Algorithm 1 client operations at client c

1: **function** GET(k, rc)
2: $s \leftarrow$ `PARTITION`(k)
3: $(v, ct, ot) \leftarrow$ `S.GET` $-$ `REQUEST`(k, ct_c, ot_c, rc)
4: $ct_c \leftarrow max(ct, ct_c)$
5: $ot_c \leftarrow ot$
6: **return** v
7:
8: **function** PUT(k, v, wc)
9: $s \leftarrow$ `PARTITION`(k)
10: $(ct, ot) \leftarrow$ `S.PUT` $-$ `REQUEST`(k, v, ot_c, wc)
11: $ct_c \leftarrow max(ct, ct_c)$
12: $ot_c \leftarrow ot$
13: **return** ok

Algorithm 2 server operations at server c

```
1:  function GET-REQUEST(k, ct, ot, rc)
2:      ct_s ← max(ct, ct_s)
3:      if rc = local then
4:          v ← store_s[k]
5:          return < v, ct_s, aot_s >
6:      if rc = majroity then
7:          v ← snapshot_s[k] at cot_s
8:          return < v, ct_s, cot_s >
9:
10:     function PUT-REQUEST(k, ct, ot, wc)
11:         ct_s ← max(ct, ct_s)
12:         ct_s ← TICK()
13:         aot_s ← ct_s
14:         store_s[k] ← v
15:         oplog_s ← oplog ∘ < k, v, aots >
16:         while runtime < wc.waittime do
17:             if count(server.aot ≥ aot_s) ≥ wc.w then
18:                 return < ct_s, aot_s >
19:         return error
20:
21:     function REPLICATE
22:         if IsSECONDARY(s) then
23:             < oplog, cot, ct > ← SYNCSRC(s).PULL − OPLOG(ct_S, aot_S)
24:             ct_s ← max(ct, ct_s)
25:             for (k, v, ot) ∈ oplog do
26:                 store_s[k] ← v
27:                 aot_s ← ot
28:             oplog_S ← oplog_S ∘ oplog
29:             send < s, aot_s > to primary(s)
30:             update snapshot_s to cot
31:
32:     function PULL-OPLOG(ct, aot)
33:         ct_s ← max(ct, ct_s)
34:         oplog ← oplog entries after aot in oplog_s
35:         return <oplog, cot, ct_s>
```

Algorithm 3 clock management at server c

```
1:  function TICK
2:      if ct_s.sec ≥ clock_s then
3:          return < ct_s.sec, ct_s.counter + 1 >
4:      else
5:          return < clock_s, 0 >
```