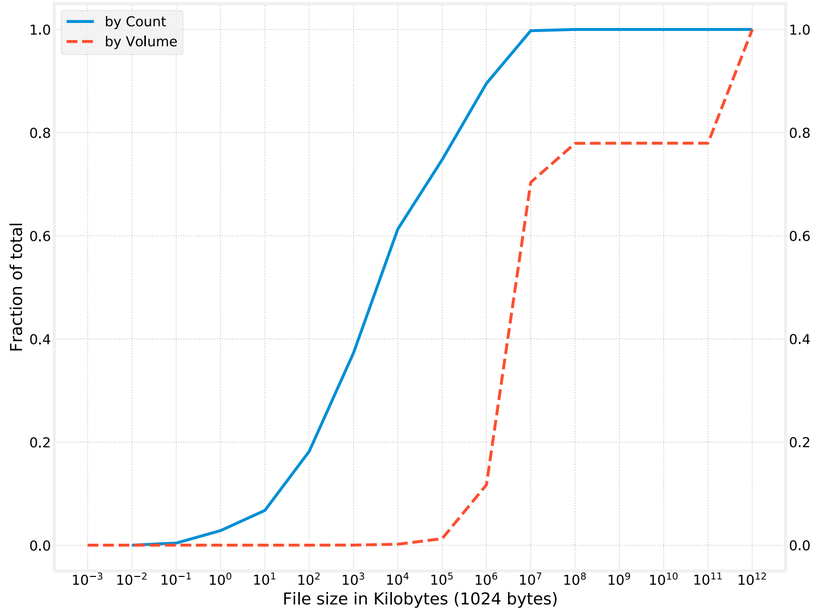
Logic and Lattices for Distributed Programming

Neil Conway, William R. Marczak, Peter Alvaro, Joseph M. Hellerstein and David Maier

Presented by Dev
Nov 16 2018


ROLLING THROUGH THE BAY


by Count
by Volume
Fraction of total
1.0
0.8
0.6
0.4
0.2
0.0
10^{-3} 10^{-2} 10^{-1} 10^{0} 10^{1} 10^{2} 10^{3} 10^{4} 10^{5} 10^{6} 10^{7} 10^{8} 10^{9} 10^{10} 10^{11} 10^{12}
File size in Kilobytes (1024 bytes)

