DBFT

二元 DBFT

本论文的二元一致性算法依赖于二元广播通信接口,BV-broadcast.在一个 BV-broadcast 实例中，每个进程 p_i 广播一个二元值以及接受（BV-delivers）一个二元值集合，该只读二元值集合 bin_values_i 存储在本地。bin_values_i 初始化为空集，随着每个新值到达而递增。BV-broadcast 具有以下四个性质：

1. 合约性.如果至少 t + 1 个非恶意节点调用 BV-broadcast 广播相同的值 v，那么值 v 最终会被加入每一个非恶意节点 p_i 的 bin_values_i
2. 可验证性。对于每一个非恶意节点 p_i，如果 $v \in bin\_values_i$，那么 v 由一个非恶意节点调用 BV-broadcast 进行广播
3. 统一性。如果一个非恶意节点将 v 加入 bin_values_i 集合，那么 v 最终会被每一个非恶意节点 j 加入 bin_values_j 集合
4. 可终止。每一个非恶意节点 j 的 bin_values_j 最终都是非空

从以上 4 条性质可以得到结论：每个非恶意节点的 p_i 的集合 bin_values_i 都会变得非空，且相等，包含所有非恶意节点广播的值（达成共识），且排除恶意节点发出来的值。

本地变量以及消息类型

每个进程维护以下局部变量。

- est_i:本地当前未确定值，由本地进程初始化
- r_i: 本地维护的 DBFT 某轮次值
- bin_values_i[1::]:二元值数组; bin_valuesi[r],表示第 r 轮由 BV-broadcast 生成的值
- b_i: 辅助二元值
- $values_i$: 辅助二元值集合

算法使用两种消息类型，EST 与 AUX。它们出现在每一轮当中，因此这两个消息类型总是与表示轮次的值 r 一起出现。

- EST[r]:进程 pi 在第 r 论调用 BV-broadcast 广播当前未确定值 est_i
- AUX[r]()：进程 pi 广播其当前值 bin_values_i[r]

安全的异步拜占庭二元一致性算法

算法分成 3 个阶段。

阶段 1：广播二元值，过滤拜占庭进程的值。算法 1 line 04.

对于每一个进程 pi，在 BV_broadcast()算法内，接受到 t+1 个进程发出的相同值 v 之后，进程 pi 重新广播该值。只有当进程 pi 接受到 2t+1 个进程发送的 v 值之后，进程 pi 才会调用函数 BV-delivers 确认 v 值，将 v 加入 bin_values_i[r]（算法 1， line 14）。接着，pi 等待直到 bin_values_i[r]不为空。

阶段 2：交换 est 值，收敛到一致性状态

第二阶段对应算法 line 06-07. 在这个阶段，pi 广播消息 AUX[r]()，内容是 $bin\_values_i[r]$(算法 1 line 06). 然后 pi 等待直到接收到满足以下两个条件的 $values_i$ 集合。

- 集合 $values_i$ 的值来自 n-t 个进程发送的消息 AUX[r]()
- $values_i$ 属于 $bin\_values_i[r]$。由于 BV-broadcast 过滤拜占庭进程的值，即使拜占庭节点伪造消息 AUX[r]()，$values_i$ 集合也只会包含非拜占庭节点提议的值。

因此，在任意的第 r 轮，在执行完代码 line 07 以后，$values_i$ 属于{0，1}集合，并且只包含由非拜占庭进程调用 BV-broadcast 广播的值（代码 line 04）

阶段 3：收敛到 r % 2，其中 r 是指第 r 轮 DBFT

这个阶段是单纯的本地计算。如算法 1 line 08 - line 12 所示。进程 pi 尝试在 b = r % 2 值达成共识，取决 $values_i$ 的内容。

- 如果 $values_i$ 只包含一个值，那么 pi 更新 est 值，并且作为达成一致性值的候选,调用 decide(v)完成。
- 如果 $values_i$ = {0，1}，pi 不能达成共识。因为两个值都是由非拜占庭节点提议，为了保证收敛，pi 选择 r % 2 的值。

