nodes). In heavy split-brain status, the system keep consistency and available, meanwhile it keeps the same remarkable performance as normal working condition.

C. Leader Switch with Multi Write

In this test, we switch leaders every 10 seconds. Leader proposal 40,000 decrees per seconds and follower proposal 25,600 decrees to build heavy conflict scenario. All decrees are succeed, system have smooth throughput and works stable. Almost all the successful decrees proposed by the leader go with fast round, and have 2 ballot messages sent by leader to the other 2 legislator nodes on average. Each successful decree proposed by follower with normal round has 4 messages (2 prepare messages and 2 Ballot messages) sent by follower proposer on average. Conflicting decrees not increase the communication cost. At 53th seconds in Figure (3), the leader node (leader 2) was killed without any predicted warning, it's throughput drops to 0. Meanwhile node 1 turns to new leader immediately, system keeps working, all proposed decrees from node 1 succeed, system keep available seamlessly.

D. Online Service Data

PaxosMirror is the data cache service of the Happy Poker online game. A Paxos group consists of 3 PaxosMirror server process. Each day there are 10.9 billion writes in all nodes, which means 3.6 billion writes from clients were processed. More than 50 million players update data in PaxosMirror each day. In the past 6 months, we switched leaders and rebooted

servers in the purpose of updating version or testing, and the whole system always stays available and all data (up to 13.9 billion data nodes) keep consistency. Much larger scale application based on Decree Promise Paxos is planned.

VI. Conclusion

In the past decades, some well-known Paxos based projects were developed. Googles Chubby [8] is a reliable distributed lock service. The GFS [18], MegaStore [9] and Bigtable [19] use Chubby in several ways. DP Paxos as a pipelined single-phase Paxos extension without lease could provide better availability with same cost.

The Decree Promise Paxos, stems from the core of Paxos theory. The algorithm defined 0 fast Round to save cost, keep the final target and solve the multi-fast-round voted confusion with significantly lower costs. The Decree level Prepare-Promise actually has the similar features as the normal Prepare-Promise. With Decree Promise design we got some progress: (1) Under normal conditions, the server cluster leader could concurrently run with fast Round which only needs the majority of nodes alive. This leads to lower costs and better performance in general.

(2) The Follower nodes run decrees in parallel with normal round when some client requests reach.

(3) With the No lease design, the system keeps working with normal rounds when the leader goes down or is not stable. As long as the majority of all nodes are available, then the system will be functional. The system can make a smooth transition to the next leader, without unavailable gap when majority nodes are available.

(4) Implementation with engineering techniques and tricks to help avoid unnecessary competitions, allowing fewer time-outs and retries.

Consistency is guaranteed in any cases. With Decree Promise Paxos, it is possible to implement widely useful and reliable infrastructure services with better performance and lower cost.

VII. Acknowledgments

It has been an exciting adventure working on the research of Paxos theory and its engineering implementation. Although the process has been painful and accompanied by self-doubts, We are able to persevere and complete our work with the help of many people. Many thanks to Zhiming Nie, Yao Lu, Wu Hu, Jian tang and Feng Ai, who are Tencents specialists in distributed system designs. Theirs invaluable advice, encouragement and recommendations, helped us advance the work. Special thanks to Mr. Jiatao Xu and his PaxosStore team. The Paxos-based PaxosStore is a successful project in Tencent. Together, we share creative ideas and discuss solutions. Theirs restless efforts and aspirations for making the data service even better motivated me to keeping on working. Thanks for Mr. Ruoxu Yang and Mr. Andrew Seid providing strong support to polish and revision this paper, that indeed helps a lot. Thanks JIAT and SDIAT, for their strong support to improve and publish the research. Finally, thanks to our colleagues, whose ethics and dedications to deliver the finest services and products to our users encouraged us to focus on this research.

References

[1] LAMPORT L. The part-time parliament. ACM Transactions Computer Systems 16, 2 (May 1998), 133169.

[2] LAMPORT L. Paxos made simple. ACM SIGACT news 32, 4 (Dec 2001), 18-25.

[3] LAMPORT L. Fast Paxos. Distributed Computing 19, 2 (2006), 79-103.

[4] LAMPORT L, Massa M. Cheap Paxos. The International Conference on Dependable Systems and Networks, DSN (2005)

[5] Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., et al. Spanner: Google's globally-distributed database. Usenix Conference on Operating Systems Design and Implementation, 2012, Vol.31, pp.251-264.

[6] LAMPORT L. Generalized consensus and Paxos. Tech. Rep. MSR-TR-2005-33, Microsoft Research, (2005)

[7] Ongaro D, Ousterhout J, In Search of an Understandable Consensus Algorithm. Draft of October, (2014)

[8] Burrows M. The Chubby lock service for loosely-coupled distributed systems. Symposium on Operating Systems Design and Implementation. USENIX Association, (2006), 335-350.

[9] Baker J, Bond C, Corbett J, et al. Megastore: Providing Scalable, Highly Available Storage for Interactive Services. CIDR 2011, Fifth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, (Jan 2011), Online Proceedings. DBLP, (2011), 223-234.

[10] Brasileiro F, Greve F, Mostfaoui A and Raynal M. Consensus in One Communication Step. International Conference on Parallel Computing Technologies. Springer-Verlag, 2001:42-50.

[11] Gray J, Lamport L. Consensus on transaction commit. Acm Transactions on Database Systems. 2006, 31(1):133-160.

[12] Lampson B W. How to build a highly available system using consensus. International Workshop on Distributed Algorithms. Springer-Verlag, 1996:1-17.

[13] Prisco R D, Lampson B W, Lynch N A. Revisiting the Paxos Algorithm. International Workshop on Distributed Algorithms. Springer-Verlag, 1997:111-125.

[14] Davidson S B, Garcia-Molina H, Skeen D. Consistency in Partitioned Networks. Acm Computing Surveys, 1985, 17(3):341-370.

[15] Gray C, Cheriton D. Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. Acm Sigops Operating Systems Review, 1989, 23(23):202-210.

[16] Santos N. Optimizing Paxos with batching and pipelining. Elsevier Science Publishers Ltd. 2013.

[17] Chandra, Tushar D, Griesemer, et al. Paxos made live: an engineering perspective. Proceedings of Annual Acm Symposium on Principles of Distributed Computing, 2007:398-407.

[18] Ghemawat S, Gobioff H, Leung S T. The Google file system. Acm Sigops Operating Systems Review, 2003, 37(5):29-43.

[19] Chang F, Dean J, Ghemawat S, et al. Bigtable: A Distributed Storage System for Structured Data. ACM Transactions on Computer Systems (TOCS), 2008, 26(2):4.

[20] Boichat R, Dutta P, Guerraoui R. Deconstructing paxos. Acm Sigact News, 2003, 34(1):47-67.