Computer Science 425
Distributed Systems

CS 425 / ECE 428

Consensus

What is Consensus?

- N processes
- Each process p has
  - input variable xp : initially either 0 or 1
  - output variable yp : initially b (b=undecided) – can be changed only once
- Consensus problem: design a protocol so that either
  1. all non-faulty processes set their output variables to 0
  2. Or non-faulty all processes set their output variables to 1
  3. There is at least one initial state that leads to each outcomes 1 and 2 above
  4. (There might be other conditions too, but we'll consider the above weaker version of the problem).

Let's Solve Consensus!

- **Processes fail only by *crash-stop*ping**
- **Synchronous system: bounds on**
  - **Message delays**
  - **Max time for each process step**

  e.g., multiprocessor (common clock across processors)
- **Asynchronous system: no such bounds!**

  e.g., The Internet! The Web!

Consensus in Synchronous Systems

For a system with at most *f* processes crashing, the algorithm proceeds in *f+1* **rounds** (with timeout), using basic multicast (B-multicast).

- A round is a numbered period of time where processes know its start and end (kinda like an hour, only smaller)

- $Values^r_i$: the set of proposed values known to process P_i at the beginning of round *r*.
- Initially $Values^0_i = \{\}$; $Values^1_i = \{v_i = \text{xp}\}$

```
for round r = 1 to f+1 do
        multicast (Values^r_i) // e.g., B-multicast
        Values^{r+1}_i ← Values^r_i
        for each V_j received
            Values^{r+1}_i = Values^{r+1}_i ∪ V_j
                end
end
yp=d_i = minimum(Values^{f+2}_i)
```

Why does the Algorithm Work?

- Proof by contradiction.
- Assume that two non-faulty processes differ in their final set of values.
- Suppose p_i and p_j are these processes.
- Assume that p_i possesses a value v that p_j does not possess.
  - → In the last ($f+1$) round, some third process, p_k, sent v to p_i, but crashed before sending v to p_j.
  - → In the f-th round, p_k possessed the value v while p_j did not.
  - → In the f-th round, some fourth process, p_{k2}, sent v to p_k, but crashed before sending v to p_j.
  - → Proceeding in this way, we infer at least one crash in each of the preceding rounds.
  - → But are $f+1$ rounds ==> $f+1$ failures. Yet we assumed f crashes ==> contradiction.

Consensus in an Asynchronous System

- **Messages have arbitrary delay, processes arbitrarily slow**
- **Impossible to achieve!**
  - **even a single failed process is enough to avoid the system from reaching agreement!**
  - **Key observation: a slow process indistinguishable from a crashed process**
- **Impossibility Applies to *any* protocol that claims to solve consensus!**
- **Proved in a now-famous result by Fischer, Lynch and Patterson, 1983 (FLP)**
  - **Stopped many distributed system designers dead in their tracks**
  - **A lot of claims of "perfect reliability" vanished overnight**

Summary

- **Consensus Problem**
  - agreement in distributed systems
  - Solution exists in synchronous system model (e.g., supercomputer)
  - Impossible to solve in an asynchronous system (e.g., Internet, Web)
    - » Key idea: with only one process failure and arbitrarily slow processes, there are always sequences of events for the system to decide any which way. Regardless of which consensus algorithm is running underneath.
  - FLP impossibility result