Errata for: RAMBO: A Robust, Reconfigurable Atomic Memory Service for Dynamic Networks

Seth Gilbert
NUS, Singapore
seth.gilbert@comp.nus.edu.sg

Nancy A. Lynch
MIT, Cambridge, USA
lynch@theory.lcs.mit.edu

Alexander A. Shvartsman
U. of Connecticut,
Storrs, CT, USA
aas@cse.uconn.edu

Overview

We have discovered several small typos in Figure 6 of "RAMBO: A Robust, Reconfigurable Atomic Memory Service for Dynamic Networks" (*Distributed Computing*, Volume 23, Number 4, pages 225–272). In this errata sheet, we correct the typos, and explain the relevant changes.

Specifically, due to typos in the preparation of the manuscript, there was a bug in the "handshake" mechanism by which the processes synchronized their phases. Fixing these typos required fixing the indentation on one line, and inserting two omitted lines of pseudocode. The basic behavior of the algorithm remains unchanged, and the fixes have led to no changes in the analysis of the algorithm.

Details

This note refers specifically to mistakes in Figure 6 on p. 240 of the original manuscript. We have reproduced below a corrected version of Figure 6. There are three corrections to note:

- Line 36 was omitted in the original manuscript. After $pnum\text{-}local$ is updated in line 35, the operational phase number $op.pnum$ should be updated.
- Line 62 was incorrectly indented in the original manuscript, implying that it is executed only when $op.type = read$. In fact, line 62 should be executed in all cases.
- Line 63 was omitted in the original manuscript. After $pnum\text{-}local$ is updated in line 62, the operational phase number $op.pnum$ should be updated.

In addition, these changes led to changes in the line numbering, which are reflected in the revised text.

Discussion

The RAMBO algorithm performs read and write operations in two phases: a query phase and a propagation phase. In each phase, certain information is sent to a "quorum," and the phase completes when the initiator has received a response from every node in some "quorum." (Note that since RAMBO is reconfigurable, the exact definition of a sufficient quorum depends on the set of active configurations and is implemented as a fixed point; however these details are not relevant for the current discussion.)

The initiator of a phase must ensure that the responses it has received are in fact part of the current phase. Notably, the initiator should not accidently count a response from an earlier phase. (Such a "late response" from an earlier phase does not indicate that the sender has received a message sent by the initiatoras part of the phase.)

In RAMBO, this phase structure is enforced by a simple "handshake" protocol. When a process initiates a phase, it chooses a unique phase number—larger than all previous phase numbers—and attaches that phase number to its messages that are associated with that phase. When a response includes a phase number at least as large the phase number associated with that operation (Figure 6, line 30), then the initiator can be certain that the response is part of that phase. (See Section 5.2 for a more detailed discussion of this phase number mechanism.)

The mistakes in Figure 6 were associated with this handshake mechanism. Notably, in two cases (lines 36 and 63), the initiator incremented the phase number correctly (i.e., $pnum\text{-}local$), but omitted to update the *operation*, i.e., to add the updated phase number to the operation record op. In the other case (line 62), the phase number was only incremented in one case of an **if** clause, rather than in all cases.

Thus, any implementation of the RAMBO algorithm must be careful to increment the phase number correctly whenever a phase is begun.

Transitions:

```
1  Input read_i
2  Effect:
3      if status ∉ {idle, failed} then
4          pnum-local ← pnum-local + 1
5          op.pnum ← pnum-local
6          op.type ← read
7          op.phase ← query
8          op.cmap ← cmap
9          op.acc ← ∅
10
11 Input write(v)_i
12 Effect:
13     if status ∉ {idle, failed} then
14         pnum-local ← pnum-local + 1
15         op.pnum ← pnum-local
16         op.type ← write
17         op.phase ← query
18         op.cmap ← cmap
19         op.acc ← ∅
20         op.value ← v
21
22 Input recv(⟨world', v, t, cm, snder-phase, rcver-phase⟩)_{j,i}
23 Effect:
24     if status ∉ {idle, failed} then
25         status ← active
26         world ← world ∪ world'
27         if t > tag then (value, tag) ← (v, t)
28         cmap ← update(cmap, cm)
29         pnum-vector(j) ← max(pnum-vector(j), snder-phase)
30         if op.phase ∈ {query, prop} and rcver-phase ≥ op.pnum then
31             op.cmap ← extend(op.cmap, cm)
32             if op.cmap ∈ Usable then
33                 op.acc ← op.acc ∪ {j}
34             else
35                 pnum-local ← pnum-local + 1
36                 op.pnum ← pnum-local
37                 op.acc ← ∅
38                 op.cmap ← cmap
39         else if gc.phase ∈ {query, prop} and rcver-phase ≥ gc.pnum
40             gc.acc ← gc.acc ∪ {j}
41
42 Output send(⟨world', v, t, cm, snder-phase, rcver-phase⟩)_{i,j}
43 Precondition:
44     status = active
45     j ∈ world
46     ⟨world', v, t, cm, snder-phase, rcver-phase⟩ =
47       ⟨world, value, tag, cmap, pnum-local, pnum-vector(j)⟩
48 Effect: None
49 Internal query-fix_i
50 Precondition:
51     status = active
52     op.type ∈ {read, write}
53     op.phase = query
54     ∀k ∈ ℕ, c ∈ C : (op.cmap(k) = c)
55         ⇒ (∃R ∈ read-quorums(c) : R ⊆ op.acc)
56 Effect:
57     if op.type = read then
58         op.value ← value
59     else
60         value ← op.value
61         tag ← ⟨tag.seq + 1, i⟩
62     pnum-local ← pnum-local + 1
63     op.pnum ← pnum-local
64     op.phase ← prop
65     op.cmap ← cmap
66     op.acc ← ∅
67
68 Internal prop-fix_i
69 Precondition:
70     status = active
71     op.type ∈ {read, write}
72     op.phase = prop
73     ∀k ∈ ℕ, c ∈ C : (op.cmap(k) = c)
74         ⇒ (∃W ∈ write-quorums(c) : W ⊆ op.acc)
75 Effect:
76     op.phase = done
77
78 Output read-ack(v)_i
79 Precondition:
80     status = active
81     op.type = read
82     op.phase = done
83     v = op.value
84 Effect:
85     op.phase ← idle
86
87 Output write-ack_i
88 Precondition:
89     status = active
90     op.type = write
91     op.phase = done
92 Effect:
93     op.phase ← idle
94
95
96
```

Figure 6: $Reader\text{-}Writer_i$: Read/write transitions