A SADL Appendix

A.1 SADL Algorithm

Algorithm 2 SADL Algorithm for process p_i, $i \in 0..n-1$

Local State:
- lastCompletedRounds[], an array of n elements (with n the number of replicas) that keeps track of the last SADL-batch for which at least $n - f$ SADL-votes were collected for each replica
- chains[][], a 2D array that saves the i^{th} SADL-batch created by replica p_j
- buffer, a queue storing incoming client commands
- awaitingAcks $\leftarrow$ **false**, a boolean variable which states whether this replica is waiting for SADL-votes

Require: maximum batch time and batch size

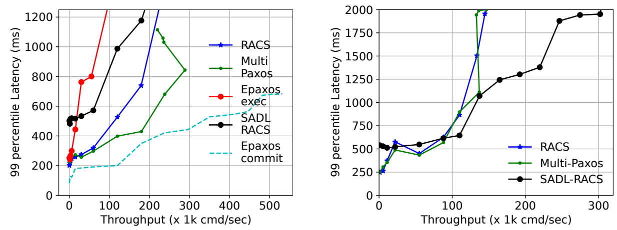
Upon receiving a batch of client commands *cl*:
  push *cl* to buffer

Upon (size of incoming buffer reaching batch size **or** maximum batch time is passed) **and** awaitingAcks is **false**:
  B$_{\text{parent}}$ $\leftarrow$ the SADL-batch corresponding to chains[p_i][lastCompletedRounds[p_i]] (i.e. B$_{\text{parent}}$ is the last SADL-batch proposed by p_i that received $n - f$ SADL-votes)
  B $\leftarrow$ (lastCompletedRounds[p_i]+1, B$_{\text{parent}}$, buffer.popAll()) // create new SADL-batch
  isAwaiting $\leftarrow$ **true**
  broadcast <new-SADL-Batch, B>

Upon receiving <new-SADL-batch, B> from p_j
  chains[p_j][B.round] $\leftarrow$ B
  lastCompletedRounds[p_j] $\leftarrow$ B.parent.round
  send <SADL-vote, B.round> to p_j

Upon receiving $n - f$ <SADL-vote, r> for the same r **and** r = lastCompletedRounds[p_i]+1 **and** awaitingAcks is **true**
  awaitingAcks $\leftarrow$ **false**
  lastCompletedRounds[p_i]+=1

procedure getClientCommands()
  return lastCompletedRounds



(a) 99% Latency 3 replicas **(b)** 99% Latency 11 replicas

Figure 9. Throughput versus tail latency for WAN normal-case execution, comparing pipelined RACS and SADL-RACS to pipelined Multi-Paxos and pipelined Epaxos with 3 and 11 replica ensembles

A.2 Correctness and Complexity

Proof of Availability: A replicate(B) operation succeeds when B is created and sent to all the replicas, and only after receiving at least $n - f$ SADL-votes. Since each replica saves B in the *chains* array, it is guaranteed that B will persist as long as $n - f$ replicas are alive due to quorum intersection. Hence fetch(B) eventually returns.

Proof of Causality: Causality follows from the fact that each replica extends its chain of SADL-batches, and because each replica creates a batch with round r only after completing the replicate operation of the batch with round $r - 1$.

Complexity The SADL algorithm has a linear complexity: for each batch of client commands, one SADL-batch is broadcast to all replicas and each of these replicas replies to the sender with a <SADL-vote>.

A.3 Hybrid SADL-pipelining protocol

The hybrid SADL-pipelining protocol involves two steps: (1) a calibration phase and (2) a deployment phase. In the calibration phase, the system administrator first deploys SADL-RACS and pipelined-RACS protocols, separately, similar to Fig. 6b, in the given replica and front-end setup, and obtains the throughput versus median latency relationship, which we refer to as the *performance table*.

