Timed Consistency: Unifying Model of Consistency Protocols in Distributed Systems

Francisco J. Torres-Rojas[1] and Esteban Meneses[2]

[1] *Centro de Investigaciones en Computación e Informática Avanzada* (CIenCIA) and Instituto Tecnológico de Costa Rica {torres@ic-itcr.ac.cr}
[2] *Centro de Investigaciones en Computación* (CIC) and Instituto Tecnológico de Costa Rica {emeneses@ic-itcr.ac.cr}

Abstract. Ordering and timeliness are two different aspects of consistency of shared objects in distributed systems. *Timed consistency* [12] is an approach that considers simultaneously these two elements according to the needs of the system. Hence, most of well known consistency protocols are candidates to be unified under the Timed consistency approach, just by changing some of the time or order parameters.

Key words: timed consistency, consistency protocols, distributed systems.

1 Introduction

A distributed application is built up from processes executed at different nodes from possibly distant locations. One important problem arises when we want to maintain the consistency of the state shared by such processes. This task cannot be considered trivial, given the failures in the system and delays in the communication architecture. When conceiving a protocol that guarantees the consistency in distributed systems, two important elements must be faced: ordering and timeliness [3, 12]. The ordering aspect defines the possible orders in which operations can be executed and perceived by the participant sites, while the timeliness defines how soon the effects of a operation in some process are known by the other processes.

Many distributed applications are built over distributed systems that use shared information through caching and replication of data. Objects are generally cached at user sites to enhance reliability or to improve performance. Thus, one of the major problems is keeping all the copies of the same object consistent. This means, for instance, that if one user update a copy of some object, we need to ensure that the other copies are consistent. However, many users could be changing objects concurrently and these operations induce some causal relationships among them. *Timed consistency* [12] is a model where both issues, ordering and timeliness, are addressed. These two elements can be set independently, giving different arrangements for different requirements. This paper presents this model as a unifying vision of most of consistency protocols in distributed systems.

The idea behind the unification of consistency protocols consists in locating each protocol in some point for the different axes: time and order. Consider, for example, two popular consitency protocols: *linearizability*([5]) and *sequential consistency*([8]). Intuitively, linearizability requires that a **read** operation returns the value written by the *last* preceeding **write** event in real-time occurrence order. On the other side, sequential consistency states that a multiprocess program exectues correctly if its results could have been produced by executing that program on a single processor system. *Timed consistency*([12]) requires that Δ time units after some **write** is produced its effects should be visible for all the rest of sites in the distributed system. Timed consistency unify the former two protocols by changing the value for Δ from $\Delta = 0$ for linearizability and $\Delta = \infty$ for sequential consistency.

Some fundamental concepts of consistency models in distributed systems are reviewed in Section 2. The timed consistency model for distributed objects is presented in Section 3. In Section 4, we explore the timed consistency approach as a unifying model for different protocols. The paper is concluded in Section 5.

2 Consistency revisited

A distributed system consists of N user processes and a distributed data storage. Because of caching and replication, several, possibly different, copies of the same data objects might coexist at different sites of the system. Thus, a consistency model, understood as a contract between processes and the data storage, must be provided. There are multiple consistency models [1–3, 5, 8, 11, 12].

The *global history* $\mathcal{H}$ of a distributed system is the partially ordered set of all operations occurring at all sites of the system. $\mathcal{H}_i$ is the sequence of operations that are executed on site i and it is a totally ordered set. If **a** occurs before **b** in $\mathcal{H}_i$ we say that **a** precedes **b** in *program order*. In order to simplify, we assume that all operations are either **read** or **write**, and that each value written is unique. These operations take a finite, non-zero time to execute, so there is a time elapsed from the instant when a **read** or **write** "starts" to the moment when such operation "finishes". Nevertheless, for the purposes of this paper, we associate an instant to each operation, called the *effective time* of the operation. We will say that **a** is *executed* at time t if the effective time of **a** is t.

If $\mathcal{D}$ is a set of operations, then a *serialization* of $\mathcal{D}$ is a linear sequence S containing exactly all the operations of $\mathcal{D}$ such that each **read** operation to a particular object returns the value written by the most recent (in the order of S) **write** operation to the same object. If $\prec$ is an arbitrary partially ordered relation over $\mathcal{D}$, we say that serialization S *respects* $\prec$ if $\forall$ **a**, **b** $\in \mathcal{D}$ such that **a** $\prec$ **b** then **a** precedes **b** in S.

