Consensus with Failure Detector

Antonio Bucchiarone

Fondazione Bruno Kessler, Trento – Italy
bucchiarone@fbk.eu

27 November 2019

The Consensus Problem

- Each non faulty process *has to propose a value* to *all* the other processes.
- They must *agree on a common value* among the proposed ones.
- Agreement in Asynchronous Distributed Systems
- One of the most important problems is determining whether a participating process is *still correct* (but slow) or *has crashed.*
- *Fischer, Lynch, and Paterson (FLP)* proved that it is indeed impossible to predict failures perfectly in a fully asynchronous system without putting further assumptions in place.
- It is impossible to solve problems like consensus and atomic broadcast deterministically even for a single process failure.

Failure Detectors

- *Chandra and Toueg* address this problem in their seminal papers, introducing ***unreliable failure detectors***.

- They work around the FLP limitation by allowing processes to suspect that others have failed, usually based on *liveness criteria*, thus effectively bringing them back into synchrony.

- They introduce two main properties of such **failure detectors**: *completeness* and *accuracy*.

Completeness

- *Completeness* guarantees that **all failed processes** are **eventually permanently suspected** by a correct process.

- Completeness is sub-divided into:

  - ***strong completeness***, under which all failed processes are eventually suspected by all correct processes, and

  - ***weak completeness***, under which all failed processes are eventually suspected by *some* correct process.

Accuracy

- *Accuracy* ensures that ***a correct process is not suspected by any correct process***.
- ***Strong accuracy*** ensures that *all correct processes are never suspected by any correct process.*
- ***Weak accuracy*** ensures that *at least one correct process is never suspected by any correct process*.
- By further relaxing these accuracy properties, two additional versions arise:
  - ***eventual strong accuracy***, under which strong accuracy is guaranteed after some time in the future, and
  - ***eventual weak accuracy***, under which weak accuracy is guaranteed after a future time.

Classes of Failure Detectors

FONDAZIONE BRUNO KESSLER

Detector	Completeness	Accuracy
Perfect	Strong	Strong
Eventually Perfect	Strong	Eventually Strong
Strong	Strong	Weak
Eventually Strong	Strong	Eventually Weak
Weak	Weak	Weak
Eventually Weak	Weak	Eventually Weak
	Weak	Strong
	Weak	Eventually Strong

Two important applications of *failure detectors* are:

(1) leader election and

(2) consensus in asynchronous distributed systems.

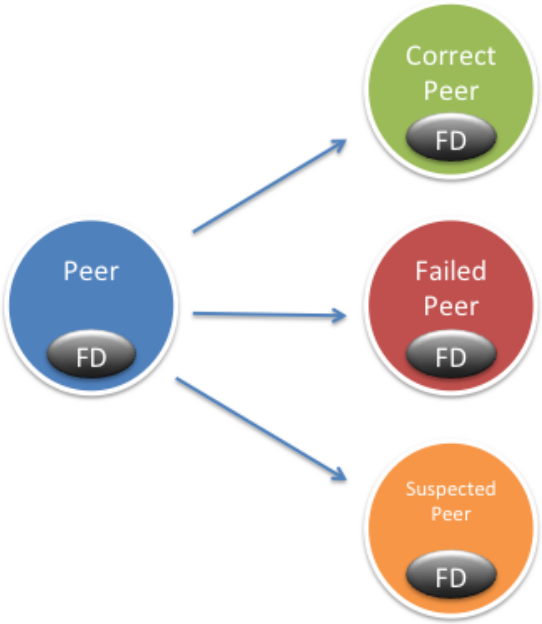
Implementation Steps

- Then *simulation* of **Strong Completeness and (Eventually) (Weak) Accuracy Failure Detector** can be done as follows:
  - Initially there is a list of ActorRef (processes) ,i.e. *{P1, P2, P3, ..., Pn}*
  - Each ActorRef *Pi* will be centrally assigned (in FDCentralInfo) the "crashed" tick *Ci* which means the process crash at the specified tick, i.e. this information is available initially to every processes as a map of *{P1 = C1, P2 = C2, ..., Pn = Cn}*
  - Each other ActorRef *Pj* will "realize" that the process *Pi* is crashed at a certain later tick *Cij >= Ci*, i.e. this information (in FailureDetector) is different from process to process *{P1 = {C12, ..., C1n}, P2 = {C21, ..., C2n}, ... Pn = {Cn1, ... Cn(n-1)}}*
  - And thus, at a certain tick *T*, the list of suspects for a process *Pi* can be constructed as *{Pj}* where *Cij >= Ci*

