How to circumvent these impossibility results?

• change the model:
  • synchronous model
  • partially synchronous models
• direct approach:
  • Dwork, Lynch and Stockmeyer Consensus in the presence of partial synchrony J. ACM 88
  • Dolev, Dwork and Stockmeyer On the minimal synchronism needed for distributed consensus. J. ACM 87
Failure Detector Chandra & Toueg

• PODC 91 - J. ACM 96
• Distributed oracles that give hints on the failure pattern (e.g. set of suspected processes)
• A FD is basically defined by:
  – a completeness property: actual detection of failure
  – an accuracy property: restrict the mistake that a FD can make
• Failure pattern $F(t)$: set of crashed processes at time $t$ 
  \[ F : T \to 2^\Pi \]

• A failure detector history $H(t,p)$ is the set of processes that are suspected to be crashed 
  \[ H : T \times \Pi \to 2^\Pi \]
• A failure detector $D$ provides (possibly incorrect) information about the failure pattern $F$. Formally, failure detector $D$ is a function that maps each failure pattern $F$ to a set of failure detector histories $D(F)$.

• $D: F \rightarrow D(F)$
• P : perfect (each crashed process will be suspected, no correct process is suspected)
• S : strong (each crashed process will be suspected, at least one correct process is never suspected)
• P : perfect (each crashed process will be suspected, no correct process is suspected)
• S : strong (each crashed process will be suspected, at least one correct process is never suspected)
  – here directly if a FD is perfect it is strong:
• P is « stronger » than S
Comparison:

• $D \geq D'$ if there exists a (distributed) algorithm where processes query $D$ and output $D'$

• (trivially $P \geq S$ but not $S \geq P$)
• $\langle\rangle P :$ eventually $P$
• $\langle\rangle S :$ eventually $S$
Hints about failure

• How many « information » about failures ?
  – Sufficient : with the oracle the problem can be solved
  – Necessary: with less information the problem cannot be solved
Leader $\Omega$

- There exists a correct process $q$ and a time $t$ after which such that every correct process trusts $q$

- The output of the failure detector is one process

- Eventually the output of all processes is the same correct process
<>S and Ω are equivalent

- Ω ≥<>S
- <>S ≥Ω

- each process p : V[p] : array of size n
- (code of p)
- each k steps :
  - Q=query the failure detector <>S
  - for all processes q : if q in Q then V[p][q] is incremented
  - compute V[p][q] = max V[x][q]
  - output of Ω = q such that V[p][q] is the smallest element of V[p][*]
weakest failure detector

- Problem P
- Define a failure detector D
  - D can solve P
  - if D’ solves P then D’ ≥ D
    » D is a weakest failure detector to solve P
    » A weakest failure detector encapsulates the exact amount of information on failure necessary and sufficient to solve a problem
weakest failure detector?

• Given problem $P$ and problem $P'$, $\text{wk}(P)$ ($\text{wk}(P')$) the weakest failure detector for $P$ (for $P'$)

• if $\text{wk}(P) < \text{wk}(P')$ ($\text{wk}(P) \leq \text{wk}(P')$ and not $\text{wk}(P') \leq \text{wk}(P)$) then $P'$ is harder than $P$ ( $P'$ needs more information about failure than $P$ )

• chase the weakest failure detector for problems (...many papers... )
weakest failure detector?

• Every problem that is solvable with a failure detector has a weakest failure detector [JT08]

• Problem: (1) given a run, it is possible to determine whether the problem requirements are met in the run, and (2) an algorithm is considered to solve the problem if every run of the algorithm meets the problem requirements.
Weakest: Chandra, Hadzilacos&Toueg

- PODC 92 - J. ACM 96

Weakest failure detector to solve consensus in message passing with a majority of correct processes: leader
If D can be used to solve consensus then $D \geq \Omega$

- Processes construct a directed acyclic graph (DAG) that represents a “sampling” of failure detector values in the run and some temporal relationships between the values sampled.
- This DAG can be used to simulate runs of Consensus with D
- By considering several initial configurations of Consensus, we obtain a forest of simulated runs of Consensus
- It is possible to extract the identity of a process p that is correct in the run (the same correct process for all processes)
Weakest (shared memory)

- $\Omega$: consensus [HL 94]
- Anti-$\Omega$: set-agreement [Zielinski10]
  - Each query to the anti-$\Omega$ returns a process id; the specification ensures that there is a correct process whose id is returned only finitely many times.
  - anti-$\Omega$ is equivalent to a vector of $n$-1 processes such that one coordinate is an $\Omega$
Implementation