Intuitively, one would like that any **read** on a data item `X` returns a value corresponding to the results of the most recent **write** on `X`. In a groupware editing system this could mean that any change to some section of the document must be seen for every other user as soon as it is required. Assuming the existence of

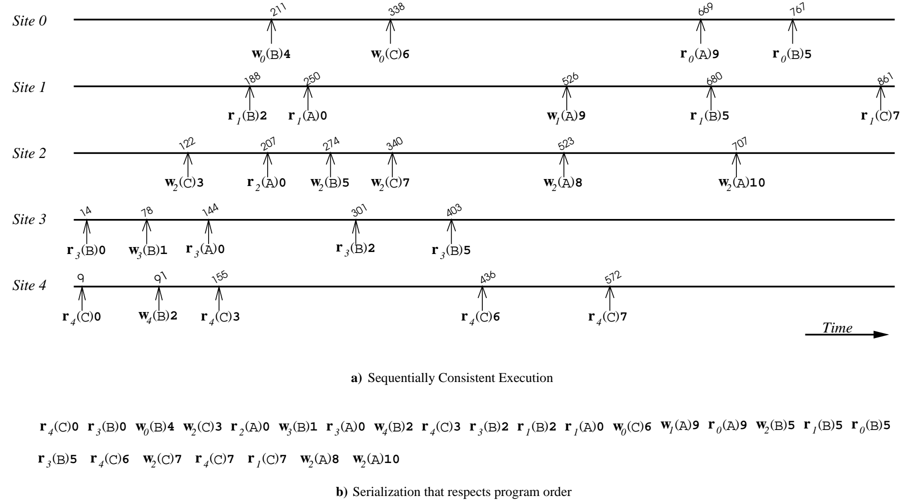
absolute global time, this behavior can be modeled with *linearizability* [5] (also called *atomic consistency*, [9]):

Definition 1. *History $\mathcal{H}$ satisfies* linearizability (**LIN**) *if there is a serialization S of $\mathcal{H}$ that respects the order induced by the effective times of the operations.*

A weaker, but more efficient, model of consistency is *sequential consistency* as defined by Lamport in [8]:

Definition 2. *History $\mathcal{H}$ satisfies* sequential consistency (**SC**) *if there is a serialization S of $\mathcal{H}$ that respects the program order for every site in the system.*

SC does not guarantee that a **read** operation returns the most recent value with respect to real-time, but just that the result of any execution is the same as if the operations of all sites were executed in some sequential order, and the operations of each individual site appear in this sequence in the order specified by its program. For instance, History $\mathcal{H}$ presented in 1.**a)** is sequentially consistent, because 1.**b)** shows the required serialization S. Although **SC** can be implemented in a more efficient way than **LIN** and it is a programmer-friendly model, it has been shown that **SC** has performance problems [1, 11]. An efficiency-based comparison between *SC* and *LIN* has been made in [4]. On the other side, a *SC* induction protocol from a *LIN* perspective appears in [10].



a) Sequentially Consistent Execution

$\mathbf{r}_4(C)0$ $\mathbf{r}_3(B)0$ $\mathbf{w}_0(B)4$ $\mathbf{w}_2(C)3$ $\mathbf{r}_2(A)0$ $\mathbf{w}_3(B)1$ $\mathbf{r}_3(A)0$ $\mathbf{w}_4(B)2$ $\mathbf{r}_4(C)3$ $\mathbf{r}_3(B)2$ $\mathbf{r}_1(B)2$ $\mathbf{r}_1(A)0$ $\mathbf{w}_0(C)6$ $\mathbf{w}_1(A)9$ $\mathbf{r}_0(A)9$ $\mathbf{w}_2(B)5$ $\mathbf{r}_1(B)5$ $\mathbf{r}_0(B)5$
$\mathbf{r}_3(B)5$ $\mathbf{r}_4(C)6$ $\mathbf{w}_2(C)7$ $\mathbf{r}_4(C)7$ $\mathbf{r}_1(C)7$ $\mathbf{w}_2(A)8$ $\mathbf{w}_2(A)10$

b) Serialization that respects program order

Fig. 1. Distributed history compliant with sequential consistency

An even weaker model of consistency is *causal consistency* [2]. First, let $\mathcal{H}_{i+w}$ be the set of all the operations in $\mathcal{H}_i$ plus all the **write** operations in $\mathcal{H}$. Second,

we modify the *happens-before* relationship "→" for message passing systems as defined in [7] to order the operations of $\mathcal{H}$. Let **a**,**b** and **c** $\in \mathcal{H}$, we say that **a** → **b**, i.e., **a** happens-before (or *causally precedes*) **b**, if one of the following holds:

1. **a** and **b** are executed on the same site and **a** is executed before **b**.
2. **b** reads an object value written by **a**.
3. **a** → **c** and **c** → **b**.

