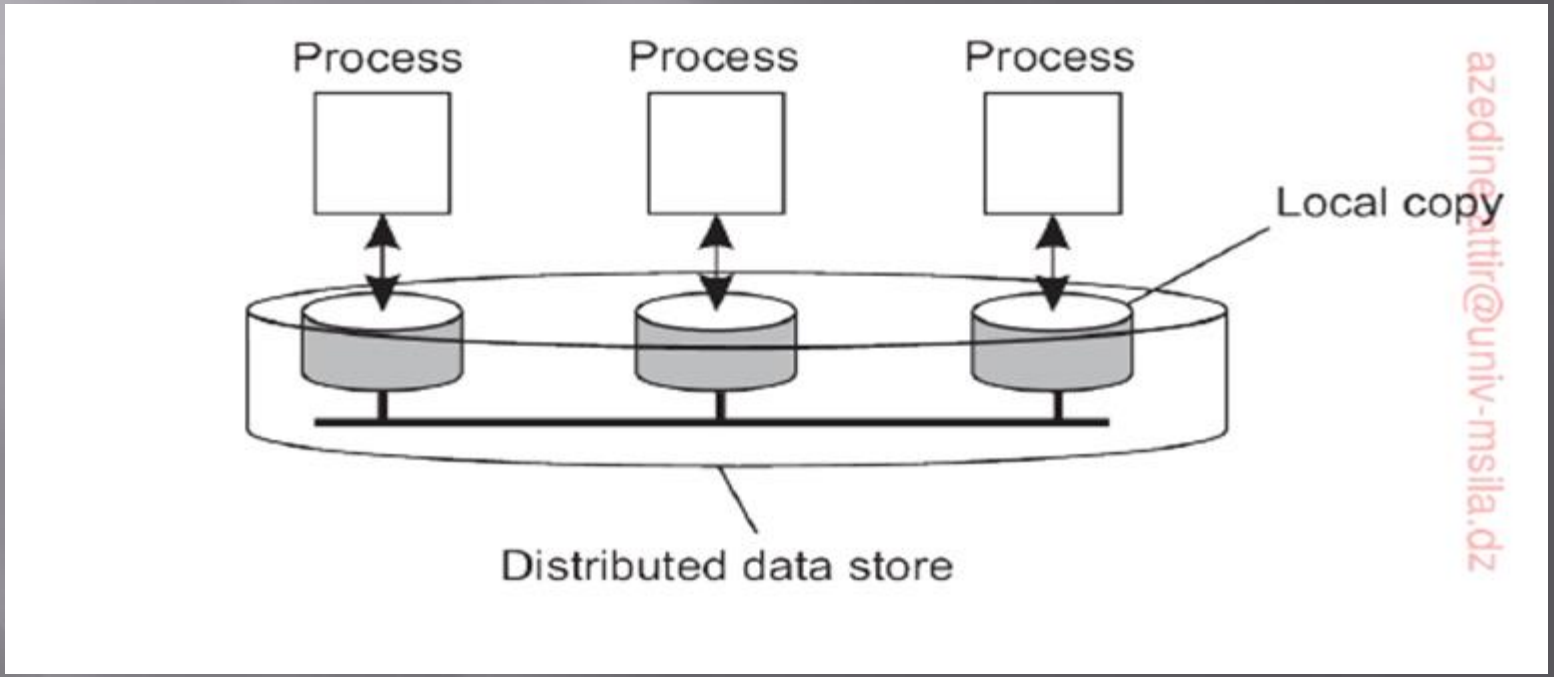
La cohérence de données dans les systèmes distribués

Cours 8

The general organization of a logical data store, physically distributed and replicated across multiple processes.

Andrew S. Tanenbaum and Marteen van Steen.Distributed Systems. Principles and Paradigms.Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition,2007



The general organization of distributed data store

- Write operations are propagated to the other copies. A data operation is classified as a write operation when it changes the data, and is otherwise classified as a read operation.

Cont..

- Replicating data poses consistency problems that cannot be solved efficiently in a general way.
- Only if we loosen consistency can there be hope for attaining efficient solutions.
- Unfortunately, exactly what can be tolerated is highly dependent on applications. Page 359-360 of textbook.

Consistency in distributed Systems

- In computer science, consistency models are used in distributed systems like **distributed shared memory systems** or **distributed data stores** (such as a filesystems, databases, optimistic replication systems or web caching).

B. Les modèles de consistance

- Normally, a process that performs a read operation on a data item, expects the operation to return a value that shows the results of the last write operation on that data.

Cont…

- One consistency model can be considered stronger than another if it requires all conditions of that model and more. In other words, a model with fewer constraints is considered a weaker consistency model.