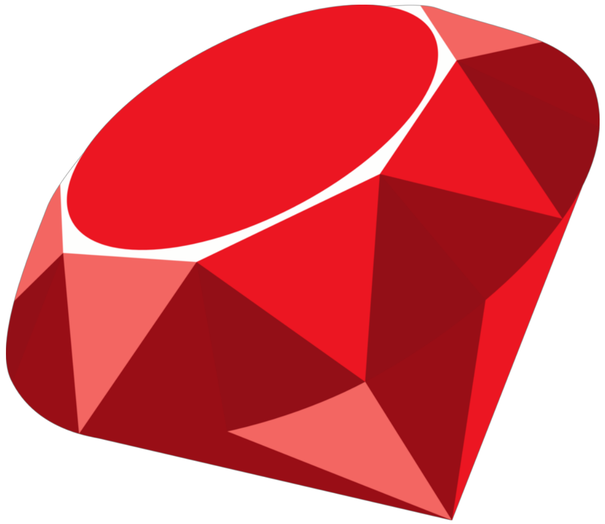
<~ bloom

Fact

Eval

Do

Timestep



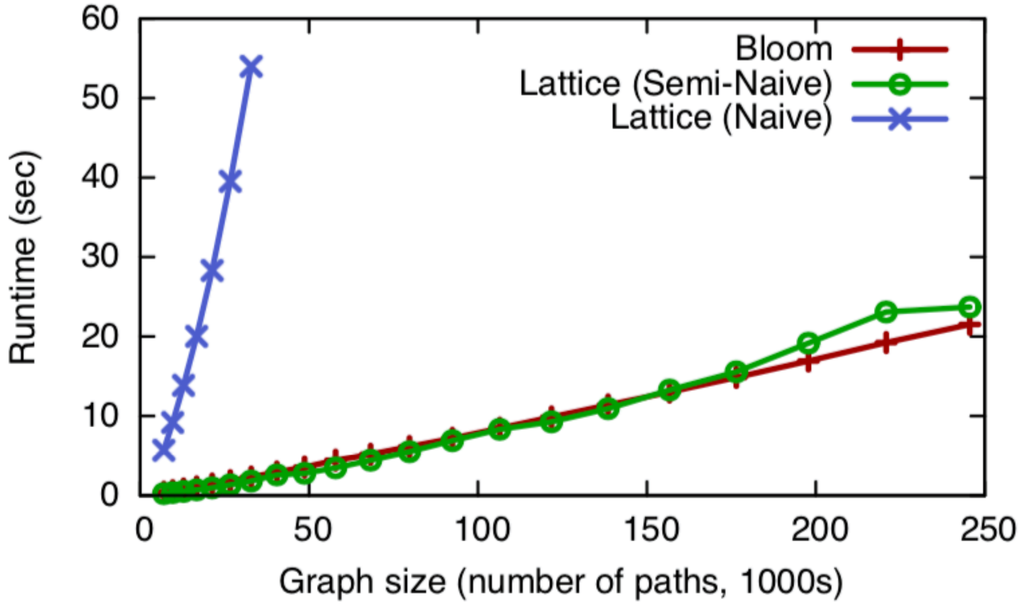
```
1  QUORUM_SIZE = 5
2  RESULT_ADDR = "example.org"

4  class QuorumVote
5    include Bud

7    state do
8      channel :vote_chn, [:@addr, :voter_id]
9      channel :result_chn, [:@addr]
10     table   :votes, [:voter_id]
11     scratch :cnt, [] => [:cnt]
12   end

14   bloom do
15     votes      <= vote_chn {|v| [v.voter_id]}
16     cnt        <= votes.group(nil, count(:voter_id))
17     result_chn <~ cnt {|c| [RESULT_ADDR] if c >= QUORUM_SIZE}
18   end
19 end
```

Figure 2: A non-monotonic Bloom program that waits for a quorum of votes to be received.

```
1  QUORUM_SIZE = 5
2  RESULT_ADDR = "example.org"

4  class QuorumVoteL
5    include Bud

7    state do
8      channel :vote_chn, [:@addr, :voter_id]
9      channel :result_chn, [:@addr]
10     lset    :votes
11     lmax    :cnt
12     lbool   :quorum_done
13   end

15   bloom do
16     votes       <= vote_chn {|v| v.voter_id}
17     cnt         <= votes.size
18     quorum_done <= cnt.gt_eq(QUORUM_SIZE)
19     result_chn  <~ quorum_done.when_true { [RESULT_ADDR] }
20   end
21 end
```

Figure 3: A monotonic Bloom^L program that waits for a quorum of votes to be received.

$$\vee \ max() \ min() \ \cup$$


Bloom
Lattice (Semi-Naive)
Lattice (Naive)
Runtime (sec)
Graph size (number of paths, 1000s)
0
10
20
30
40
50
60
0
50
100
150
200
250

```
12  module MonotoneReplica
13    include CartProtocol

15    state { lmap :sessions }

17    bloom do
18      sessions <= action_msg do |m|
19        c = LCart.new({m.op_id => [ACTION, m.item, m.cnt]})
20        { m.session => c }
21      end
22      sessions <= checkout_msg do |m|
23        c = LCart.new({m.op_id => [CHECKOUT, m.lbound, m.addr]})
24        { m.session => c }
25      end

27      response_msg <~ sessions do |session, cart|
28        cart.is_complete.when_true {
29          [cart.checkout_addr, session, cart.summary]
30        }
31      end
32    end
33  end
```

Figure 10: Cart server replica in Bloom^L that supports a monotonic (coordination-free) checkout operation.