Two distinct operations **a** and **b** are *concurrent* if none of these conditions hold between them.

Definition 3. *History $\mathcal{H}$ satisfies* causal consistency (**CC**) *if for each site i there is a serialization S_i of the set $\mathcal{H}_{i+w}$ that respects causal order "→".*

Thus, if **a**,**b** and **c** $\in \mathcal{H}$ are such that **a** writes value **v** in object `X`, **c** reads the same value **v** from object `X`, and **b** writes value **v'** into object `X`, it is never the case that **a** → **b** → **c**. **CC** requires that all causally related operations be seen in the same order by all sites, while different sites could perceive concurrent operations in different orders. **CC** is a model of consistency weaker than **SC**, but it can be implemented efficiently [2, 11].

3 Timed Consistency Model

In neither **SC** nor **CC** real-time is explicitly captured, i.e., in the serializations of $\mathcal{H}$ or $\mathcal{H}_{i+w}$ operations may appear out of order in relation to their effective times. For instance, in Figure 1 the serialization in part **b)** shows event **r**$_1$(`B`)**5** occurring before **w**$_2$(`A`)**8**, but the latter event occurred at time 523, while the former occurred at time 680. In **CC**, each site can see concurrent **write** operations in different orders. On the other hand, **LIN** requires that the operations be observed in their real-time ordering. Ordering and time are two different aspects of consistency. One avoids conflicts between operations, the other addresses how quickly the effects of an operation are perceived by the rest of the system.

Timed consistency (**TC**) as proposed in [12] requires that if the effective time of a **write** is t, the value written by this operation must be visible to all sites in the distributed system by time $t + \Delta$, where Δ is a parameter of the execution. It can be seen that when $\Delta = 0$, then **TC** becomes **LIN**. So, **TC** can be considered as a generalization or weakening of **LIN**.

The execution showed in Figure 2 satisfies **SC** and **CC**. Up to the second operation of Site *1*, the execution satisfies **TC** for the value of Δ presented in this figure, but, by that same instant, **LIN** is no longer satisfied. After this point, the execution is not even timed because there are **read** operations in Site *1* that start more than Δ units of real-time after Site *0* writes the value **7** into object `X` and these **read** operations do not return this value.

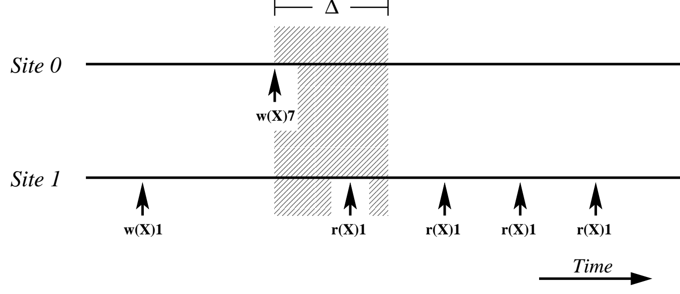


Fig. 2. A non-timed sequentially consistent execution

3.1 Reading on Time

In *timed* models, the set of values that a **read** may return is restricted by the amount of time that has elapsed since the preceding **writes**. A **read** occurs *on time* if it does not return stale values when there are more recent values that have been available for more than Δ units of time. This definition depends on the properties of the underlying clock used to assign timestamps to the operations in the execution. Let $T(\mathbf{a})$ be the real-time instant corresponding to the effective time of operation **a**.

Definition 4. *Let $\mathcal{D} \subseteq \mathcal{H}$ be a set of operations and S a serialization of $\mathcal{D}$. Let* **w**, **r** $\in \mathcal{D}$ *be such that* **w** *writes a value into object* `X` *that is later read by* **r**, *i.e.,* **w** *is the closest* **write** *operation into object* `X` *that appears to the left of* **r** *in serialization S. We define the set $\mathcal{W}_{\mathbf{r}}$, associated with* **r**, *as: $\mathcal{W}_{\mathbf{r}}$=\{**w'** $\in \mathcal{D}$ | (**w'** writes a value into object* `X`*) $\wedge$ ($T(\mathbf{w}) < T(\mathbf{w'}) < T(\mathbf{r})$ - Δ)\}. We say that operation* **r** ***occurs*** *or* ***reads on time*** *in serialization S, if $\mathcal{W}_{\mathbf{r}} = \emptyset$. S is* **timed** *if every* **read** *operation in S occurs on time.*