Solutions au problème d'incohérence (Consistency)

- Modèle de cohérence.
- Les modèles de cohérence : sont des contrats basés sur des règles. Ils permettent de gérer plusieurs copies de données dans des mémoires dans un systèmes répartis.
  - ✓ La cohérence stricte
  - ✓ Cohérence séquentielle.
  - ✓ Cohérence causale.
  - PRAM (Pipelined RAM)
  - Cohérence processeurs (faible, relâchée, par entrée).

a- La cohérence stricte

- La règle de la cohérence stricte est la suivante : Toute lecture d'une position de mémoire x retourne la valeur stockée par l'écriture la plus récente.
- Plus contraignante
- Elle implique que lorsque la mémoire est basée sur la cohérence stricte :
  - Tous les processus voient instantanément toutes les écritures qui y sont effectuées
  - Lorsqu'une lecture est effectuée, la dernière mise à jour et fournie. Peu importe lorsque la prochaine écriture se produira

Annotation et exemple

- *W(x)a* : écrire *a* dans *x* et *R(y)b* : la lecture de *y* return la valeur *b*.
- La valeur initiale de chaque variable est 0.

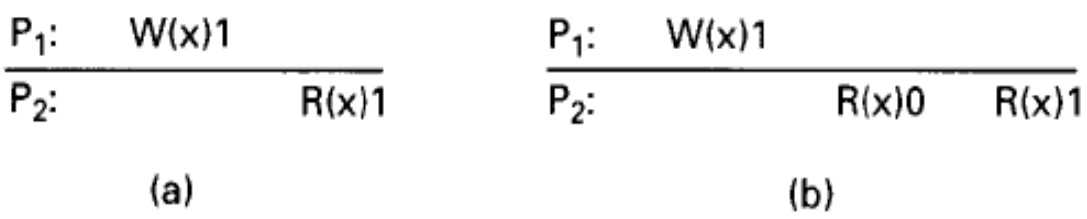


Fig. 6-12. Behavior of two processes. The horizontal axis is time. (a) Strictly consistent memory. (b) Memory that is not strictly consistent.

b- Cohérence séquentielle

- First defined by Lamport [1979], A data store is said to be sequentially consistent when it satisfies the following condition:
- The result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program.

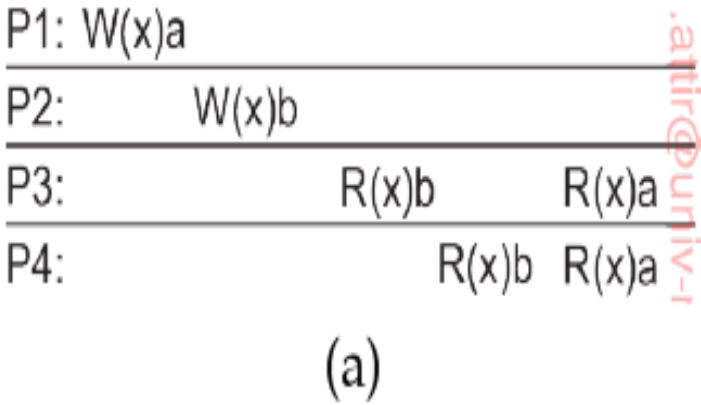
b- Cohérence séquentielle

- What this definition means is that when processes run concurrently on (possibly) different machines, any valid interleaving of read and write operations is acceptable behavior, but all processes see the same interleaving of operations.
- Also, a process "sees" the writes from all processes but only through its own reads.
- Note that nothing is said about time.

b- Cohérence séquentielle

- Le résultat de toute exécution est le même que si les opérations de tous les processus étaient exécutées dans un ordre séquentiel quelconque, ***et*** les opérations de chaque processus apparaîtrait dans cette séquence dans l'ordre spécifié par leur programme.

Exemple



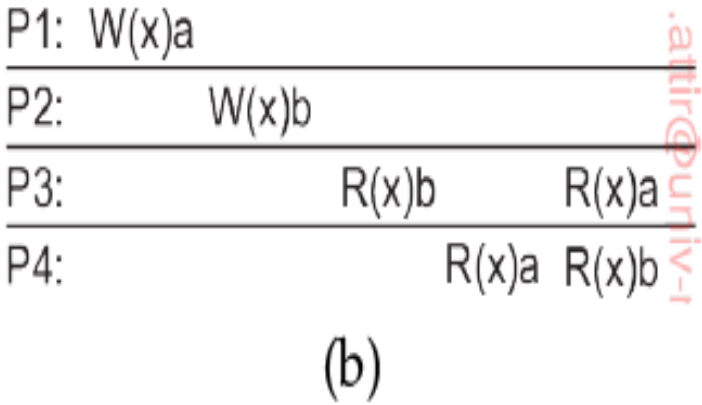


Figure 7.5: (a) A sequentially consistent data store. (b) A data store that is not sequentially consistent.