Then, in the hybrid SADL-pipelining deployment, the RACS layer first starts to replicate command batches using pipelining (without SADL) and the synchronous path leader L_v of RACS monitors the throughput and the median latency. When the median latency reaches the saturation median latency of RACS according to the *performance table*, the RACS layer automatically proposes a reconfiguration command to enable the SADL instead of pipelining. To consistently enable the reconfiguration across all the participating replicas, we use a similar method to Raft replica set reconfiguration (see [43]: section 6 Cluster membership changes). Once the reconfiguration takes effect, all the replicas switch to SADL. Similarly, if the throughput drops beyond a threshold w.r.t the *performance table*, the RACS leader automatically proposes a reconfiguration to switch back to pipelining.

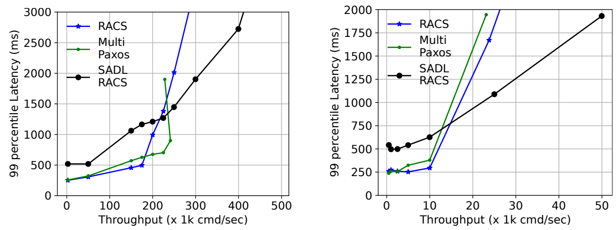
With hybrid SADL-pipelining enabled, SADL-RACS delivers the optimal throughput and latency, for all arrival rates.

B Evaluation Supplementary

B.1 Scalability Tail Latency

Fig. 9 depicts the tail latency of RACS and SADL-RACS under different replica sizes.

Fig. 10 depicts the tail latency of RACS and SADL-RACS under different payload sizes.



(a) 99 % Latency 17B **(b)** 99% Latency 265B

Figure 10. Throughput versus tail latency for WAN normal-case execution, comparing pipelined RACS and SADL-RACS to pipelined Multi-Paxos with 17B and 265B payload sizes

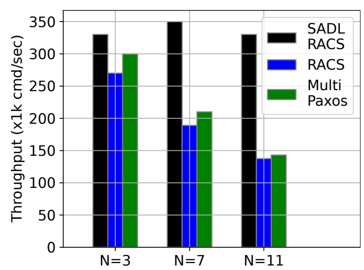
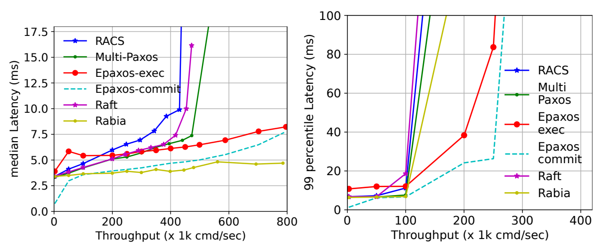


Figure 11. WAN scalability with Redis backend



(a) Median Latency on LAN **(b)** 99% Latency on LAN

Figure 12. Throughput versus latency for LAN normal-case execution, comparing RACS to Rabia, Multi-Paxos, EPaxos, and Raft using 5 replicas

B.2 Scalability with Redis

Fig. 11 depicts the scalability of SADL-RACS, w.r.t increasing number of replicas, when deployed with a real-world benchmark; Redis[37] as the backend. In this experiment, we measure the saturation throughput, under 800ms median latency upper bound. Fig. 11 confirms that SADL-RACS can sustain a throughput of at least 330k cmd/sec, when the replication factor is increased from 3–11, in-contrast, Multi-Paxos and RACS sacrifice the throughput, when scaling to higher replication factors. Hence we re-confirm the claim **C4.1**.

B.3 RACS LAN Normal Case Performance

We designed RACS and SADL for the WAN, however, for the completeness of the evaluation, we also present the LAN performance. Fig. 12 depicts the experiment results, where we experiment with 5 replicas and 5 clients, deployed in the same AWS region.

