COMP28112 – Lecture 16
Replication

Reasons for Replication (1)

- Reliability
  - **Redundancy** is a key technique to increase availability. If one server crashes, then there is a replica that can still be used. Thus, failures are tolerated by the use of redundant components. Examples:
    - There should always be at least two different routes between any two routers in the internet.
    - In the Domain Name System, every name table is replicated in at least two different servers.
    - A database may be replicated in several servers to ensure that the data remains accessible after the failure of any single server.

Trade-off: there is some cost associated with the maintenance of different replicas.

Redundancy

- Wouldn't you prefer to use an **expensive** airplane with triple-redundancy for all hardware resources [1] as opposed to a **cheap** airplane with a single-resource (no redundancy)?
- **Availability** of a replicated service over a period of time:
  - 1 – Probability(all replicas have failed)
  - Probability(all replicas have failed) =
    Probability(replica 1 has failed) ×
    Probability(replica 2 has failed) ×
    Probability (replica 3 has failed).
  - Probabilities are between 0 (=no chance/impossible) and 1 (=certainty).

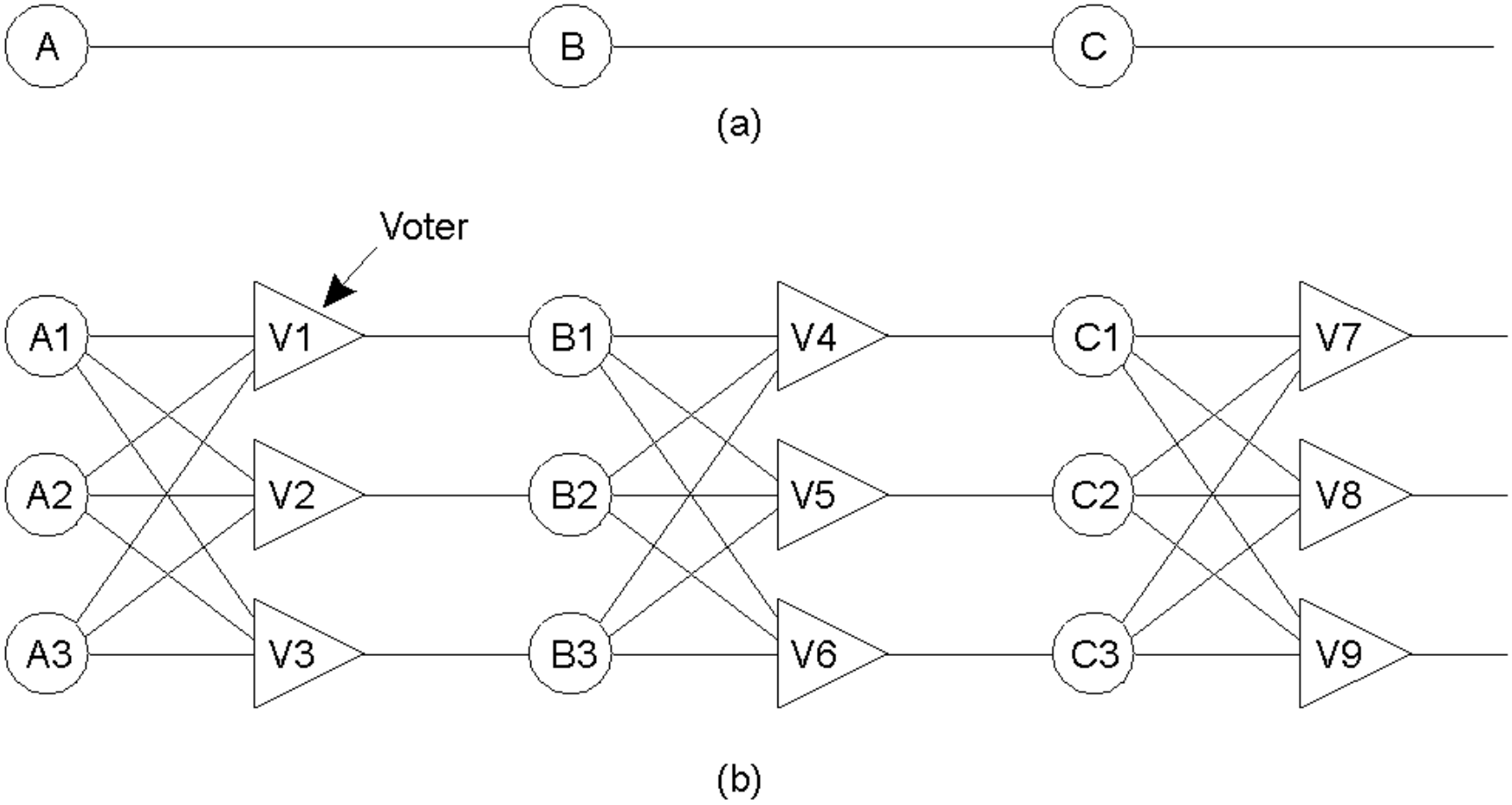
[1] "Triple-triple redundant 777 primary flight computer". Proceedings of the 1996 IEEE Aerospace Applications Conference.

