CS 5860
Tue Nov 22, 2011

Consensus Protocols

Logical Properties

All decided values are input values.

All e:E(Decide). Exists e':E(Input).
e' < e & Decide(e) = Input(e')

We can see that P2 will imply P1, so we take P2 as part of the requirements.

Event Classes

If X is an event class, then E(X) are the events in that class. Note E(X) effectively partitions all events E into E(X) and E-E(X), its complement.

Every event in E(X) has a value of some type T which is denoted X(e). In the case of E(Input) the value is the typed input, and for E(Decide) the value is the one decided.

Further Requirements for Consensus

The key safety property of consensus is that all decisions agree.

Any two decisions have the same value.

This is called agreement.

All e1,e2: E(Decide). Decide(e1) = Decide(e2).

Specific Approaches to Consensus

Many consensus protocols proceed in rounds, voting on values, trying to reach agreement. We have synthesized two families of consensus protocols, the 2/3 Protocol and the Paxos Protocol families.

We structure specifications around events during the voting process, defining E(Vote) whose values are pairs <n,v>, a ballot number, n, and a value, v.

Properties of Voting

Suppose a group G of n processes, Pi, decide by voting. If each Pi collects all n votes into a list L, and applies some deterministic function f(L), such as majority value or maximum value, etc., then consensus is trivial in one step, and the value is known at each process in the first round – possibly at very different times.

The problem is much harder because of possible failures.

Fault Tolerance

Replication is used to ensure system availability in the presence of faults. Suppose that we assume that up to f processes in a group G of n might fail, then how do the processes reach consensus?

The TwoThirds method of consensus is to take n = 3f +1 and collect only 2f+1 votes on each round, assuming that f processes might have failed.

Example for f = 1, n = 4

Here is a sample of voting in the case T = {0,1}.

0	0	1	1	inputs

0 _11	_011	001_	00_1	collected votes
1	1	0	0	next vote

00_1	001_	0_11	_011
0	0	1	1

where f is majority voting, first vote is input

Specifying the 2/3 Method

We can specify the fault tolerant 2/3 method by introducing further event classes.

E(Vote), E(Collect), E(Decide)

E(Vote): the initial vote is the <0,input value>, subsequent votes are <n,f(L)>

E(Collect): collect 2f+1 values from G into list L

E(Decide): decide v if all collected values are v

The Hard Bits

The small example shows what can go wrong with 2/3. It can waffle forever between 0 and 1, thus never decide.

Clearly if there is are decide events, the values agree and that unique value is an input.

Can we say anything about eventually deciding, e.g. liveness?

Liveness

If f processes eventually fail, then our design will work because if f have all failed by round r, then at round r+1, all alive processes will see the same 2f+1 values in the list L, and thus they will all vote for v' = f(L), so in round r+2 the values will be unanimous which will trigger a decide event.

Example for f = 1, n = 4

Here is a sample of voting in the case T = {0,1}.

0	0	1	1	inputs
0 01_	001_	001_	_011	collected votes
0	0	0	1	next vote

000_	00_1	0_01	_001
0	0	0	0

where f is majority voting, first vote is input, round numbers omitted.

Safety Example

We can see in the f = 1 example that once a process Pi receives 2/3 unanimous values, say 0, it is not possible for another process to over turn the majority decision.

Indeed this is a general property of a 2/3 majority, the remaining 1/3 cannot overturn it even if they band together on every vote.

Safety Continued

In the general case when voting is not by majority but using f(L) and the type of values is discrete, we know that if any process Pi sees unanimous value v in L, then any other process Pj seeing a unanimous value v' will see the same value, i.e. v = v' because the two lists, Li and Lj at round r must share a value, that is they intersect.