RACS vs Multi-Paxos and Raft: We first observe that RACS achieves a saturation throughput of 420k cmd/sec, under a median latency upper bound of 10ms, that is comparable to the saturation throughput of Multi-Paxos (450k) and Raft (440k). Under normal case executions, RACS, Multi-Paxos, and Raft have 1 round-trip latency, serialized through a leader, hence provide comparable performance.

RACS vs EPaxos: Second, we observe that Epaxos outperforms RACS, both in terms of latency and throughput. We use similar reasoning as Section 7.3, to illustrate this behaviour.

RACS vs Rabia: Finally, we observe that Rabia delivers 800k cmd/sec throughput under 5ms median latency, outperforming RACS by 380k cmd/sec in throughput, and by 5ms in median latency. Rabia uses multiple leaders, thus avoids the leader bottleneck present in RACS. Moreover, Rabia relies on the "natural" ordering of messages inside a data centre, where it is guaranteed that a majority of the replicas receive the same message, sent at the same time. Rabia exploits this data-centre-specific "natural" message ordering and uses Ben-Or[7] to check whether the network ordering indeed has achieved consensus (or not). This makes Rabia messages lightweight because the consensus messages no longer have to carry the payload. In contrast, RACS, Multi-Paxos and Raft carry the request payload inside the consensus messages, thus incurring higher latency. Rabia's approach of using consensus to confirm the "natural" ordering inside a data centre can readily be combined with RACS, Multi-Paxos and Raft, however, falls outside the scope of this work.

C RACS Formal Proofs

C.1 Definition

elected-asynchronous block: We refer to an asynchronous block B_f generated in view v with level 2 as an elected-asynchronous block, if the common-coin-flip(v) returns the index of the proposer p_l who generated B_f in the view v and if the <asynchronous-complete> for B_f exists in the first $n-f$ <asynchronous-complete> messages received. An elected-asynchronous block is committed same as a synchronously-committed block.

C.2 Proof of safety

We first show that for a given rank (v,r), there exists a unique block. In the following lemmas C.1,C.2,C.3,C.4,C.5, and C.6 we consider different formations of blocks with the same rank.

Lemma C.1. *Let B, $\tilde{B}$ be two synchronous blocks with rank (v,r). Then B and $\tilde{B}$ are the same.*

Proof. Assume by contradiction B and $\tilde{B}$ to be different. Then, according to line 15, both B and $\tilde{B}$ were created by the leader of view v, L_v. Assume L_v created B first and then $\tilde{B}$. Then, by construction, $\tilde{B}$ has a rank greater than (v,r). Hence, a contradiction. Thus $B = \tilde{B}$. ◻

Lemma C.2. *Let B, $\tilde{B}$ be two elected-asynchronous blocks with rank (v, r). Then B and $\tilde{B}$ are the same.*

Proof. Assume by contradiction B and $\tilde{B}$ to be different. Then, according to line 56-58, both leaders who sent B and $\tilde{B}$ in an <asynchronous-complete> message were elected with the same common-coin-flip(v). Since no replica can equivocate, i.e. no replica sends <asynchronous-complete> message for two different blocks with the same rank (v, r), and because the common-coin-flip(v) returns a unique leader for each v, this is a contradiction. Thus $B = \tilde{B}$. ◻

Lemma C.3. *Let B, $\tilde{B}$ be two asynchronous blocks with rank (v, r) and level 1, such that both blocks are parents of an elected-asynchronous block of the same view v. Then B and $\tilde{B}$ are the same.*

Proof. Assume by contradiction B and $\tilde{B}$ to be different. Then both B and $\tilde{B}$ can have a distinct child level 2 elected-asynchronous block with rank $(v, r + 1)$ (see line 49). According to lemma C.2, this is a contradiction. Thus $B = \tilde{B}$. ◻

Lemma C.4. *Let B be a synchronous block which receives $n-f$ <vote>s. Then there cannot exist a level 1 asynchronous block $\tilde{B}$ that is a parent of a level 2 elected-asynchronous block where B and $\tilde{B}$ have rank (v, r).*