Example

- If the probability of a system failing is 0.3 (30%), the probability of two copies of the system failing at the same time is 0.3x0.3=0.09, the probability of three copies all failing at the same time is 0.3^3=0.027, the probability of ten copies all failing at the same time is 0.3^{10}=0.000005905. This means that with ten copies we can have availability of 99.9994095%.
- To compute probability (p) that a single component has failed (i.e., the component is not available), use mean time between failures (f) and mean time to repair a failure (t): $p = t/(f+t)$
- The degree of redundancy has to strike a balance between benefits and the additional cost needed to implement it!

Failure Masking by Redundancy

(see Tanenbaum, fig 8.2, p.327)



Triple modular redundancy (a form of n-modular redundancy)

Reasons for Replication (2)

- **Performance**
  - By placing a copy of the data in the proximity of the process using them, the time to access the data decreases. This is also useful to achieve scalability. Examples:
    - A server may need to handle an increasing number of requests (e.g., google.com, ebay.com). Improve performance by replicating the server and subsequently dividing the work.
    - **Caching**: Web browsers may store locally a copy of a previously fetched web page to avoid the latency of fetching resources from the originating server. (cf. with processor architectures and cache memory)

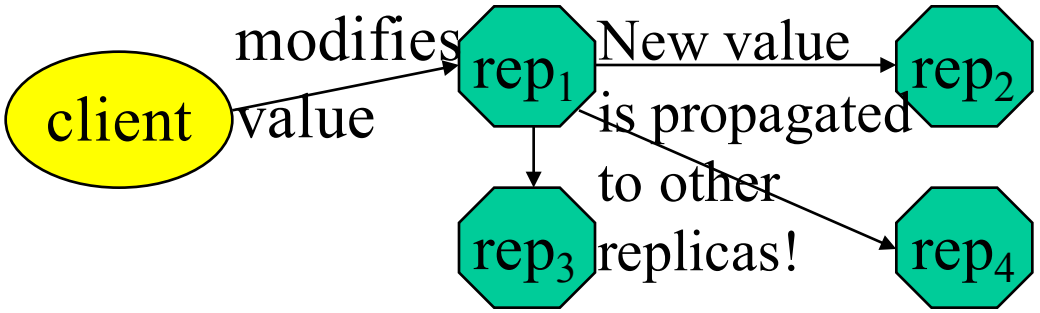
How many replicas?

(or, how do we trade cost with availability?)

- **Capacity Planning**: the process of determining the necessary capacity to meet a certain level of demand (a concept that extends beyond distributed computing)
- Example problem:
  - Given the cost of a customer waiting in the queue;
  - Given the cost of running a server;
  - Given the expected number of requests;
  - How many servers shall we provide to guarantee a certain response time if the number of requests does not exceed a certain threshold?
- Problems like this may be solved with mathematical techniques. **Queuing theory**, which refers to the mathematical study of queues (see Gross & Harris, Fundamentals of queueing theory) may be useful.
- ***Not all situations can be modelled analytically using mathematical approaches. Simulation may be useful (cf lab exercise 3)***
- Check the lecture on lab exercise 3…

The price to be paid…

- Besides the cost (in terms of money) to maintain replicas, what else could be against replication?
- Consistency problems:
  - If a copy is modified, this copy becomes different from the rest. Consequently, modifications have to be carried out on all copies to ensure consistency. When and how those modifications need to be carried out determines the price of replication. Example (4 replicas):



The cure may be worse than the disease!!!

The price to be paid… (cont)

- Keeping multiple copies consistent may itself be subject to serious scalability problems!
- In the example before, when an update occurs it needs to be propagated to all other replicas. No other processes should read the same value from the other replicas before the update happened… However:
  - What if it is unlikely that there will ever be a request to read that same value from other replicas?
  - At the same time with the update, there is a request to read this value from another process – which one came first?
- Global synchronisation takes a lot of time when replicas are spread across a wide area network.
- **Solution**: Loosen the consistency constraints! So, copies may not be always the same everywhere… To what extent consistency can be loosened depends on the access and update patterns of the replicated data as well as on the purpose for which those data are used.