c- La cohérence causale Hutto et Ahamad (1990)

- For a data store to be considered causally consistent, it is necessary that the store obeys the following condition:
- 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 on different machines.

c- La cohérence causale Hutto et Ahamad (1990)

- Cette règle est une version affaiblie de la règle de cohérence séquentielle, qui tient compte de la relation causale **potentielle** ou pas entre les événements. La règle stipule :
- Les écritures qui ont **potentiellement** une relation causale doivent être vus par tous les processus dans le même ordre. Des écritures parallèles peuvent être vus dans un ordre différent sur différentes machines.

c- La cohérence causale Hutto et Ahamad (1990)

- Suppose that process P1 writes a data item x. Then P2 reads x and writes y. Here the reading of x and the writing of y are potentially causally related because the computation of y may have depended on the value of x as read by P2.
- On the other hand, if two processes spontaneously and simultaneously write two different data items, these are not causally related.
- Operations that are not causally related are said to be **concurrent.**

Exemple

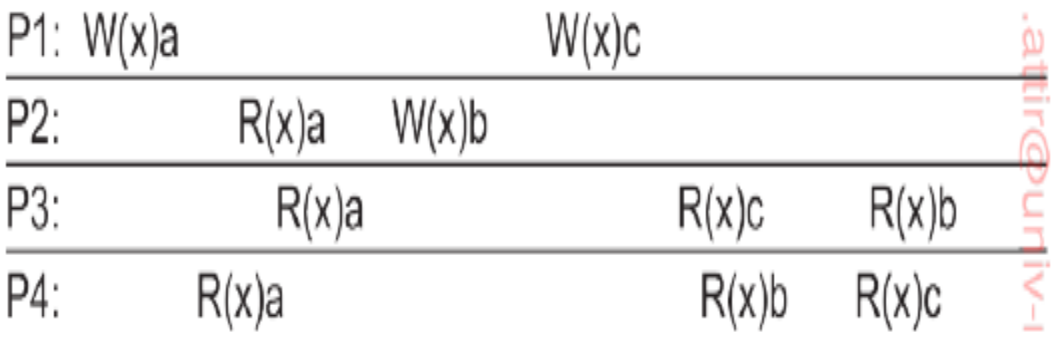


Figure 7.10: This sequence is allowed with a causally-consistent store, but not with a sequentially consistent store.

Exemple d'application des modèle de cohérence

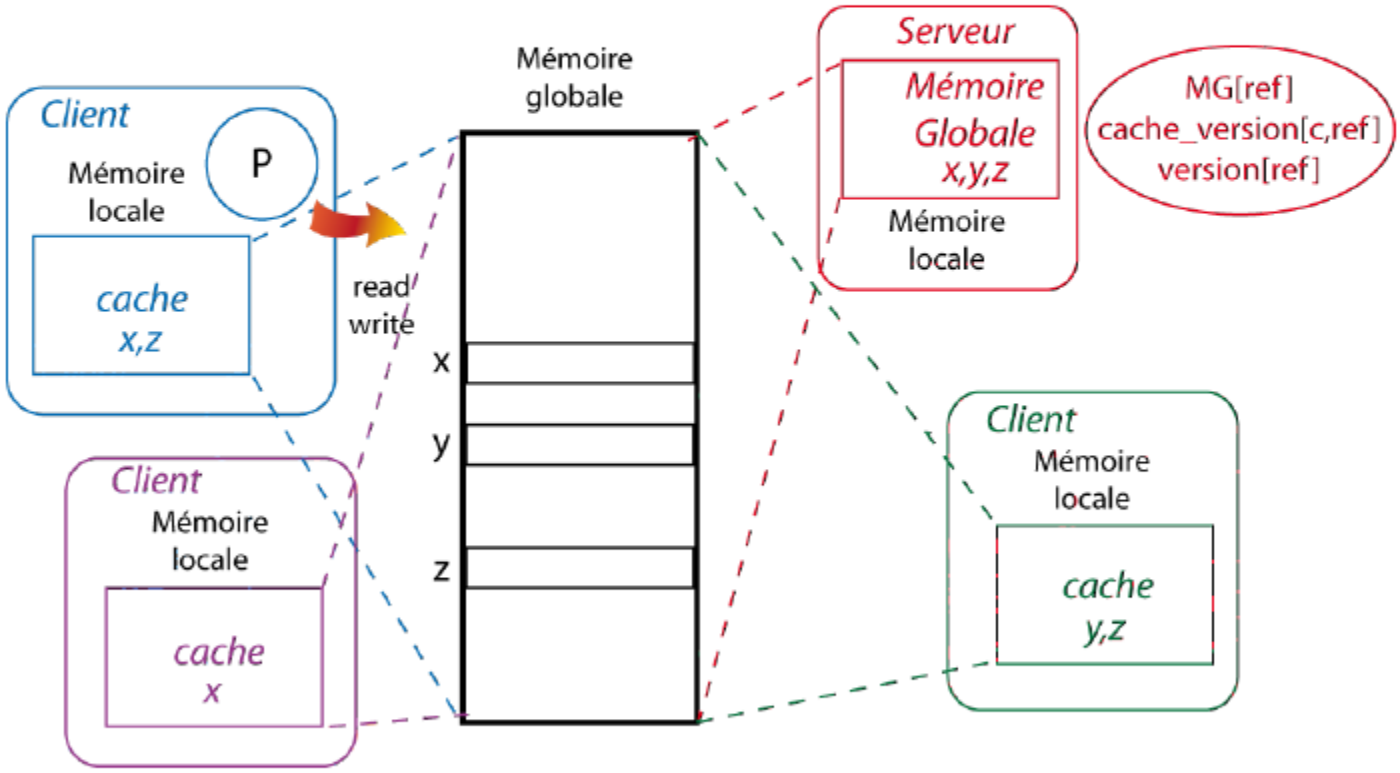
- Cours : Algorithmique repartie, Notion de coherence dans les memoires reparties, Gerard Padiou. Departement Informatique et Mathematiques appliquees ENSEEIHT. 5 novembre 2012