Proof. Assume by way of contradiction that $\tilde{B}$ exists. Because B received $n - f$ votes, at least $n - f$ replicas saw B before $\tilde{B}$ (see line 21). $\tilde{B}$ has received $n - f$ <vote-async> (see line 46) from replicas who could not have seen B before (see line 40). Because $n - f > \frac{n}{2}$, this is a contradiction. Hence $\tilde{B}$ does not exist. ◻

Lemma C.5. *Let B be a synchronous block which receives $n-f$ votes with rank (v, r). Then there cannot exist an elected-asynchronous block $\tilde{B}$ of level 2 with rank (v, r).*

Proof. Assume by way of contradiction that $\tilde{B}$ exists. Because B received $n - f$ votes, at least $n - f$ replicas saw B before $\tilde{B}$ (see line 21). $\tilde{B}$ has received $n - f$ <vote-async> (see line 46) from replicas who could not have seen B before (see line 40). Because $n - f > \frac{n}{2}$, this is a contradiction. Hence $\tilde{B}$ does not exist. ◻

Lemma C.6. *Let B be a level 1 asynchronous block that is the parent of level 2 elected-asynchronous block in the same view. Then there cannot exist a level 2 elected-asynchronous block $\tilde{B}$ with rank (v, r).*

Proof. Assume by way of contradiction that $\tilde{B}$ exists. The level 1 parent block of $\tilde{B}$ had rank $(v, r - 1)$ (see line 49) and was created after receiving $n - f$ timeout messages with rank $(v, r - 2)$ (see line 37). On the other hand, B was created after receiving $n-f$ timeout messages with rank $(v, r-1)$. Because $n-f > \frac{n}{2}$, this is a contradiction. Hence $\tilde{B}$ does not exist. ◻

Theorem C.7. *Let B and $\tilde{B}$ be two blocks with rank (v, r). Each of B and $\tilde{B}$ can be of type: (1) synchronous block which collects at least $n-f$ votes or (2) elected-asynchronous block or (3) level 1 asynchronous block which is a parent of an elected-asynchronous block. Then $\tilde{B}$ and B are the same.*

Proof. This holds directly from Lemma C.1, C.2, C.3, C.4, C.5 and C.6. ◻

Theorem C.8. *Let B and $\tilde{B}$ be two adjacent blocks, then $\tilde{B}.r = B.r + 1$ and $\tilde{B}.v \geq B.v$.*

Proof. According to the algorithm, there are three instances where a new block is created.

- Case 1: when $isAsync$ = false and L_v creates a new synchronous block by extending the $block_{high}$ with rank (v,r)(see line 15). In this case, L_v creates a new block with round $r+1$. Hence the adjacent blocks have monotonically increasing round numbers.
- Case 2: when $isAsync$ = true and upon collecting $n - f$ <timeout> messages in view v (see line 32). In this case, the replica selects the $block_{high}$ with the highest rank (v, r), and extends it by proposing a level 1 asynchronous block with round $r + 1$. Hence the adjacent blocks have monotonically increasing round numbers.
- Case 3: when $isAsync$ = true and upon collecting $n - f$ <vote-async> messages for a level 1 asynchronous block (see line 46-47). In this case, the replica extends the level 1 block by proposing a level 2 block with round $r + 1$. Hence the adjacent blocks have monotonically increasing round numbers.

The view numbers are non decreasing according to the algorithm. Hence Theorem C.8 holds. ◻

Theorem C.9. *If a synchronous block B_c with rank (v, r) is committed, then all future blocks in view v will extend B_c.*

Proof. We prove this by contradiction.

Assume there is a committed block B_c with $B_c.r = r_c$ (hence all the blocks in the path from the genesis block to B_c are committed). Let block B_s with $B_s.r = r_s$ be the round r_s block such that B_s conflicts with B_c (B_s does not extend B_c). Without loss of generality, assume that $r_c < r_s$.

