Sistemas Distribuídos

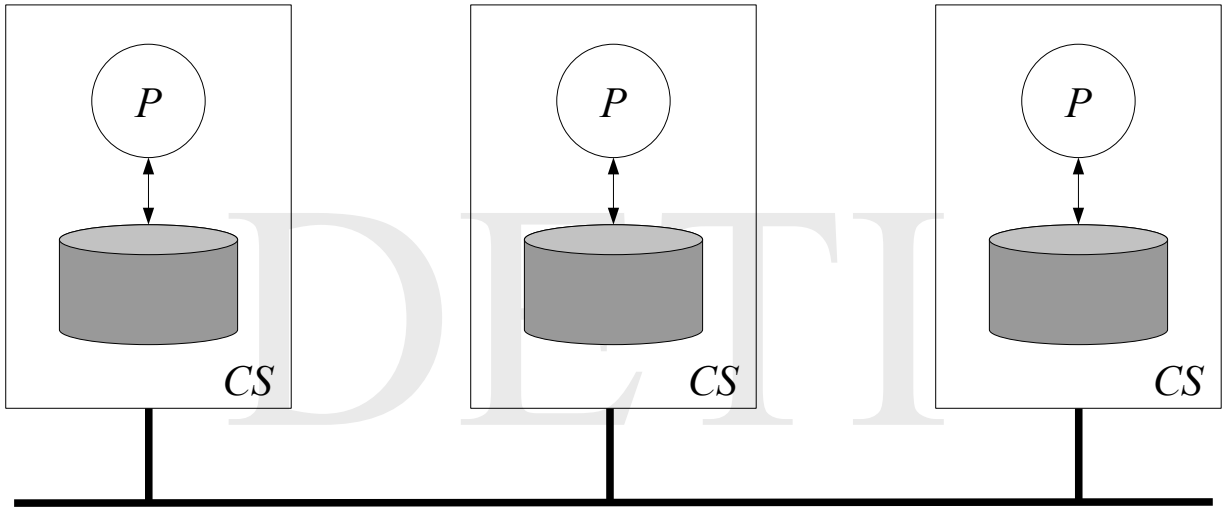
Consistency and Replication

António Rui Borges

Summary

- *Region of distributed storage*
  - *Characterization*
- *Consistency models*
  - *Criterion for 'correct operation'*
  - *Types*
    - *Strict consistency*
    - *Linearizability*
    - *Sequential consistency*
    - *Causal consistency*
    - *FIFO consistency*
- *Suggested reading*

Region of distributed storage - 1



A *region of distributed storage* can be thought of as a memory, a data base, or a file system, replicated over a set of processing nodes.

Region of distributed storage - 2

- it is assumed that
  - each processing node has a direct access to its local copy of the whole region
  - only two types of operations are relevant: *write*-like operations, which generate a modification of the contents of some register, and *read*-like operations, consisting of the remaining operations
  - when a *write*-like operation occurs, it must be propagated to all the copies so that an update takes place in each of them, *read*-like operations may, or may not, be propagated
  - access to the local copies of the region of distributed storage is thought to be carried out in parallel, that is, simultaneously, by the associated processes.

Consistency models

One defines *consistency model* as the set of rules, or contract, that if complied by the intervening processes, lead to the ‘correct operation’ of the region of distributed storage.

The issue, then, is defining with precision what one means by ‘correct operation’ and apply the concept in a systematic way.

Each particular model is characterized by specifying clearly the range of values a *read*-like operation may return vis a vis the *write*-like operations that have occurred previously.

When concurrency is assumed in access to data, the consistency models belong to a *data-centric* class.

Criterion for 'correct operation' - 1

Let $op_{i,0}$, $op_{i,1}$, $op_{i,2}$,…, with i = 0, 1,…, N-1, be any given sequence of *write*-like and *read*-like operations performed by the process i over the region of distributed storage.

Each operation is characterized by its type, arguments and returned value, and is supposed to be synchronous, that is, the next operation can only be executed when the one before has been terminated.

It is also assumed that the N processes, each performing its own sequence of operations, access in parallel the region of distributed storage. If the region were centralized, instead of distributed, the sequences of operations performed by the different processes would intermix in some manner, each time the accesses were instantiated, producing a global sequence of the type

$$op_{2,0}, op_{2,1}, op_{0,0}, op_{1,0}, op_{1,1}, op_{2,2}, op_{2,3}, op_{0,1}, op_{1,2}, \ldots .$$

Criterion for 'correct operation' - 2

The *criterion for 'correct operation'* of the region of distributed storage is now defined by referring to one or more global canonical sequences that establish the pattern of operation and which are produced by intermixing the partial sequences of the individual processes.