- implementing a FD in (as weak as possible) partially synchronous system
implementing $\Omega$ in shared memory

• There exists $k$, there exists a correct process $c$ such that for all processes $p$ there is at least one step of $c$ each $k$ steps of $p$. 
Relation between wait free and t-resilence

The Borowsky-Gafni (BG) simulation algorithm

• a set of $t+1$ asynchronous sequential processes may wait-free simulate an algorithm for $n$ processes t-resilient ($n>t$)

• the reverse is also true (and trivial)
Application

• If we know that it is impossible to achieve wait free consensus for two processes

• Then we deduce by BG that it is impossible to achieve 1-resilient consensus for n processes
Application

- Proof by contradiction

- Assume there exists an algorithm $A$ that achieves consensus for $n$ processes 1-resilient.

- $P_0$ and $P_1$ simulate WF the algo $A$ for $n$ processes 1-resilient with their initial value. When one of the simulated process decides, $P_0$ and $P_1$ adopt this decision.

Colorless tasks
k-set agreement

• n processes can solve k-set agreement (k-1)-resilient

• processes 1 to k write their input value in shared memory and decide their input value

• other processes read the shared memory until they read some value and decide this value
• **Thm**: $n$ processes cannot solve $k$-set agreement $k$-resilient

• (known result): it is impossible to solve wait free set agreement ( $k+1$ processes decide at most $k$ values)

• **proof by contradiction** assume $n$ processes we can solve $k$-set agreement $k$-resilient

• with $k+1$ processes we can simulate WF $n$ processes $k$-resilient that executes the $k$-set agreement algorithm (BG simulation)

• Contradicting the known result.
BG simulation

n + 1 processes may wait free simulate
m + 1 processes n-resilient (m \geq n)
BG : main ideas

- Let $S_0, S_1, \ldots, S_n$ be the simulators

- Let $p_0, p_1, p_2, \ldots, p_m$ the simulated processes. This processes have to execute a code $C_i$ and used shared memory $M$
S0, S1,..Sm simulate steps of pi by consensus
Shared Memory

MEM, an $n$ by $m$ array of registers.

For each $i,j$, $MEM[i][j]$ contains a pair ($val$, $steps$):
$MEM[i][j].val$, initially the initial value of $j$’s register in $A$
$MEM[i][j].steps \in \mathbb{N}$, initially 0

For each $i$ $MEM[i][\_\_]$ can be written to by $i$

CONSENSUS, an $m$ by infinite array of consensus objects

Process Local Variables

input $\in I_i$, initially the input of real process $i$

state, steps arrays of size $m$, initially arbitrary

decided, a boolean, initially arbitrary

Thread Local Variables

$k', i', j, j', steps \in \mathbb{N}$, initially arbitrary

val, v, w, o, variables, initially arbitrary

mymem, an array of $n$ variables, initially arbitrary
Assume first that all simulators are correct

upon PROPOSE(input)
    decided ← false
Parallel for all $j = 1..m$
    $w \leftarrow CONSENSUS_{(n-1)}[j][0](input)$
    $state[j] \leftarrow init\_state_j(w)$
    $steps[j] \leftarrow 1$
repeat forever
    if $nextop_j(state[j]) = (WRITE, v)$ then
        $MEM[i][j] \leftarrow (v, steps[j])$
        $state[j] \leftarrow trans_j(state[j], DONE)$
    else if $nextop_j(state[j]) = (READ, j')$ then
        for $i' = 1..n$
            $mymem[i'] \leftarrow MEM[i'][j']$
        $(val, steps) \leftarrow VMAX(mymem)$
        $(w, k') \leftarrow CONSENSUS_{(n-1)}[j][steps[j]](val, steps)$
        $state[j] \leftarrow trans_j(state[j], w)$
    else if $nextop_j(state[j]) = (OUTPUT, o)$ then
        if $\neg decided$ output $o$; decided $\leftarrow true$
        $steps[j] \leftarrow steps[j] + 1$
• Parallel for all $j=1,\ldots,m$. The simulator takes one step for each $j$

• For a wait free simulation replaces CONSENSUS by safe agreement and when the simulator has to execute the consensus it executes the three first instructions of safe agreement (the simulator may block only one safe agreement)
and message passing?