Pi 如果达成共识，不能够立刻退出循环，而是继续停留在 bin_propose 内，在接下来的两轮协助其它进程完成共识。

```
operation bin_propose(v_i) is
(01)  est_i ← v_i; r_i ← 0;
(02)  while (true) do
(03)     r_i ← r_i + 1;
(04)     BV_broadcast EST[r_i](est_i); // add to bin_values[r_i] upon BV_delivery
(05)     wait_until (bin_values_i[r_i] ≠ ∅);
(06)     broadcast AUX[r_i](bin_values_i[r_i]);
(07)     wait_until (messages AUX[r_i](b_val_p(1)), ..., AUX[r_i](b_val_p(n−t)) have been received
                        from (n − t) different processes p(x), 1 ≤ x ≤ n − t, and their contents are
                        such that ∃ a non-empty set values_i where (i) values_i = ∪_{1≤x≤n−t} b_val_p(x)
                        and (ii) values_i ⊆ bin_values_i[r_i]);
(08)     b_i ← r_i mod 2;
(09)     if (values_i = {v}) // values_i is a singleton whose element is v
(10)       then est_i ← v; if (v = b_i) then decide(v) if not yet done end if;
(11)       else  est_i ← b_i
(12)     end if;
(13)  end while.

(14)  when B-VAL[r](v) is BV-delivered by BV_broadcast[r] do
         bin_values_i[r] ← bin_values_i[r] ∪ {v};
```

Fig. 1. A safe algorithm for the binary Byzantine consensus in $\mathcal{BAMP}_{n,t}[t < n/3]$

下面是安全且活性的二元拜占庭一致性算法，Psync. 这个算法不需要使用签名或者随机化，同时具有如下特性：

- 时间最优，能够在 O(t)消息延迟内终止
- 如果所有的非拜占庭节点提议相同的值，程序能够在 O(1)时间内终止。
- 不需要恢复操作，并且不需要等待协调者的信息·

额外的本地变量与消息类型. 除了维护 est_i, r_i, bin_values_i [r]以及 $values_i$ 之外，进程 pi 还需要维护以下局部变量。