These canonical sequences are merely virtual, they do not have actually to occur. Their role is to serve as certification of what one calls '*correct operation*' to a given instantiation of parallel access to the region of distributed storage.

Based on what is perceived in the instantiation, the inexistence of a canonical sequence which obeys to the properties of the referenced consistency model, allows one to conclude that the model is not valid.

Strict consistency - 1

Any read-like operation performed on the register x returns the value produced by the most recent write-like operation on x.

It is the ideal situation which can only be achieved in monoprocessor systems.

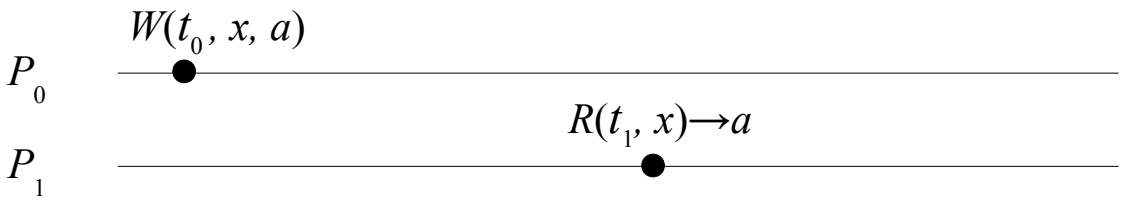
This model can not be implemented in parallel machines, because the notion of 'most recent' event is ambiguous in this context. The ambiguity stems from the fact the propagation speed of any physical signal is necessarily finite.

Therefore,

- the local clocks of the different processing nodes can not be synchronized with absolute accuracy, resulting in the impossibility of having a global clock to order chronologically the events
- the execution time of write like and read like operations is not null.

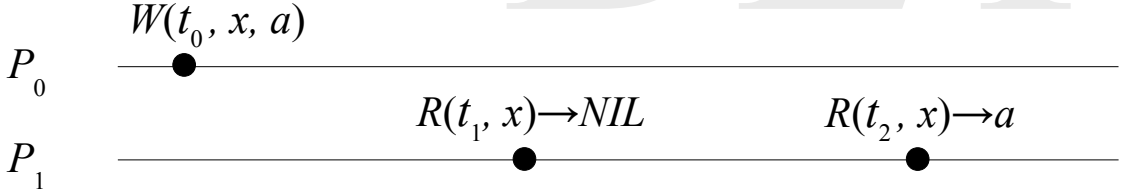
Hence, events can not be ordered in terms of an unique time standard.

Strict consistency - 2



strictly consistent storage region

canonical sequence: $t_0 < t_1 \Rightarrow W_0(t_0, x, a)$ - $R(t_1, x) \rightarrow a$



non strictly consistent storage region

there is no canonical sequence which mimics the results

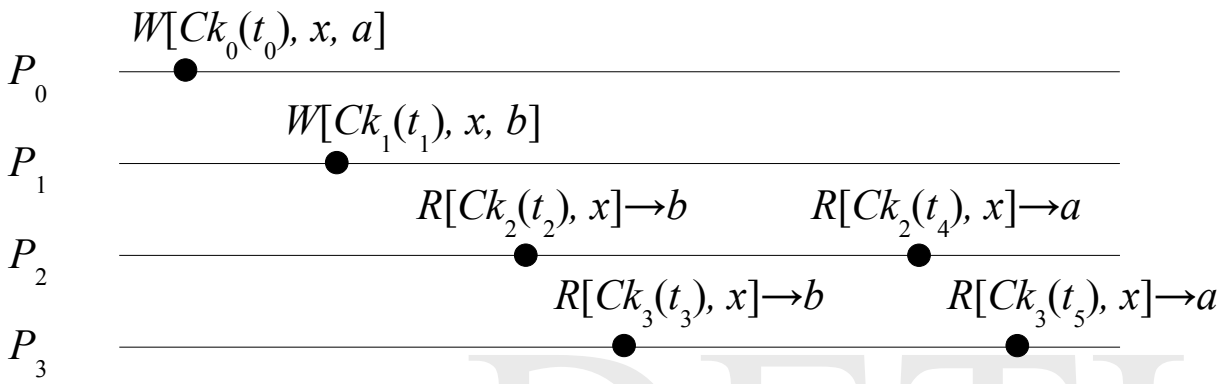
Linearizability - 1

Parallel access to a register x is seen by all involved processes as if the performed operations were ordered in an unique and well-defined sequence by keeping the chronological order that is locally perceived (Lamport / Herlihy & Wing).