Let block B_f with $B_f.r = r_f$ be the first valid block formed in a round r_f such that $r_s \geq r_f > r_c$ and B_f is the first block from the path from genesis block to B_s that conflicts with B_c; for instance B_f could be B_s. L_v forms B_f by extending its $block_{high}$ (see line 15). Due to the minimality of B_f (B_f is the first block that conflicts with B_c), $block_{high}$ contain either B_c or a block that extends B_c. Since $block_{high}$ extends B_c, B_f extends B_c, thus we reach a contradiction. Hence no such B_f exists. Hence all the blocks created after B_c in the view v extend B_c. ◻

Theorem C.10. *If a synchronous block B with rank (v, r) is committed, an elected-asynchronous block $\tilde{B}$ of the same view v will extend that block.*

Proof. We prove this by contradiction. Assume that a synchronous block B is committed in view v and an elected-asynchronous block $\tilde{B}$ does not extend B. Then, the parent level 1 block of $\tilde{B}$, $\tilde{B_p}$, also does not extend B.

To form the level 1 $\tilde{B_p}$, the replica collects $n-f$ <timeout> messages (see line 32), each of them containing the $block_{high}$. If B is committed, by theorem C.9, at least $n-f$ replicas should have set (and possibly sent) B or a block extending B as the $block_{high}$. Hence by intersection of the quorums $\tilde{B_p}$ extends B, thus we reach a contradiction. □

Theorem C.11. *At most one level 2 asynchronous block from one proposer can be committed in a given view change.*

Proof. Assume by way of contradiction that 2 level 2 asynchronous blocks from two different proposers are committed in the same view. A level 2 asynchronous block B is committed in the asynchronous phase if the common-coin-flip(v) returns the proposer of B as the elected proposer (line 56). Since the common-coin-flip(v) outputs the same elected proposer across different replicas, this is a contradiction. Thus all level 2 asynchronous blocks committed during the same view are from the same proposer.

Assume now that the same proposer proposed two different level 2 asynchronous blocks. According to the line 49, and since no replica can equivocate, this is absurd.

Thus at most one level 2 asynchronous block from one proposer can be committed in a given view change. □

Theorem C.12. *Let B be a level 2 elected-asynchronous block that is committed, then all blocks proposed in the subsequent rounds extend B.*

Proof. We prove this by contradiction. Assume that level two elected-asynchronous block B is committed with rank (v, r) and block $\tilde{B}$ with rank $(\tilde{v}, \tilde{r})$ such that $(\tilde{v}, \tilde{r}) > (v, r)$ is the first block in the chain starting from B that does not extend B. $\tilde{B}$ can be formed in two occurrences: (1) $\tilde{B}$ is a synchronous block in the view $v+1$ (see line 15) or (2) $\tilde{B}$ is a level 1 asynchronous block with a view strictly greater than v (see line 37). (we do not consider the case where $\tilde{B}$ is a level 2 elected-asynchronous block, because this directly follows from C.7)

If B is committed, then from the algorithm construction it is clear that a majority of the replicas will set B as $block_{high}$. This is because, to send a <asynchronous-complete> message with B, a replica should collect at least $n-f$ <vote-async> messages (see line 46). Hence, its guaranteed that if $\tilde{B}$ is formed in view v+1 as a synchronous block, then it will observe B as the $block_{high}$, thus we reach a contradiction.

