

# Proof of the FAW Protocols

In this document, we demonstrate the correctness of the FAW protocol (§1), its variant for two-phase commit (2PC) (§2) and the RAINTX transaction protocol (§3). Also note two *errata* of the original paper (§4).

## 1 FAW Write Procedure

The form of the three-phase update procedure results from safety constraints. We first establish a *generic* write procedure that models *all possible* write paths. Then we enforce the safety constraints on the generic model and show that the FAW write procedure is necessary and optimal.

### 1.1 Generic Write Model

We consider a stripe of  $n$  chunks using a  $k$ -of- $n$  fault tolerance scheme. A write request modifies  $u$  chunks (including the  $m$  redundant chunks). A traditional logging method has to duplicate all  $u$  chunks, while our procedure allows  $\delta$  chunks to be updated in-place, saving  $\delta$  chunk writes.

After logging and updating  $u - \delta$  chunks, the remaining steps of the write procedure can be viewed as a repeat of a step that in-place updates some of the  $\delta$  chunks. We assume, in the  $i$ th step,  $\epsilon_i$  chunks are updated in-place in parallel, and before that step,  $d_i$  chunks have been updated in-place by previous steps ( $d_1 = 0$  denotes the first step). Therefore,  $d_{i+1} = d_i + \epsilon_i$ , for  $i \in \mathbb{N}^+$ . If  $d_i + \epsilon_i < \delta$ , that means not all  $\delta$  chunks can be updated in the  $i$ th step, so we have to do a next step; otherwise, the write procedure ends. Various write paths result from choosing a different  $\epsilon$  in every step. A write path consists of a series of steps  $(0, e_1), (d_2, e_2), \dots, (\delta, 0)$ : in the first step, we update  $e_1$  chunks while no chunks have been updated; in the second step, we update  $e_2$  chunks while  $d_2$  chunks (which equals to  $e_1$  in this step) have been updated; so on and so forth; finally, we have updated all  $\delta$  chunks and thus we should update no chunks in the end step. We visualize the steps in an  $\epsilon$ -dimensional space (see Figure 1). A point in the figure represents a step in a write path.

### 1.2 Optimization and Constrains

We formalize the problem of finding the optimal and safe write procedure of FAW as an optimization problem. In this optimization problem, the *objective* is to maximize  $\delta$ , while the *constraints* are to guarantee the safety of data.

**Constraints.** To express the constraints, we have to count the chunks of different versions, as summarized in Table 1. Note that a chunk is regarded as holding both new and old versions (“new/old”) when it is logged and updated; also, if it does not change at all in the write path (“not updated”), it is regarded as holding both versions. A chunk holds none (“—”) if it is being updated in-place.

As described in the original paper, we assume a distribution of  $f$  is a tuple  $(f_0, f_1, f_2)$ , where  $f_0$  failures destroy chunks holding both versions,  $f_1$  failures destroy old-version only chunks, and  $f_2$  destroy new-version only chunks. Then, the invariant is equivalent to satisfying one of the following two constraints.

*C1 – the number of old-version chunks is greater than or equal to  $k$ :* The number of old-version chunks is  $(n - u) + (u - \delta) + (\delta - d_i - \epsilon_i)$ , as Table 1 lists. Among all  $f$  failures,  $f_0 + f_1$  destroy old-version chunks. So, we must have  $(n - u) + (u - \delta) + (\delta - d_i - \epsilon_i) - (f_0 + f_1) \geq k$  to recover using *roll-back*.

*C2 – the number of new-version chunks is greater than or equal to  $k$ :* Similarly, summing up all new-version chunks as in Table 1 and considering  $f_0 + f_2$  failures that destroy new-version chunks, we must have  $(n - u) + (u - \delta) + d_i - (f_0 + f_2) \geq k$  to recover using *roll-forward*.

We construct the worst-case failure distribution by assuming an *adversary* who attempts to make

State	# of chunks	Version
not updated	$n - u$	new/old
logged and updated	$u - \delta$	new/old
updated in-place	$d_i$	new
being-updated in-place	$\epsilon_i$	—
not-yet-updated in-place	$\delta - d_i - \epsilon_i$	old

Table 1: Chunk states. A chunk holds the old or/and the new version of data.

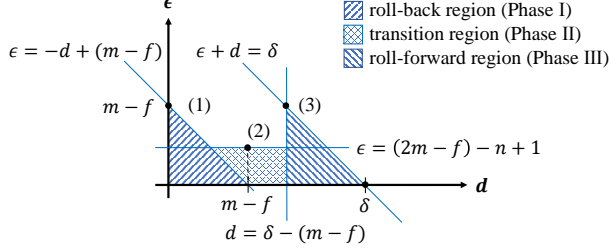


Figure 1: FAW steps in a  $d - \epsilon$  space. A point  $(d, \epsilon)$  means one step to update  $\epsilon$  chunks when  $d$  chunks have been updated. Constraints produce feasible regions (e.g., the left side of the line  $\epsilon = -d + (m - f)$  satisfies  $\epsilon \leq -d + (m - f)$ , which is Inequality 1). Points (1) – (3) represent three steps of a write path.

best use of the  $f$  failures to maximize damaged data. Our protocol and the adversary are *gaming* against each other.