This model supposes there is some synchronization procedure of the local clocks which *total orders* the events (*it is an approximation to the notion of global clock*). The goal is to ensure that the intervening processes always get the most updated data.

One aims to convey the idea that, for the sequences of operations carried out by the intervening processes in parallel, there is a canonical sequence of global execution of the same operations over a concentrated virtual region which transmits an image of unique execution, consistent with the chronological execution of the operations as they are locally perceived.

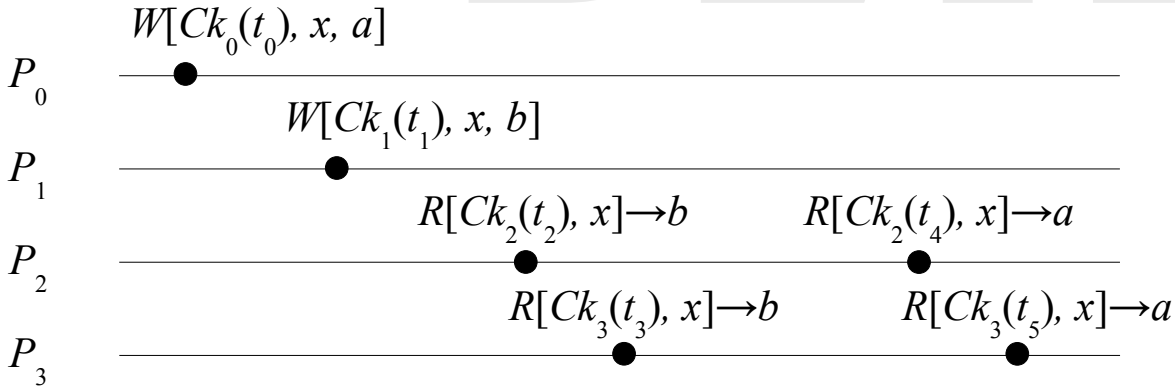
Linearizability - 2



linearizable storage region

canonical sequence: $Ck_1(t_1) < Ck_2(t_2)$, $Ck_3(t_3) < Ck_0(t_0) < Ck_2(t_4)$, $Ck_3(t_5)$ $\Rightarrow$

$\Rightarrow$ $W_1[Ck_1(t_1), x, b]$ - $R_{2,3}(-, x) \rightarrow b$ - $W_0[Ck_0(t_0), x, a]$ - $R_{2,3}(-, x) \rightarrow a$



non linearizable storage region

$Ck_0(t_0) < Ck_1(t_1)$ $\Rightarrow$ *there is no canonical sequence which mimics the results*

Linearizability - 3

Implementation

All *read*-like and *write*-like operations must be propagated and acknowledged by all the processes managing the local copies of the region of distributed storage, before the associated action takes place.

This can be achieved by *total ordering* the events associated with the operations through the use of a *logic scalar* clock, as the local clock, and adjusting it according to the rules prescribed by Lamport.

Possible application areas

All the cases where the most strict fairness is required. Government support services or financial transaction systems, for instance.