- $timer_i$，本地计数器；$timeout_i$，超时限制。它们一起用来保证假设 synch

- $coord_i$，索引当前轮次的（弱）协调者
- aux_i，辅助集合，用来存储当前弱协调者想要提议的值

第 r 轮的弱协调者利用消息类型 COORD_VALUE[r]广播提议值。下面是扩展版本的算法。

- 在 line New 1，pi 等待 bin_values_i 值不为空，然后设置本地计时器。计时器用在 line M-05 用于同步弱协调者发送的信息。计时器每过一轮都会递增
- Line New 4，等待(n-t)个不同进程发送 AUX[r]()消息，然后设置本地计数器。（这行代码主要是从程序设计角度找到一个设定定时器的恰当时机）
- Line New2,New3, M-06,以及 New5 实现了允许让弱协调者发送提议值

```
operation bin_propose(v_i) is
(01)  est_i ← v_i; r_i ← 0;
      timeout_i ← 0;
(02)  while (true) do
(03)     r_i ← r_i + 1;
(Opt1)   if (est_i = −1) then est_i ← 1; // "fast-path" for round 1, only used in the reduction in Sect. IV
(04)        else BV_broadcast EST[r_i](est_i);
         end if;
(New1)   wait_until (bin_values_i[r_i] ≠ ∅);
         timeout_i ← timeout_i + 1; set timer_i to timeout_i;
(New2)   coord_i ← ((r_i − 1) mod n) + 1;
         if (i = coord_i) then
           {w} = bin_values_i[r_i]; // w is the first value to enter bin_values_i[r_i]
           broadcast COORD_VALUE[r_i](w)
         end if;
(M-05)   wait_until ((bin_values_i[r_i] ≠ ∅) ∧ (timer_i expired));
(New3)   if ((COORD_VALUE[r_i](w) received from p_coord_i) ∧ (w ∈ bin_values_i[r_i]))
           then aux_i ← {w}
           else aux_i ← bin_values_i[r_i]
         end if;
(M-06)   broadcast AUX[r_i](aux_i);
(New4)   wait_until (a message AUX[r_i]() has been received from (n − t) different processes);
         set timer_i to timeout_i;
(M-07)   wait_until ((messages AUX[r_i](b_val_p(1)), ..., AUX[r_i](b_val_p(n−t)) have been received
                      from (n − t) different processes p(x), 1 ≤ x ≤ n − t, and their contents are
                      such that ∃ a non-empty set values_i where (i) values_i = ∪_{1≤x≤n−t} b_val_p(x)
                      and (ii) values_i ⊆ bin_values_i[r_i]) ∧ (timer_i expired));
(New5)   if (when considering the whole set of the messages AUX[r_i]() received, several sets
             values1_i, values2_i, ... satisfy the previous wait predicate) ∧ (one of them is aux_i)
                      then values_i ← aux_i end if; // values_i is either defined here or at line M07
(08)     b_i ← r_i mod 2;
(09)     if (values_i = {v}) // values_i is a singleton whose element is v
(10)        then est_i ← v; if (v = b_i) then decide(v) if not yet done end if;
(11)        else est_i ← b_i
(12)     end if;
(Opt2)   if (decided in round r_i) then // the following are termination conditions
           wait until (bin_values_i[r_i] = {0, 1}) // only go to the next round when necessary
         else if (decided in round r_i − 2) then halt end if; // everyone has decided by now
         end if;
(13)  end while.
```

弱协调者 p_k 广播消息 COORD_VALUE[r_i](w)，其中 w 是第一个进入 bin_values 数组的值(line New2).如果 p_k 是非拜占庭节点，非拜占庭节点的计时器足够且消息延迟由上限，那么所有的非拜占庭节点都会接收到这个消息。可以推导，程序将会在 r+1 或者（r+2）达成一致，取决于当前 p_k 的值于 ri % 2 的值是否相等。

归约为二元 DBFT 的多值 DBFT

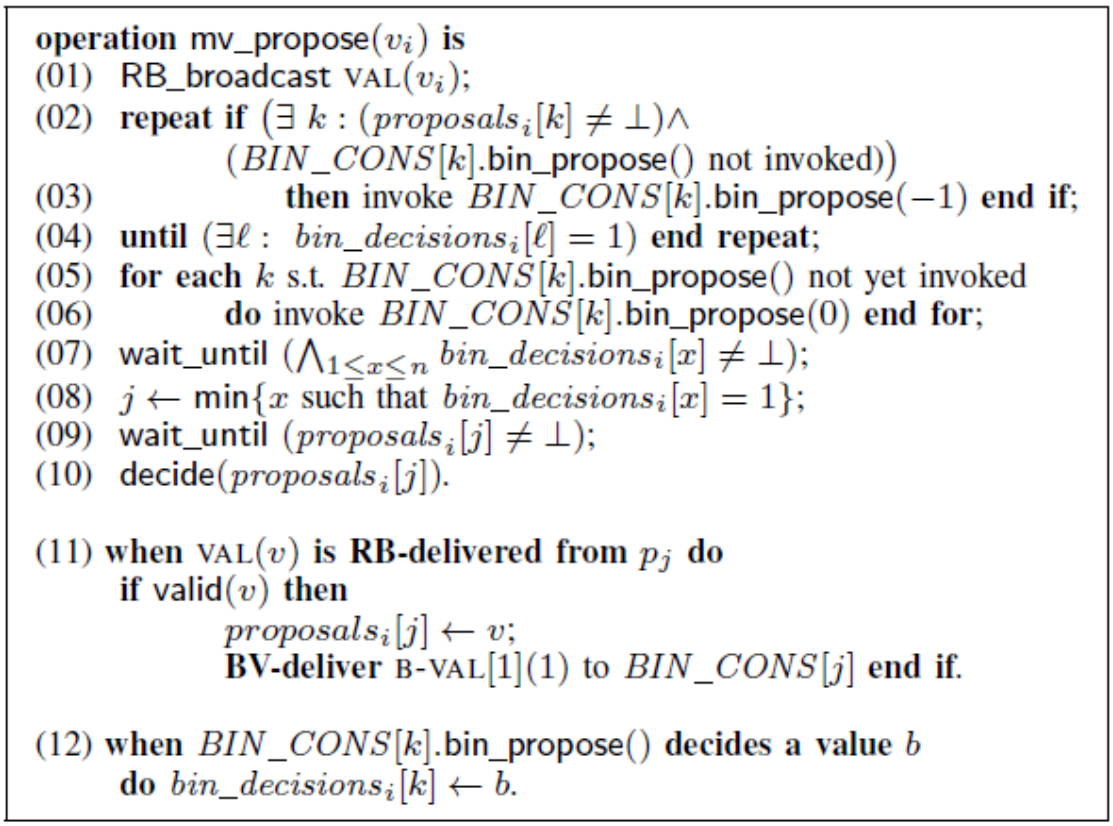
所有进程在全局二元拜占庭对象 BIN_CONS[1..n]协作，在进程 pk 提议的值上达成共识。