Consistency Models

- A consistency model is a contract between processes and the data store. It says that if processes agree to obey certain rules, the store promises to work correctly.
- Normally, we expect that a read operation on a data item expects to return a value that shows the results of the last write operation on that data.
- In the absence of a global clock, it is difficult to define precisely which write operation is the last one!
- E.g., three processes, A, B, C, and three shared variables, x,y,z originally initialised to zero in 3 replicas – each process reads values from a different replica:

```
x=1                 y=1                 z=1
print(y,z)          print(x,z)          print(x,y)
```

There are many possible interleavings (e.g., A1,A2,B1,B2,C1,C2; C1,B1,B2,A1,C2,A2, etc) – the consistency model will specify what is possible…

Strict (tight) consistency

- Any read to a shared data item x returns the value stored by the most recent write operation on x.
- Implication: There is an absolute time ordering of all shared accesses. How can we achieve this?
- This makes sense in a local system. However, in a distributed system, this is not realistic since it requires absolute global time (e.g., how can you define ‘most recent’?)

Sequential Consistency

The result of any execution is the same as if operations of all processes were executed in some sequential order and the operations of each process appear in the order specified by the program.

- Example with 4 processes (P1, P2, P3, P4) – the horizontal axis indicates time.

<u>sequentially consistent</u>

P1: W(x)a
P2: W(x)b
P3: R(x)b R(x)a
P4: R(x)b R(x)a

<u>sequentially inconsistent</u>

P1: W(x)a
P2: W(x)b
P3: R(x)b R(x)a
P4: R(x)a R(x)b

R(x)a – read value a from variable x
W(x)a – write value a to variable x

Causal consistency

- Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order by different processes.
- Example:

not causally consistent:

```
P1: W(x)a
P2:        R(x)a  W(x)b
P3:                      R(x)b  R(x)a
P4:                      R(x)a  R(x)b
```

causally consistent:

```
P1: W(x)a
P2:              W(x)b
P3:                        R(x)b  R(x)a
P4:                        R(x)a  R(x)b
```

The following is allowed with a causally consistent store but not with a sequentially consistent store:

```
P1:      W(x)a                          W(x)c
P2:             R(x)a  W(x)b
P3:                           R(x)a                R(x)c  R(x)b
P4:                           R(x)a                R(x)b  R(x)c
```

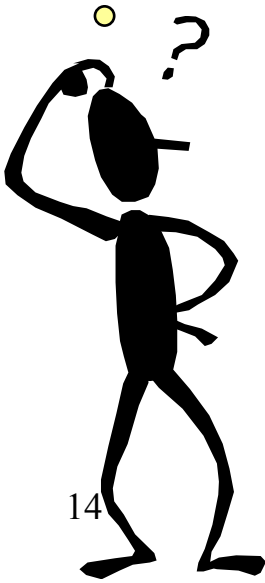
A thought on Causality

http://en.wikipedia.org/wiki/Causality
"The work of philosophers to understand causality and how best to characterize it extends over millennia"

Many more consistency models exist!