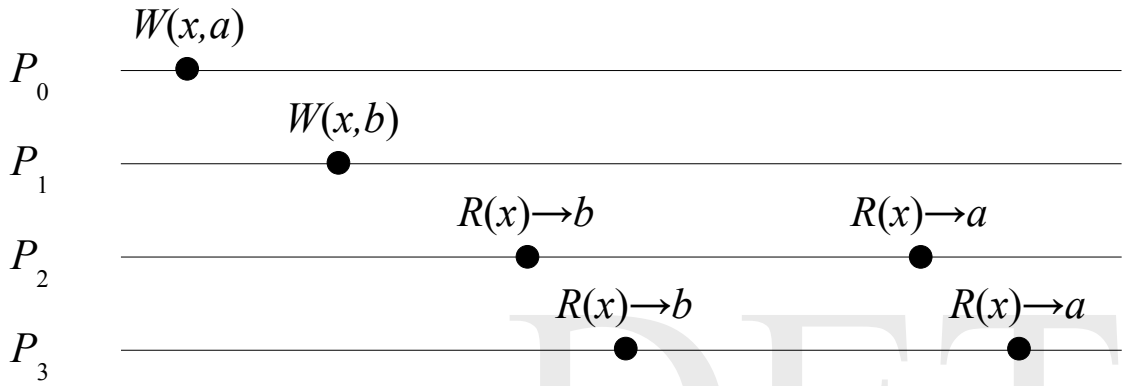
Sequential consistency - 1

Parallel access to a register x is seen by all involved processes as if the performed operations were ordered in an unique and well-defined sequence where the operations of each individual process keep the order of local execution (Lamport / Herlihy & Wing).

The first observation is that time, whether global, or local, does not play an explicit role in this framing. The second is that, because of this, it is not necessary to total order all the occurring events.

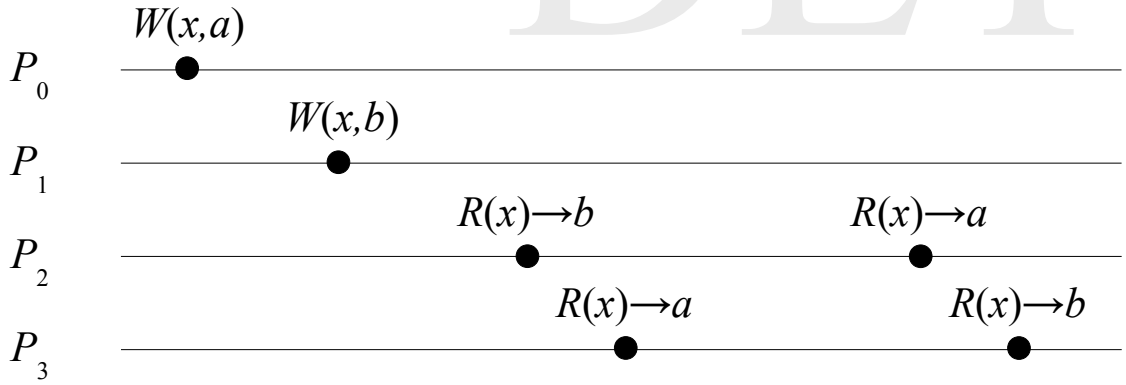
One aims to convey the idea that, for the sequences of operations carried out by the intervening processes in parallel, there is a canonical sequence of global execution of the same operations over a concentrated virtual region which transmits an image of unique execution, consistent with the successive execution of the local operations.

Sequential consistency - 2



sequentially consistent storage region

canonical sequence: $W_1(x,b)$ - $R_{2,3}(x)\rightarrow b$ - $W_0(x,a)\rightarrow a$ – $R_{2,3}(x)\rightarrow a$



non sequentially consistent storage region

there is no canonical sequence which mimics the results

Sequential consistency - 3

Implementation

Only *write*-like operations must be propagated and acknowledged by all the processes managing the local copies of the region of distributed storage, before the associated action takes place.

This can be achieved by *total ordering* the events associated with the *write*-like operations through the use of a *logic scalar* clock, as the local clock, and adjusting it according to the rules prescribed by Lamport.

Possible application areas

All the cases where fairness is in general required. Booking systems and e-commerce, for instance.

Causal consistency - 1

Parallel access to a register x is seen by all involved processes as if the performed operations, which keep between them a causal relation, were ordered in an unique and well-defined sequence. The remaining operations, said to be concurrent, may be perceived in any order by the different processes (Hutto / Ahamad).

In order to have a clear understanding of what this criterion states, it is necessary to define in detail the meaning of *causality* as it is used here.