第一阶段：进程 pi 利用 RB_broadcast 广播值 v（line 1，11）。如果 pi 调用 RB_delivers 确认一个由进程 pj 广播的合法值 v，那么进程 pi 将这个值存储到 $proposal_i[j]$。同时，设置 BIN_CONS[j]为 1（bin_values_i 为 1）

第二阶段：这个阶段进程 pi 开始参与一系列二元一致性实例（line 02-04）。进程 pi，传递参数-1，触发一个二元一致性实例 k（由进程 pk 广播的值，pi 接收）。-1 的意思是，跳过 BV-broadcast，直接发送值为 1 的 AUX 消息，以便在一轮次中达成共识。一个优化是，进程 pi 利用 RB-deliver 投递提议值，也可以同时利用 BV-deliver 投递值 1。如果某个实例 *l* 达到共识，即可退出该循环。

第三阶段：进程 pi 开始参与其它二元实例（line 05-06）。Pi 已经确认某个 *l* 实例在值 1 上达成共识，针对它没有参与的其它实例触发 bin_propose(0)。

第四阶段：pi 确定一个共识值，挑选下标最小的二元对象 j 作为本次达成共识的值，$proposals_i[j]$，完成归约。



operation mv_propose(v_i) **is**
(01) RB_broadcast VAL(v_i);
(02) **repeat if** $(\exists\, k : (proposals_i[k] \neq \perp) \wedge$
($BIN\_CONS[k]$.bin_propose() not invoked))
(03) **then** invoke $BIN\_CONS[k]$.bin_propose(-1) **end if**;
(04) **until** $(\exists \ell :\ bin\_decisions_i[\ell] = 1)$ **end repeat**;
(05) **for each** k s.t. $BIN\_CONS[k]$.bin_propose() not yet invoked
(06) **do** invoke $BIN\_CONS[k]$.bin_propose(0) **end for**;
(07) wait_until $(\bigwedge_{1 \leq x \leq n} bin\_decisions_i[x] \neq \perp)$;
(08) $j \leftarrow \min\{x$ such that $bin\_decisions_i[x] = 1\}$;
(09) wait_until $(proposals_i[j] \neq \perp)$;
(10) decide($proposals_i[j]$).

(11) **when** VAL(v) **is RB-delivered from** p_j **do**
if valid(v) **then**
$proposals_i[j] \leftarrow v$;
BV-deliver B-VAL[1](1) to $BIN\_CONS[j]$ **end if**.

(12) **when** $BIN\_CONS[k]$.bin_propose() **decides a value** b
do $bin\_decisions_i[k] \leftarrow b$.