• message passing:
  • send/receive messages with asynchronous communication (each message sent by a process is eventually received by a correct process)
  • Process may crash (stop the execution)
  • process p is correct if it doesn’t crash (makes an infinity of steps)
  • t- resiliency at most t processes may crash
Reliable Broadcast

- primitives $\text{Rbcast Rdeliver}$
  - **Agreement**: if $p$ correct $\text{Rdeliver m}$ then every correct process $\text{Rdeliver m}$
  - **Validity**: If $p$ correct $\text{Rbcast m}$ then $p$ $\text{Rdeliver m}$
  - **Integrity**: If $p$ $\text{Rdeliver m}$ then there is a process $q$ that has $\text{Rbcast m}$
RBcast

Algorithm for process $p$:

To execute $\text{Rbcast}(m)$
   send $(m)$ to $p$

$\text{Rdeliver}(m)$ occurs when

upon receive$(m)$ do
   if has not previously executed $\text{Rdeliver}(m)$
      then
         send $(m)$ to all
         $\text{Rdeliver}(m)$
ABcast

• primitives \textbf{ABcast ABdeliver}:

  • RBcast properties:

  • \textit{Total order}: If \(p\) and \(q\) \textbf{ABdeliver} \(m\) and \(m'\) then if \(p\) \textbf{ABdelivers} \(m\) before \(m'\) then \(q\) \textbf{ABdelivers} \(m\) before \(m'\)

ABCcast is « universal »: (very informal) (active replication)

  • state machine replication:
    • any sequential state machine \(A\)
    • \(t+1\) processes simulate \(A\)
    • each request is made by atomic broadcast
    • then we get a \(t\)-resilient implementation of \(A\)
Atomic Broadcast and Consensus

Consensus in message passing: decision algorithm such that:

- **termination**: all correct processes decide

- **validity**: if p decides v then v is an initial value of some process

- **agreement**: if p and q decide, the decide the same value
Algorithm for process $p$:

Initialization:

\[ \text{RDelivered} := \emptyset \]
\[ \text{ADelivered} := \emptyset \]

To execute Abcast($m$)

\[ \text{Rbcast}(m) \]

Adeliver(_$) \text{ occurs when} \]

upon Rdeliver($m$) do

\[ \text{RDelivered} := \text{RDelivered} \cup \{m\} \]

do forever

\[ \text{AUndelivered} := \text{RDelivered} - \text{ADelivered} \]
\[ \text{if } \text{AUndelivered} \neq \emptyset \text{ then} \]

\[ k := k + 1 \]

propose($k, \text{AUndelivered}$)

wait for decide($k, \text{msgSet}$)

batch($k) := \text{msgSet} - \text{ADelivered}$

A-deliver all messages in batch($k$) in some deterministic order

\[ \text{ADelivered} := \text{ADelivered} \cup \text{batch}(k) \]
Conclusion

Atomic broadcast and consensus are equivalent in message passing with crash failure.

Consensus is « universal » in message passing with crash failure.
Message Passing versus Shared memory

- simulating share memory: with a *majority* of correct processes atomic registers may be implemented in asynchronous message passing models.

- *(Message-Passing and shared memory models are « equivalent » -with a majority of correct processes)*
Simulating shared registers in message passing

- simulation of a single-writer single-reader register
  - assume we have a majority of correct processes
For the writer
to write(\(v\))
\[ seq := seq + 1 \]
send \((W, v, seq)\) to all
wait until receiving \([n/2] + 1\) messages \((ACK, seq)\)

For the reader
to read()
send(\(R\)) to all
wait until receiving \([n/2] + 1\) messages \((V, v, s)\) such that \(s > seq\)
return \(val\)
such that \((V, val, S)\) has been received
and \(S\) is the max of the sequence number of received \(V\) messages

For all processes
when \((W, v, s)\) is received
if \(s > seq\) then
\[ val := v; seq := s \]
send \((ACK, s)\)
when(\(R\)) is received
send \((V, val, seq)\) to \(p_r\)
A majority of correct processes is needed

• partition argument:

• if \( n \leq 2t \) then we can partition the set of processes in two set \( S_1 \) and \( S_2 \) such that \( |S_1| \geq t \) and \( |S_2| \geq t \).

• Run A_1: all processes in \( S_1 \) are correct and all processes in \( S_2 \) are initially dead, \( p_0 \) invokes a \texttt{write(1)} at some time \( t_0 \) the write terminates

• Run A_2: all processes in \( S_2 \) are correct and all processes in \( S_1 \) are initially dead, \( p_1 \) \( S_2 \) invokes a \texttt{read()} at time \( t_0 + 1 \) the read terminates at time \( t_1 \)

• Run B: « merge » of \( A_1 \) and \( A_2 \) but no process crash. \texttt{write(1)} terminates before the \texttt{read()} and the read return 0

contradiction