Plan

1. Notion de mémoire répartie
   - Le problème
   - Modélisation
   - Cohérence forte
2. Exemple
   - Notion d'histoire
   - Notion de cohérence
3. Types de cohérence
   - Cohérence séquentielle
   - Cohérence causale
   - Cohérence PRAM
4. Exemples
   - Une solution de type client/serveur
   - Une solution par réplication

Une solution de type client/serveur

Usage de caches mais aussi d'un serveur unique « ordonnanceur »



L'algorithme

Principes

☞ L'algorithme garantit la cohérence mémoire causale

- Chaque site gère un cache de données ;
- Une lecture se fait dans le cache local si possible sinon à distance (auprès du site serveur) ;
- Toute écriture est transmise au site serveur ;
- Les données en cache sont éventuellement invalidées en réponse à des écritures ou lectures distantes ;
- L'invalidation utilise un mécanisme de version sur les écritures.

Structures de données de l'algorithme

sur le serveur :

- `MG[ref]` : la mémoire globale ;
- `version[ref]` : les numéros de version courante de chaque mot référencé ;
- `version_cache[c,ref]` : les numéros de version des références en cache chez chaque client.

pour un client c :

- $\texttt{cache}_c\texttt{[ref]}$: cache du client c pour toute variable référencée `ref` ;
- $\texttt{valide}_c$: l'ensemble des références de variables valides dans le cache du client c.

Algorithmes des opérations : site client

Lecture sur un site Client c

```
value read(ref x) {
  if (x ∉ valide_c) { /* x n'est pas en cache local */
   invalide,v = Serveur.read(x,c) ;  /* - RPC - */
   valide_c = (valide_c-invalide)+{x} ; cache_c[x] = v ;
  }
  return cache_c[x]
}
```

Écriture sur un site Client c

```
write(ref x, value v) {
  invalide = Serveur.write(x,v,c) ; /* - RPC - */
  valide_c = (valide_c-invalide)+{x} ; cache_c[x] = v ;
}
```

Algorithmes des opérations : site serveur

Lecture sur le serveur : appel d'un client *c*

```
⟨set<ref>,v⟩ read(ref x, client c) {
  cache_version[c,x]=version[x];
  return ⟨invalider(c,x),MG[x]⟩
}
```