In the second case, if $\tilde{B}$ is formed in a subsequent view, then it is guaranteed that the level 1 block will extend B by gathering from the <timeout> messages B as $block_{high}$ or a block extending B as the $block_{high}$ (see line 37), hence we reach a contradiction. □

Theorem C.13. *There exists a single history of committed blocks.*

Proof. Assume by way of contradiction there are two different histories H_1 and H_2 of committed blocks. Then there is at least one block from H_1 that does not extend at least one block from H_2. This is a contradiction with theorems C.9, C.10 and C.12. Hence there exists a single chain of committed blocks. □

Theorem C.14. *For each committed replicated log position r, all replicas contain the same block.*

Proof. By theorem C.8, the committed chain will have incrementally increasing round numbers. Hence for each round number (log position), there is a single committed entry, and by theorem C.7, this entry is unique. This completes the proof. □

C.3 Proof of liveness

Theorem C.15. *If at least $n-f$ replicas enter the asynchronous phase of view v by setting isAsync to true, then eventually they all exit the asynchronous phase and set isAsync to false.*

Proof. If $n-f$ replicas enter the asynchronous path, then eventually all replicas (except for failed replicas) will enter the asynchronous path as there are less than $n-f$ replicas left on the synchronous path due to quorum intersection, so no progress can be made on the synchronous path (see line 28) and all replicas will timeout (see line 30). As a result, at least $n-f$ correct replicas will broadcast their <timeout> message and all replicas will enter the asynchronous path.

Upon entering the asynchronous path, each replica creates a asynchronous block with level 1 and broadcasts it (see line 37-38). Since we use FIFO perfect point-to-point links, eventually all the level 1 blocks sent by the $n-f$ correct replicas will be received by each replica in the asynchronous path. At least $n-f$ correct replicas will send them <vote-async> messages if the rank of the level 1 block is greater than the rank of the replica (see line 40-41). To ensure liveness for the replicas that have a lower rank, the algorithm allows catching up, so that nodes will adopt whichever level 1 block which received $n-f$ <vote-async> arrives first. Upon receiving the first level 1 block with $n-f$ <vote-async> messages, each replica will send a level 2 asynchronous block (see line 46-50), which will be eventually received by all the replicas in the asynchronous path. Since the level 2 block proposed by any block passes the rank test for receiving a <vote-async>, eventually at least $n-f$ level 2 blocks get $n-f$ <vote-async> (see line 41). Hence, eventually at least $n-f$ replicas send the <asynchronous-complete> message (see line 53), and exit the asynchronous path. □

Theorem C.16. *With probability $p > \frac{1}{2}$, at least one replica commits an elected-asynchronous block after exiting the asynchronous path.*

Proof. Let leader L be the output of the common-coin-flip(v) (see line 56). A replica commits a block during the asynchronous mode if the <asynchronous-complete> message from L is among the first $n - f$ <asynchronous-complete> messages received during the asynchronous mode (see line 57), which happens with probability at least greater than $\frac{1}{2}$. Hence with probability no less than $\frac{1}{2}$, each replica commits a chain in a given asynchronous phase. ◻

Theorem C.17. *A majority of replicas keep committing new blocks with high probability.*

Proof. We first prove this theorem for the basic case where all replicas start the protocol with $v = 0$. If at least $n - f$ replicas eventually enter the asynchronous path, by theorem C.15, they eventually all exit the asynchronous path, and a new block is committed by at least one replica with probability no less than $\frac{1}{2}$. According to the asynchronous-complete step (see line 64), all nodes who enter the asynchronous path enter view $v = 1$ after exiting the asynchronous path. If at least $n - f$ replicas never set *isAsync* to true, this implies that the sequence of blocks produced in view 1 is infinite. By Theorem C.8, the blocks have consecutive round numbers, and thus a majority replicas keep committing new blocks.

Now assume the theorem C.17 is true for view $v = 0, ..., k - 1$. Consider the case where at least $n - f$ replicas enter the view $v = k$. By the same argument for the $v = 0$ base case, $n - f$ replicas either all enter the asynchronous path commits a new block with $\frac{1}{2}$ probability, or keeps committing new blocks in view k. Therefore, by induction, a majority replicas keep committing new blocks with high probability. ◻

Theorem C.18. *Each client command is eventually committed.*

Proof. If each replica repeatedly keeps proposing the client commands until they become committed, then eventually each client command gets committed according to theorem C.17. ◻