Prof. T. Roscoe, Prof. R. Wattenhofer

Computer Systems

Assignment 10

1 Approximate Agreement

Basic

1.1 Modified Synchronous Algorithm

We modify Algorithm 21.5 (for synchronous approximate agreement) so that now a node computes the median of its received values each iteration rather than removing the f lowest and f highest values and computing the mean of the minimum and maximum remaining values.

Algorithm 1 Synchronous Approximate Agreement

Code for node v with input x.
$I = \lceil \log_2(\texttt{max\_range}/\varepsilon) \rceil$.
$x_0 = x$.
for i in $1 \ldots I$ **do**
  Send x_{i-1} to all nodes.
  Add every received value to multiset R_i.
  $x_i = \text{median}(R_i)$.
end for
Output x_I.

a) Does correct-range validity still hold? If so, briefly explain why. If not, give a counterexample execution where it fails.

b) Does ε-Agreement still hold? If so, briefly explain why. If not, give a counterexample execution where it fails.

1.2 From Approximate Agreement to Byzantine Agreement

We want to design an **asynchronous** byzantine agreement algorithm (where nodes' inputs are bits) that relies on Algorithm 21.26 from the lecture nodes. Recall that Algorithm 21.26 achieves asynchronous approximate agreement even when $f < n/3$ of the nodes are byzantine.

Nodes proceed as follows: every node joins Algorithm 21.26 with its input bit as initial value. Once a node obtains a value x from Algorithm 21.26, it outputs 0 if $x < 0.5$ and 1 otherwise.

a) Does all-same validity hold?

b) What about agreement?

c) Assume an ideal shared coin that enables the nodes to agree on a uniformly distributed random value in $(0, 1)$. Once $f + 1$ nodes query this shared coin, the random value is sampled and all nodes learn it eventually.

How can we use this coin to achieve agreement except with probability 10^{-2024}?

Advanced

1.3 Unbounded Input Space: Quick Fix

The approximate agreement algorithms presented in the lecture rely on a publicly known `max_range` that the input space should satisfy. This allows us to (overestimate) a sufficient number of iterations. To drop this assumption **in the synchronous model** (Algorithm 21.5), we will build a mechanism that enables each node to (over)estimate a `max_range` based on the nodes' inputs. Hence, if X denotes the multiset of correct inputs, we will ask each node to estimate $\max X - \min X$.

a) How would obtaining agreement on $\max X - \min X$ help?

b) Describe in your own words why correct nodes cannot agree on $\max X - \min X$.

Instead, each node will try to *estimate* the initial range X. This can be done using one round of communication preceding the for loop of Algorithm 21.5.

c) Write an algorithm that uses one round of communication and allows each correct node v to obtain an estimation $\texttt{max\_range}_v \geq \max X - \min X$.

d) How can the algorithm from Task **c)** be used to replace the hard-coded value I in Algorithm 21.5? Keep in mind that nodes do not obtain the same value $\texttt{max\_range}_v$.

e) Can you provide an upper bound on the number of iterations in your solution in Task **d)**?

2 Consistency and Logical Clocks

Quiz

2.1 Different Consistencies

Prove or disprove the following statements:

a) Neither sequential consistency nor quiescent consistency imply linearizability.

b) If a system has sequential consistency **and** quiescent consistency, it is linearizable.

Basic

2.2 Measure of Concurrency from Vector Clocks

You are given two nodes that each have a vector logical clock that additionally logs the clock state upon receiving a message (see Algorithm 2).

Algorithm 2 Vector clocks with logging

(Code for node u)
Initialize $c_u[v] := 0$ for all other nodes v.
Upon local operation: Increment current local time $c_u[u] := c_u[u] + 1$.
Upon send operation: Increment $c_u[u] := c_u[u] + 1$ and include the whole vector c_u as d in message.
Upon receive operation: Extract vector d from message and update $c_u[v] := \max(d[v], c_u[v])$ for all entries v. Increment $c_u[u] := c_u[u] + 1$. Save the vector c_u to the log file of node u.

Assume that exactly one message gets send from one to the other node. Given the logs and current vector states of both nodes, write a short program that calculates the measure of concurrency as defined in the script (Definition 22.33). You can use your favorite programming language. The example solution will be in Python.

Advanced

Generalize your program to any number of messages exchanged between the nodes.