Écriture sur le site Serveur : appel d'un client *c*

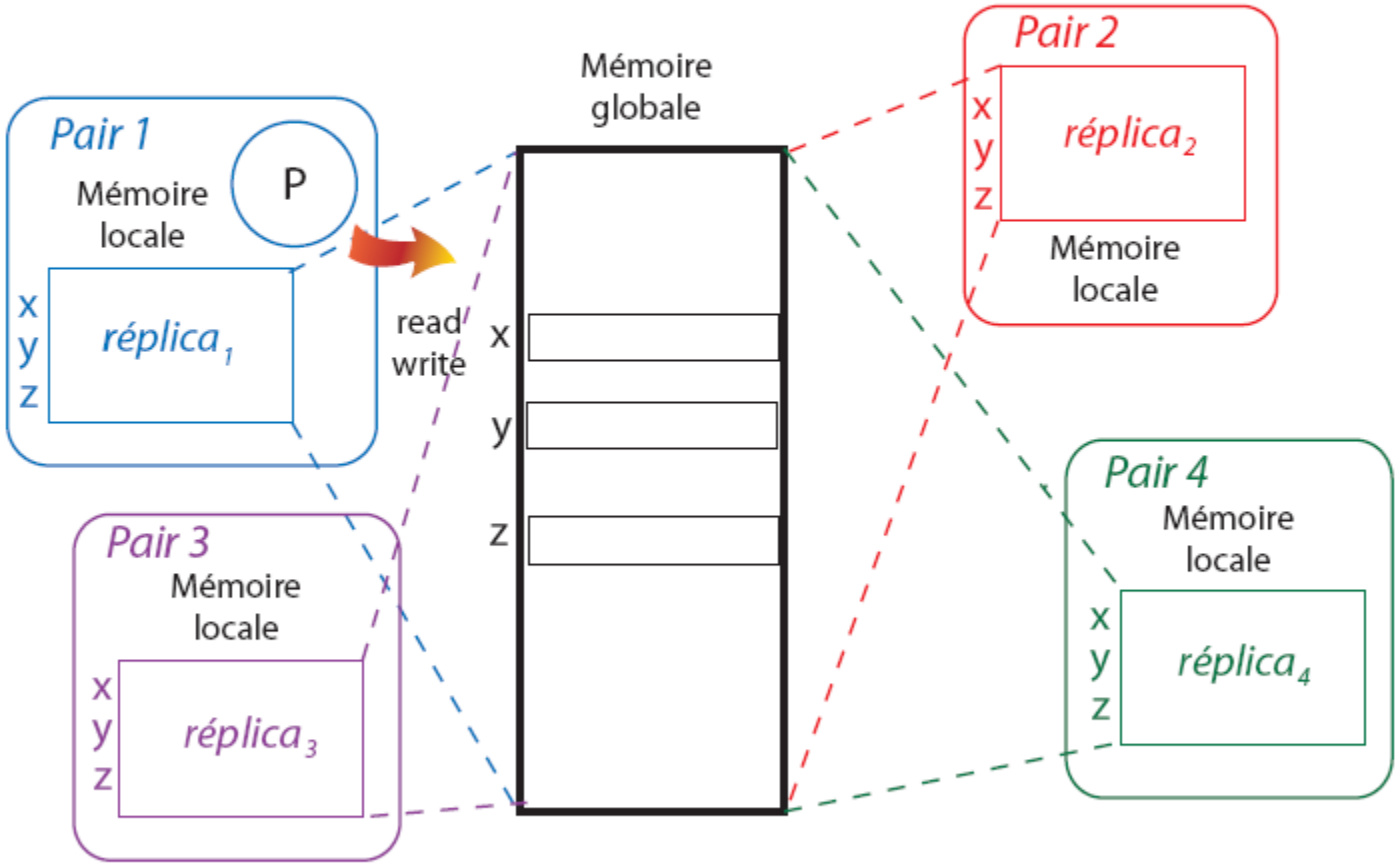
```
set<ref> write(ref x, value v, client c) {
  MG[x]=v; version[x]++; /* nouvelle version */
  cache_version[c,x]=version[x];
  return invalider(c,x)
}
```

Algorithmes des opérations : site serveur

Algorithme d'invalidation

```
set<ref> invalider(client c, ref x) {
  set<ref> invalide = {};
  for (ref y ∈ MG && y ≠ x) {
   if (cache_version[c,y]≠0 // c possède y en cache
   && cache_version[c,y]<version[y]){ // cache obsolète
    invalide = invalide + {y};
    cache_version[c,y] = 0; // y effacé du cache de c
   }
  }
  return invalide
}
```

Une solution de type répliquée



Principes de l'algorithme

Principes

☞ L'algorithme garantit la cohérence mémoire séquentielle

- Chaque site gère une copie de la mémoire globale répliquée ;
- Une lecture se fait toujours sur le réplica local ;
- Toute écriture est diffusée à tous les sites ;
- Pour obtenir la cohérence séquentielle :
  $\Rightarrow$ utiliser un protocole de diffusion totalement ordonnée ;
- Lecture optimale (locale) ;
- Écriture coûteuse (diffusion totalement ordonnée)

Algorithme des opérations

Lecture sur le site *i*

```
value read(ref x) { return réplica_i[x] }
```

Écriture sur le site *i*

```
write(ref x, value v) {
  abroadcast(x,v,i) ; /* diffusion tot. ordonnée */
  écrit_i[x].wait() ;
}
```

Réception d'une requête d'écriture issue du site *i* sur le site *j*

```
upon receive(ref x, value v, site i) {
  réplica_j[x]= v ;
  if (j==i) écrit_i[x].notify() ;
}
```

Comment ?