The first priority of the adversary must be to increase  $f_0$  because one such failure can cause damage to *both* versions anyway. Consequently, if the number of chunks with both versions is larger than  $f$ , the adversary must set  $f_0 = f$ , and  $f_1 = f_2 = 0$ . As a result, C1 becomes:

$$\epsilon_i \leq -d_i + (m - f). \quad (1)$$

And C2 becomes:

$$d_i \geq \delta - (m - f). \quad (2)$$

Otherwise (the number of chunks with both versions is less than  $f$ ), the adversary would maximize  $f_0$  by setting it to the number of chunks with both versions, i.e.,  $f_0 = (n - u) + (u - \delta)$ . Since  $f_0$  is determined, C1 becomes  $f_1 \leq m - (n - \delta) - d_i - \epsilon_i$ . In addition, substituting  $f - f_0 - f_1$  for  $f_2$ , C2 becomes  $f_1 \geq \delta - (m - f) - d_i$ . The adversary must seek  $f_1$  within the scope  $(m - (n - \delta) - d_i - \epsilon_i, \delta - (m - f) - d_i)$  so that neither C1 nor C2 is satisfiable. Therefore, we have to ensure the scope is empty. As the value of  $f_1$  is discrete, that requires  $m - (n - \delta) - d_i - \epsilon_i \geq \delta - (m - f) - d_i - 1$ , which is:

$$\epsilon_i \leq (2m - f) - n + 1. \quad (3)$$

In Figure 1, we color all feasible  $(d_i, \epsilon_i)$  points that satisfy the constraints. The left triangle region always supports roll-back in *any distribution* of failures, because any point/step therein can survive the most adversarial case which leads to Inequality 1. Similarly, any point/step in the right triangle region is constantly recoverable by roll-forward. Between them may exist a *transition region* where we cannot

say for sure whether recovery will be via rolling forward or backward. One or the other will surely work, and the exact choice depends on the failure distribution (particularly, the choice of  $f_1$  as discussed for Inequality 3).

### 1.3 Procedure and Proof

Having the model and constraints, we now solve the optimization problem: find the maximum  $\delta$  so that there is a safe write procedure. A write procedure is safe as long as every step is recoverable, i.e., located in the feasible regions in Figure 1. As long as  $\epsilon = (2m - f) - n + 1 \geq 1$ , i.e.,  $n \leq 2m - f$ , all three regions connect. That means any step can find a feasible  $\epsilon$  until  $d = \delta$  which is the end of a write path. If so, there is no safety restriction on the value of  $\delta$ , which can be up to  $u$ .

Otherwise, when  $n > 2m - f$ , the transition region is empty. But if the roll-back region and the roll-forward region connect, a write path always exists. Thus, we need  $\delta - (m - f) \leq m - f$ , i.e.,  $\delta \leq 2(m - f)$ . All in all, the maximal  $\delta$  is as follows.

$$\delta_{\max} = \begin{cases} u, & \text{if } n \leq 2m - f; \\ \min\{2(m - f), u\}, & \text{otherwise.} \end{cases} \quad (4)$$

Next, we determine the optimal write procedure. As Figure 1 shows, when  $d = 0$ , we take the first step. To reduce the overall latency, we update as many chunks as possible in parallel. As a result, this step will reach from  $d = 0$  to  $d = m - f$ , bypassing the entire roll-back region. This step corresponds to Phase I of FAW. Hence, the number of chunks to update in this phase is  $s_1 = m - f$ . Then, we go through the transition region where  $\epsilon = (2m - f) - n + 1$ . Different from a triangle-shaped region, it is possible we need more than one step to get across the region. These steps make Phase II. Finally, we can go through the roll-forward region in a single step with  $s_3 = m - f$  in Phase III. In conclusion, FAW produces a safe write procedure whose every step is recoverable, with maximum  $\delta$  as well as minimal latency (steps).

## 2 FAW-2PC Protocol

## 3 RainTx Protocol

## 4 Errata

- Figure 5(b) of the original paper should be replaced with Figure 2. The original figure does

not match our description of Step ⑤ in the text, though it is also a correct protocol – it shows an optimization we planned but did not get enough time to implement before the paper submission. The optimization is to unlock primaries as early as possible without waiting for the coordinator’s notifications. Note that our evaluation of RAINTx follows Figure 2 without the optimization. Its Step ⑤ is similar to Step ⑤ of FaRM.

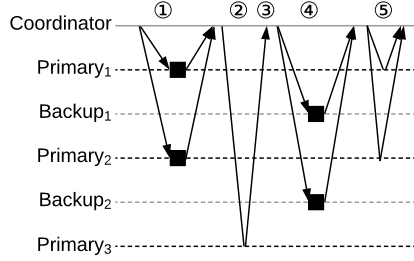


Figure 2: The commit protocol of RAINTx.

- In Figure 8(a) of the original paper, “FM-thr” and “RX-thr” should be swapped. The throughput of RAINTx (RX-thr) is higher than that of FaRM (FM-thr).