In search for lost universality

Journey to the Center of Distributed Computing
Roadmap

The lost universality
- Consensus is necessary but impossible

The quest for universality
- Consensus is sufficient

Circumventing universality
Universality

Algorithmi

Turing
The Lost Universality

The infinitely big

The infinitely small
**Counter: Specification**

A *counter* has two operations `inc()` and `read();` it maintains an integer `x init to 0

- **read():**
  - return(x)

- **inc():**
  - `x := x + 1;`
  - return(ok)
Counter: Algorithm

The processes share an array of registers Reg[1,..,N]

inc():
  Reg[i].write(Reg[i].read() +1);
  return(ok)

read():
  sum := 0;
  for j = 1 to N do
    sum := sum + Reg[j].read();
  return(sum)
Counter*:: Specification

Counter* has, in addition, operation $dec()$

$dec()$:  
if $x > 0$ then $x := x - 1$; return(ok)  
else return(no)

Can we implement Counter* asynchronously?
2-Consensus with Counter*

- Registers R0 and R1 and Counter* C - initialized to 1

- Process pI:
  - propose(vI)
  - RI.write(vI)
  - res := C.dec()
  - if(res = ok) then
    ✓ return(vI)
  - else return(R{1-I}.read())
Impossibility [FLP85, LA87]

- **Theorem:** no asynchronous algorithm implements consensus among two processes using registers

- **Corollary:** no asynchronous algorithm implements Counter* among two processes using registers

Sperner’s Lemma
Roadmap

- The lost universality
  - Consensus is necessary but impossible

- The quest for universality
  - Consensus is sufficient

- Circumventing universality
Roadmap

• The lost universality
  Consensus is necessary but impossible

• The quest for universality
  Consensus is sufficient

• Circumventing universality
The **consensus number** of an object is the maximum number of processes than can solve consensus with it.
Roadmap

- The lost universality
  - Consensus is necessary but impossible

- The quest for universality
  - Consensus is sufficient

- Circumventing universality
Consensus Universality [L78]

- **Theorem**: every object can be implemented with consensus
Eventual Synchrony

The weakest failure detector question

Indulgent algorithms: Paxos, PBFT

The next 700 BFT protocols
Hardware
RDMA

- Remote shared / protected memory

- Consensus with 2f+1 and f+1 (vs 3f+1 and 2f+1) and 2 steps (vs 4 steps) – PODC 2018/2019

- $\mu$: SMR in 1$\mu$s / 1ms – OSDI 2020
NVRAM

Persistent objects with durable linearizability / detectable recovery

Tight bound: 1 pfence per operation (SPAA 2019)

MCAS with 2 pfences and k+1 CASes per k-Cas (DISC 2020)
Distributed Payment

X000 implementations

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshi@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As
P vs NP

Asynchronous vs Synchronous

Is payment an asynchronous problem?

« To understand a distributed computing problem: bring it to shared memory » T. Lannister
Message Passing

Shared Memory
Message Passing

send

p1

p2

receive

p3
Shared Memory

Atomic Registers
Message Passing ⇔ Shared Memory
Modulo Quorums
Is payment an asynchronous problem?

Payment Object

Atomicity

Wait-freedom
Payment Object (PO): Specification

Pay(a, b, x): transfer amount x from a to b if a > x (return ok; else return no)

Important. Only the owner of a invokes Pay(a,*,*)

- Can PO be implemented asynchronously?
- What is the consensus number of PO?
A *snapshot* has operations *update()* and *scan()*; it maintains an array $x$ of size $N$.

*scan()*:
- return($x$)

*update($i,v$)*:
- $x[i] := v$;
- return(ok)
The Payment Object: Algorithm

Every process stores the sequence of its outgoing payments in its snapshot location.

To **pay**, the process scans, computes its current balance: if bigger than the transfer, updates and returns ok, otherwise returns no.

To **read**, scan and return the current balance.
PO can be implemented asynchronously

Consensus number of PO is 1

Consensus number of PO(k) is k
Faster and Simpler Payment Systems (AT2)

AT2_S (PODC 2019)

AT2_D (DNS 2020)

AT2_R (DISC 2019)
Journey to the Center of DC

- Bitcoin
- Blockchain
- Proof of work
- Smart contracts
- Ethereum
- Atomicity
- Wait-freedom
- Snapshot
- Consensus
- Quorums
- Secure Broadcast
Programming languages to the rescue

How to write a better universal Internet machine?

How to write better universal programs?