- Montrer comment l'algorithme 1 garantit la cohérence mémoire causale ?
- Montrer comment l'algorithme 2 garantit la cohérence mémoire séquentielle ?

Cours 9

Consistancy Protocols

From textbook

Reasons for replication

- There are two primary reasons for replicating data.

1. First, data are replicated to increase the reliability of a system.
2. The other reason for replicating data is performance.

Defintion

- A **consistency protocol** describes an implementation of a specific consistency model.

Primary-based protocols

- In practice, we see that distributed applications generally follow consistency models that are relatively easy to understand.
- When it comes to models that handle consistent ordering of operations, sequential consistency, notably those in which operations can be grouped through locking or transactions are popular.

- As soon as consistency models become slightly difficult to understand for application developers, we see that they are ignored even if performance could be improved.
- The bottom line is the semantics of a consistency model are not intuitively clear.
- In the case of **sequential consistency**, it turns out that **primary-based protocols prevail.**

- In these protocols, each data item **x** in the data store has an **associated primary**, which is responsible for coordinating write operations on **x**.
- A distinction can be made as to whether the primary is fixed at a remote server or if write operations can be carried out locally after moving the primary to the process where the write operation is initiated.

a- Remote-write protocols

- Known as **primary-backup protocols.**
- The simplest primary-based protocol that supports replication is the one in which all write operations need to be forwarded to a fixed single server. Read operations can be carried out locally.
- A primary-backup protocol works as shown in Figure 7.27.
- Budhijara N., Marzullo K., Schneider F., and Toueg S. The Primary-Backup Approach. In Mullender S., editor, Distributed Systems, pages 199–216. Addison-Wesley, Wokingham, 2nd edition, 1993. Cited on 399.

- A process wanting to perform a write operation on data item **x**, forwards that operation to the primary server for **x**.
- The primary performs the update on its local copy of **x**, and subsequently forwards the update to the backup servers.
- Each backup server performs the update as well, and sends an acknowledgment to the primary.
- When all backups have updated their local copy, the primary sends an acknowledgment to the initial process, which, in turn, informs the client.