Thus, one considers that

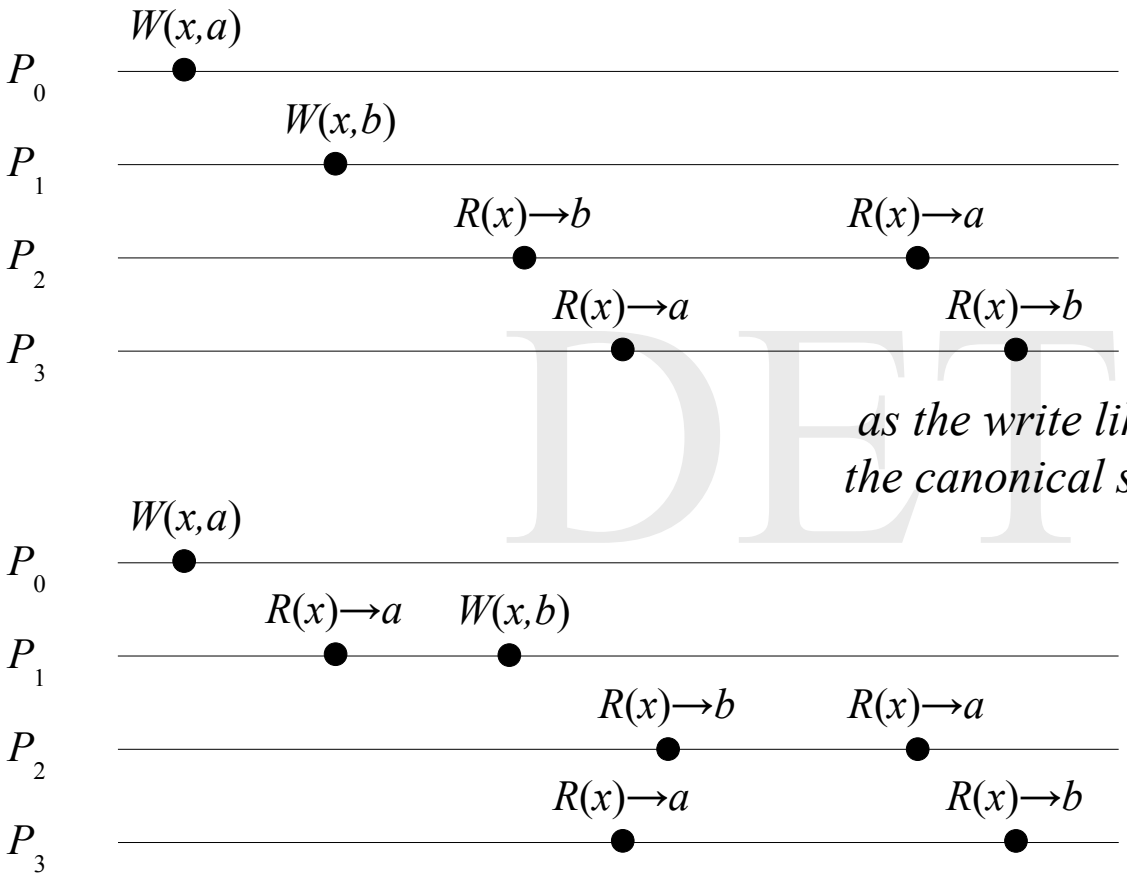
- the operations which are executed in succession by the same process are *causal* related among themselves
- a read like operation is always *causally* related to the write like operation which has supplied the value that was read, whether or not the process that executed each operation is the same.

Causal consistency - 2

The idea that is conveyed here is that, for the sequences of operations carried out by the intervening processes in parallel, there is a canonical sequence of global execution of the same operations over a concentrated virtual region which transmits an image of unique execution, consistent with the successive execution of the local operations, is restricted to the subset of operations that keep between them a causal relation.

Only for these operations the criterion of sequential consistency is imposed. All the remaining operations can be perceived in any order by the different intervening processes.

Causal consistency - 3



causally consistent storage region

as the write like operations are concurrent,
the canonical sequence is in this case empty

non causally consistent storage region

as the write like operations are causally related,
there is no canonical sequence which mimics the results

Causal consistency - 4

Implementation

Only *write*-like operations must be propagated and acknowledged by all the processes managing the local copies of the region of distributed storage.

This can be achieved by *partial ordering* the events associated with the *write*-like operations through the use of a *logic vector* clock, as the local clock, and adjusting it according to the rules prescribed for logic vector clocks.

Possible application areas

All the cases where causality is in general required. Chatting applications, for instance.