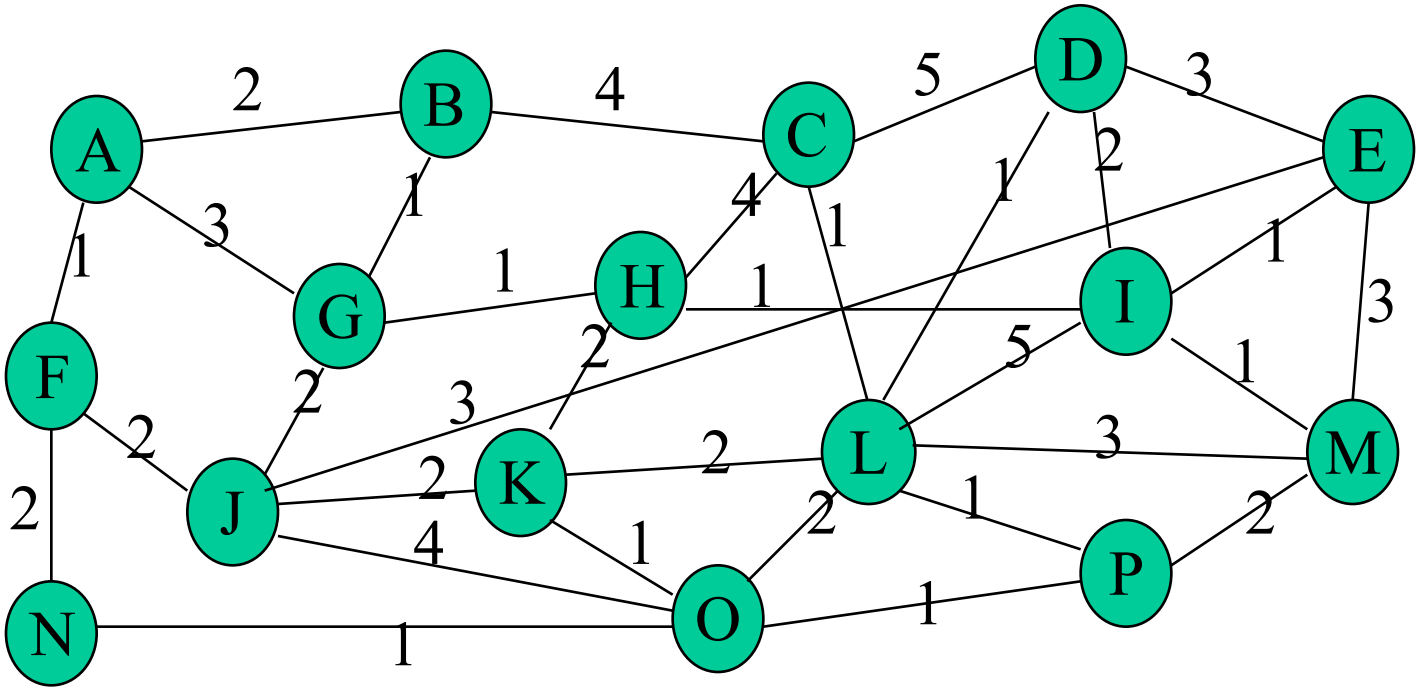
E.g., eventual consistency: given a sufficiently long period of time over which no changes are sent, all updates are expected to propagate, and eventually all replicas will be consistent.

"Eventual consistency. This is a specific form of weak consistency; the storage system guarantees that if no new updates are made to the object, eventually all accesses will return the last updated value. If no failures occur, the maximum size of the inconsistency window can be determined based on factors such as communication delays, the load on the system, and the number of replicas involved in the replication scheme. The most popular system that implements eventual consistency is the domain name system (DNS). Updates to a name are distributed according to a configured pattern and in combination with time-controlled caches; eventually, all clients will see the update." Communications of ACM January 2009, http://dl.acm.org/citation.cfm?id=1435432



Replica Management

- Where shall we place replica servers to minimize overall data transfer?
- In its general form it is a classical optimisation problem, but in practice it is often a management/commercial issue!



A greedy heuristic to find locations

(minimum k-median problem)

1. Find the total cost of accessing each site from all the other sites. Choose the site with the minimum total cost.
2. Repeat (1) above, taking also into account sites hosting replicas (i.e., recalculate costs).

   (see for more algorithms: '*on the placement of web server replicas*', INFOCOM2001)

Example: (using only nodes E, I, L, M, P from the graph in slide 14)

	E	**I**	**L**	**M**	**P**
E	0	1	5	2	4
I	1	0	4	1	3
L	5	4	0	3	1
M	2	1	3	0	2
P	4	3	1	2	0

Node M has the minimum cost!
Once we chose M, the next iteration may take into account the fact that sites attempt to access the nearest replica.

Step 2
(after we chose node M for replica hosting)

- Can (naively) just remove node M and repeat to find another node.
- Better, but more work, replace each value in the table by *min(value, value-to-M)*
- Note that this destroys any symmetry it had initially:

	E	**I**	**L**	**M**	**P**
E	0	1	5 → 2	2	4 → 2
I	1	0	4 → 1	1	3 → 1
L	5 → 3	4 → 3	0	3	1
M	2 → 0	1 → 0	3 → 0	0	2 → 0
P	4 → 2	3 → 2	1	2	0

Remember the trade-off!
Heuristics are useful when it is expensive to check exhaustively all possible combinations;
for *n* hosts & *m* replicas:
this is $n! / (m! (n-m)!)$

Conclusion

- Many of the problems related to replication and consistency have been repeatedly studied in the context of parallel and concurrent systems. We see those issues arising in:
  - Distributed file systems
  - Distributed shared memory.
- Some of the optimisation problems have been studied in the context of more generic theoretical (and operations research) problems.
- **Reading**: The Coulouris *et al* book provides some coverage in Sections 15.1-15.3 (4th edition) or 18.1-18.3 (5th edition). However, the material seems to be dispersed in various places. Better check Tanenbaum & Van Steen, "Distributed Systems", 2nd edition, Sections 7.1-7.4.