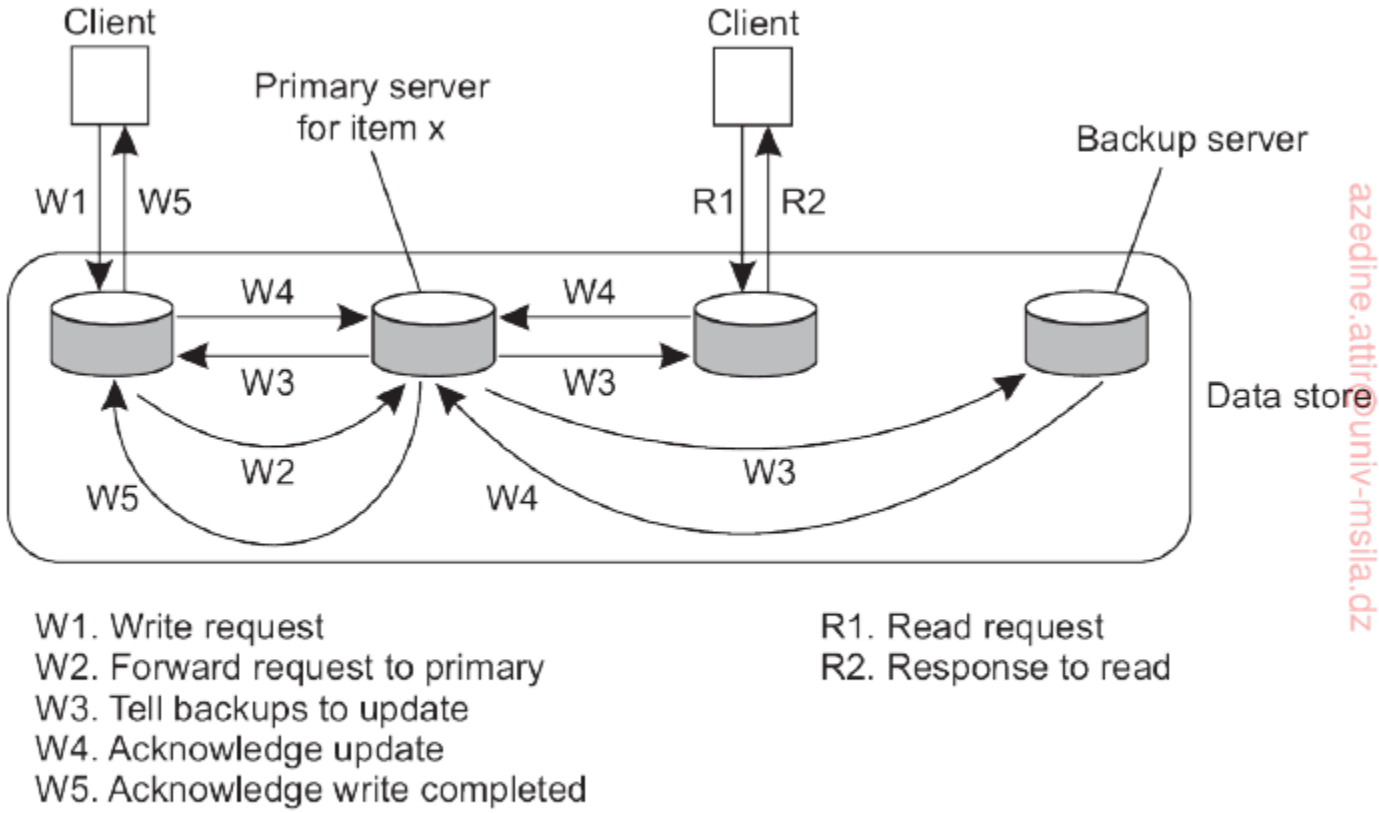


Figure 7.27: The principle of a primary-backup protocol.

- A potential performance problem with this scheme is that it may take a relatively long time before the process that initiated the update is allowed to continue. **In effect, an update is implemented as a blocking operation**.
- An alternative is to use a non-blocking approach. As soon as the primary has updated its local copy of x, it returns an acknowledgment. After that, it tells the backup servers to perform the update as well.
- Find -if exists- drawbacks of non-blocking primary-backup protocols.

- Non-blocking primary backup protocols are discussed in : Budhiraja N. and Marzullo K. Tradeoffs in Implementing Primary-Backup Protocols. Technical Report TR 92-1307, Department of Computer Science, Cornell University, 1992. Cited on 400

- Primary-backup protocols provide a straightforward implementation of sequential consistency, as the primary can order all incoming writes in a globally unique time order. Evidently, all processes see all write operations in the same order, no matter which backup server they use to perform read operations.

b- **Local-write protocols**

- A variant of primary-backup protocols.
- The primary copy migrates between processes that wish to perform a write operation.
- Whenever a process wants to update data item **x**, it locates the primary copy of x, and subsequently moves it to its own location, as shown in Figure 7.28.