Implementation Steps

```
// Initiate the global clock
GlobalClock.getClock().currentTick();

ActorSystem system = ActorSystem.create("MyActorSystem");

int N = 30;
List<ActorRef> ps = new ArrayList<ActorRef>();
for (int i = 1; i <= N; i++) {
    ps.add(system.actorOf(getProps(), "ESCA-" + String.valueOf(i)));
    //ps.add(system.actorOf(getProps(), "SCA-" + String.valueOf(i)));
    //ps.add(system.actorOf(getProps(), "RBA-" + String.valueOf(i)));
}

FDCentralInfo.getFDCentralInfo().initCrashedTicks(ps);
//ReliableBroadcastFDActor
for (ActorRef p : ps) {
    p.tell(new EventuallyStrongConsensusActor.StartMessage(FDCentralInfo.getFDCentralInfo().getCrashedTicks()), null);
    //p.tell(new ReliableBroadcastFDActor.StartMessage(FDCentralInfo.getFDCentralInfo().getCrashedTicks()), null);
}

try {
    Thread.sleep(1000);
} catch (InterruptedException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

//ps.get(0).tell(new ReliableBroadcastFDActor.BroadcastMessage(88), null);

for (ActorRef p : ps) {
    p.tell(new EventuallyStrongConsensusActor.ProposeMessage(88), null);
}
```

Failure Detectors



A *global clock* is provided through the singleton *class GlobalClock*.

```
public class GlobalClock {
    private long startTime = 0;
    private final long delta = 200;
    private GlobalClock() {
        startTime = System.currentTimeMillis();
    }

    private static GlobalClock instance = null;
    public static GlobalClock getClock() {
        if (instance == null) {
            instance = new GlobalClock();
        }
        return instance;
    }
    public int currentTick() {
        return (int) ((System.currentTimeMillis() - startTime) / delta);
    }
}
```

Start Message

```
public class EventuallyStrongConsensusActor extends UntypedActor {

    LoggingAdapter log = Logging.getLogger(getContext().system(), this);

    public static Props props() {
        return Props.create(EventuallyStrongConsensusActor.class);
    }

    public static class StartMessage {
        private final Map<ActorRef, Integer> crashedTicks;

        public StartMessage(Map<ActorRef, Integer> crashedTicks) {
            this.crashedTicks = Collections.unmodifiableMap(crashedTicks);
        }
    }

    public static class ProposeMessage {
        private final int value;

        public ProposeMessage(int value) {
            this.value = value;
        }

        public int getValue() {
            return value;
        }
    }
```

OnReceive

```
public void onReceive(Object message) throws Exception {
    if (FDCentralInfo.getFDCentralInfo().isCrashed(me)) return;
    if (message instanceof StartMessage) {
        StartMessage sm = (StartMessage) message;
        processes.addAll(sm.crashedTicks.keySet());
        fd = new FailureDetector(sm.crashedTicks, me);
    } else if (message instanceof ProposeMessage) {
        ProposeMessage am = (ProposeMessage) message;
        round = 0;
        estimate = am.getValue();
        decided = false;
        stop = false;
       log.info("Receive propose {} at round {}", estimate, round);
        onPropose();
    } else if (message instanceof Phase1Message) {
        Phase1Message p1m = (Phase1Message) message;
        //log.info("Receive P1M {} at round {}", p1m.estimate, round);
        onPropose2(p1m.round, p1m.estimate, p1m.sender);
    } else if (message instanceof Phase2Message) {
        Phase2Message p2m = (Phase2Message) message;
        //log.info("Receive P2M {} at round {}", p2m.aux, round);
        onPropose2Reply(p2m.round, p2m.aux, p2m.sender);
    } else if (message instanceof DecideMessage) {
        DecideMessage dm = (DecideMessage) message;
        //log.info("Receive DM {} at round {}", dm.value, round);
        onDecide(dm);
    } else if (message.equals("tick")) {
        onTimeout();
    } else {
        unhandled(message);
    }
}
```

Propose – Send - Decide

```
protected void onPropose() {
    if (FDCentralInfo.getFDCentralInfo().isCrashed(me)) return;
    if (stop) return;
    //log.info("Propose {} at round {}", estimate, round);
    int cID = FDCentralInfo.getFDCentralInfo().getCoordinatorID(round);
    round++;
    if(FDCentralInfo.getFDCentralInfo().isCoordinator(me, cID)) {
        log.info(" As Coordinator - Send P1M {} at round {}", estimate, round);
        Phase1Message p1m = new Phase1Message(round, estimate, me);
        broadcastMessage(p1m);
    }
    if (fd.getSuspects().contains(FDCentralInfo.getFDCentralInfo().getProcessByID(cID))) {
        onPropose2(round, estimate, me);
    }
}
```

```
protected void broadcastMessage(Object msg) {
    if (FDCentralInfo.getFDCentralInfo().isCrashed(me)) return;
    for (ActorRef p : processes) {
        if (!p.equals(me)) {
            p.tell(msg, me);
        }
    }
}
```

```
protected void onDecide(DecideMessage dm) {
    if (FDCentralInfo.getFDCentralInfo().isCrashed(me)) return;
    if (!decided) {
        estimate = dm.value;
        DecideMessage dm2 = new DecideMessage(estimate);
        broadcastMessage(dm2);
        decided = true;
        log.info("Decided {} at round {}", estimate, round);
    }
}
```