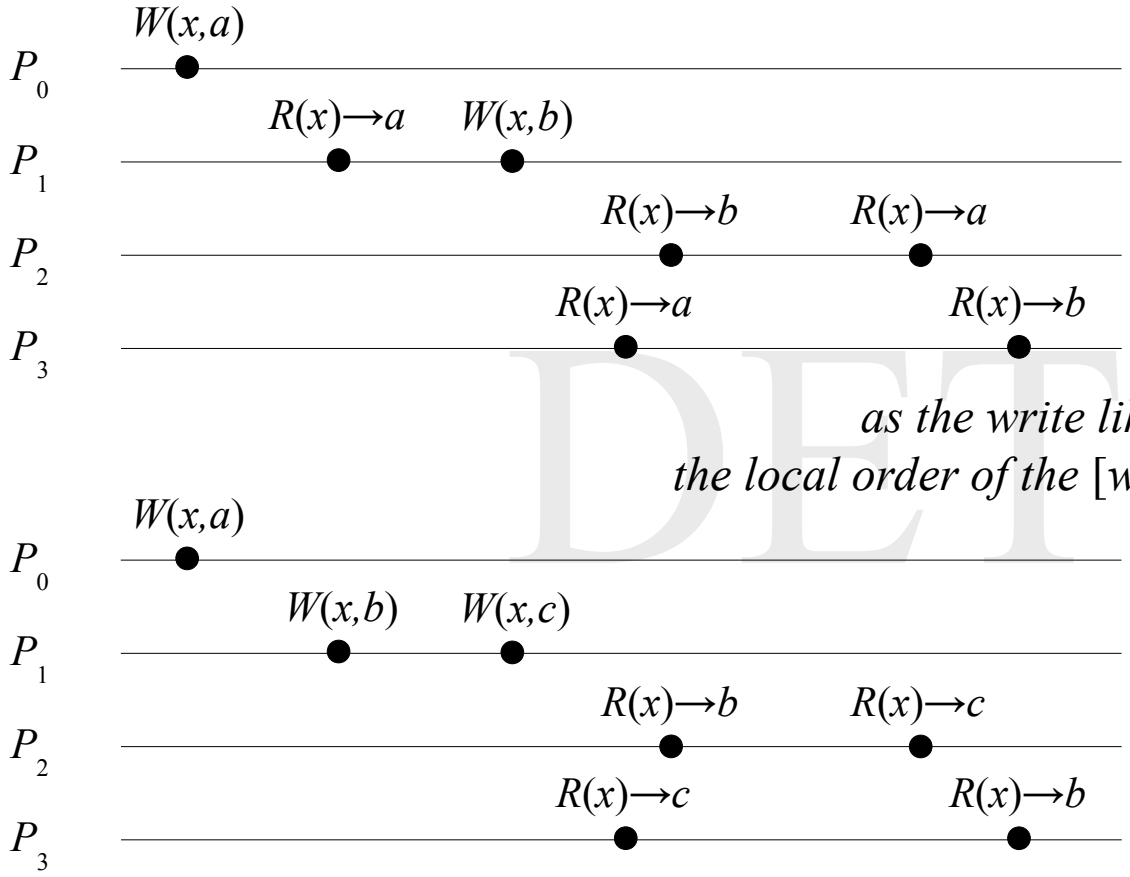
FIFO consistency - 1

Parallel access to a register x is seen by all involved processes as if the performed operations by any given process keep the order of local execution (Lipton / Sandberg).

One should note that in this case the existence of a canonical sequence of global execution of the same operations over a concentrated virtual region which transmits an image of unique execution, is irrelevant.

As long as the order of execution of the local operations is perceived as the same by all the intervening processes, and one is only referring to the write like operations, since the read like operations are not important, because they do not produce any effect, the intermixing of the partial sequences can be whatever and perceived in a different way by the intervening processes.

FIFO consistency - 2



FIFO consistent storage region

as the write like operations are concurrent, the local order of the [*write like*] *operations is trivial*

non FIFO consistent storage region

the local order of the [*write like*] *operations executed by process* P_1 *is not perceived as such by process* P_3

FIFO consistency - 3

Implementation

Only *write*-like operations must be propagated by all the processes managing the local copies of the region of distributed storage.

This can be achieved by *ordering* per process the events associated with the *write*-like operations through the use of a *message counter*.

Possible application areas

All the cases where a less degree of consistency is required. News organizations, weather forecasting, for instance.

Suggested reading

- *Distributed Systems: Concepts and Design, 4th Edition, Coulouris*, Dollimore, Kindberg, Addison-Wesley
  - Chapter 12: C*oordination and agreement*
    - Section 12.4
  - Chapter 15: *Replication*
    - Sections 15.1 and 15.2
- *Distributed Systems: Principles and Paradigms, 2nd Edition,* Tanenbaum, van Steen, Pearson Education Inc.
  - Chapter 7: *Consistency and replication*
    - Sections 7.1 and 7.2