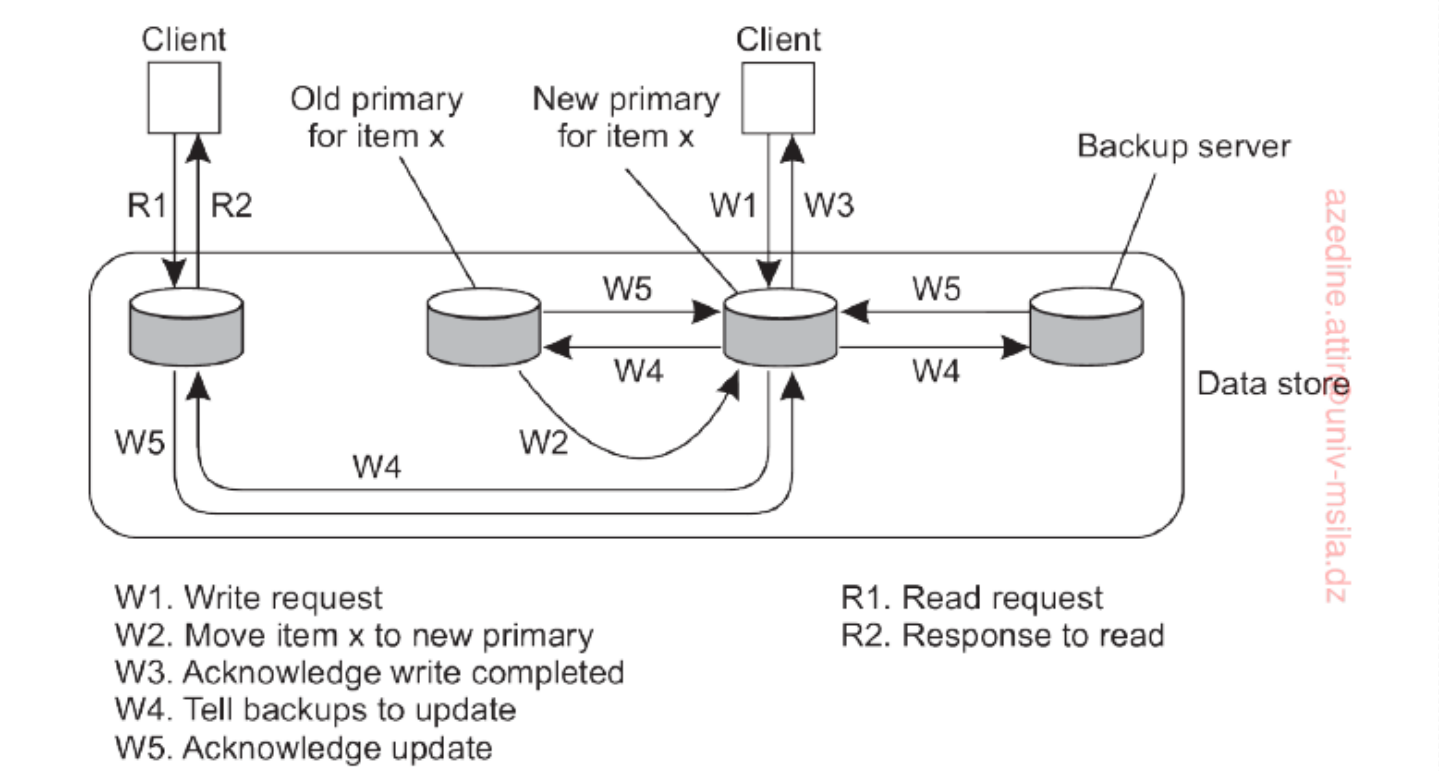


Figure 7.28: Primary-backup protocol in which the primary migrates to the process wanting to perform an update.

- The main advantage of this approach is that multiple, successive write operations can be carried out locally, while reading processes can still access their local copy.

Replicated-write protocols

- In replicated-write protocols, write operations can be carried out at multiple replicas instead of only one.

Active replication

- Each replica has an associated process that carries out update operations.
- Updates are generally propagated by means of the write operation that causes the update.
- One problem with active replication is that operations need to be carried out in the same order everywhere.

Figure From the book Distributed Network Systems From Concepts to Implementations, Ding-Zhu Du and Cauligi Raghavendra

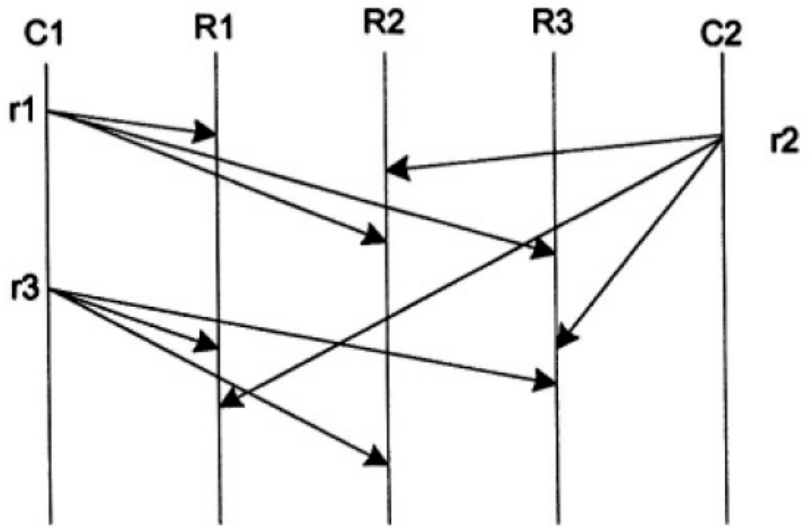


Figure 9.22. The scenario of requests arriving in different orders -- R_1, R_2 and R_3 represent replicas; c_1 and c_2, represent clients; and r_1, r_2, and r_3 represent messages. *Two clients,* c_1 *and* c_2, *send requests to all replicas;* c_1 *sends* r_1 *and* r_3, *and* c_2 *sends* r_2. *But* $\{r_1, r_2, r_3\}$ *arrives at* R_1, R_2 *and* R_3 *in the orders of,* $<r_1, r_3, r_2>$, $<r_2, r_1, r_3>$ *and* $<r_1, r_2, r_3>$ *respectively.*

- Consequently, what is needed is a total ordered multicast mechanism.
- A practical approach to accomplish total ordering is by means of a central coordinator, also called a **sequencer.**
- One approach is to first forward each operation to the sequencer, which assigns it a unique sequence number and subsequently forwards the operation to all replicas. Operations are carried out in the order of their sequence number.

Homework

- Find applications of each consistancy protocols.
- TOA of Facebook.