Figure 3 illustrates Definition 4, presenting a possible arrangement of **read** and **write** operations over the same object. Operation **r** reads a value previously written by operation **w**. Since operation $\mathbf{w}_1$ was executed before **w**, it has no effect on whether **r** is reading on time or not. Similarly, although $\mathbf{w}_4$ is more recent than **w**, the interval Δ has not elapsed yet when **r** is executed, and, thus, it is acceptable that **r** does not observe the value written by $\mathbf{w}_4$. On the other hand, operations $\mathbf{w}_2$ and $\mathbf{w}_3$ occur after **w**, and the values written by then have been available in the system for more than Δ units of time when **r** is executed. Thus, $\mathbf{w}_2$ and $\mathbf{w}_3$ are in $\mathcal{W}_{\boldsymbol{r}}$, and, therefore, operation **r** does not occur on time. The area between $T(\mathbf{w})$ and $T(\mathbf{r})$-Δ represents the interval of time associated with the set $\mathcal{W}_{\boldsymbol{r}}$, which according to definition 4 must be empty if **r** reads on time (i.e. no write operation to the same object read by **r** can occur in this interval).

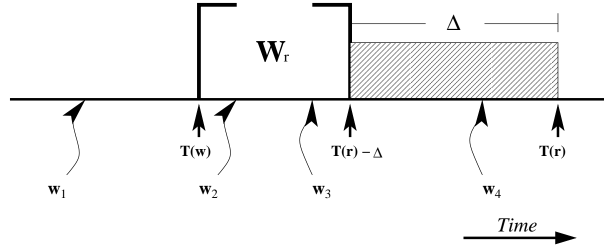


Fig. 3. Operation **r** does not read on time

3.2 Timed Sequential Consistency and Timed Causal Consistency

Now, we combine the requirements of well-known consistency models such as **SC** and **CC** with the requirement of reading on time.

Definition 5. *History* $\mathcal{H}$ *satisfies* timed sequential consistency (**TSC**) *if there is a* ***timed*** *serialization* S *of* $\mathcal{H}$ *that respects the program order for each site in the system.*

Definition 6. *History* $\mathcal{H}$ *satisfies* timed causal consistency (**TCC**) *if for each site* i *there is a* ***timed*** *serialization* S_i *of* $\mathcal{H}_{i+w}$ *that respects causal order "*$\rightarrow$*".*

Figure 4 presents a hierarchy of different consistency models. Every sequentially consistent execution is also linearizable, but the opposite is not necessarily true. Similarly, every causally consistent execution is sequentially consistent, while the contrary is not always true. The proofs for these results and some implementation details can be found in [12]. The idea behind these definitions is to show how it is possible to offer flexibility for the consistency protocol in our model; without losing the real-time considerations, consistency aspects can be relaxed, passing from **SC** to **CC**.

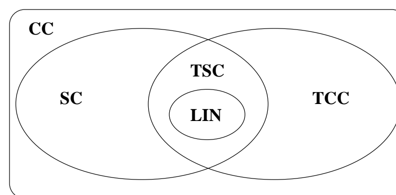


Fig. 4. Hierarchy of consistency models

4 Unifying Model

Timed consistency offers the flexibility of changing two aspects for characterizing a consistency protocol for distributed systems. This feature permits many protocols to be included inside the timed consistency framework. It can be analyzed the effect of modifying the parameters in each of these dimensions.

Firstly, the order dimension can be set according to the different set of distributed histories the system is able to admit. So, this parameter can be set to be *LIN*, *SC*, *CC* or any other ([6]). In figure 5 it is presented different values for the order dimension. We can see that linearizability is the most restrictive order value. Then sequential consistency is more weaker, but still harder than *TSO* (*total store ordering*, [6]). *CC* is still weaker than *TSO* as well as *PC*(*processor consistency*,[6]). However, there is no inclusion relation between *CC* and *PC*. The weakest order protocol is *PRAM*(*pipelined RAM*,[6]).

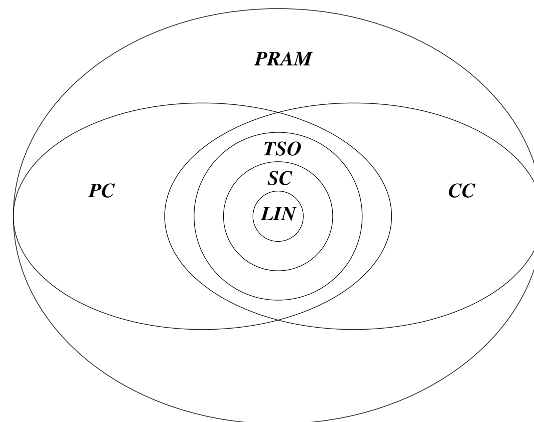


Fig. 5. Different order protocols

On the other side, the time dimension is dominated by the value of Δ. Setting different values for this parameter will produce different protocols. In figure 6 it can be seen the effects of changing the value for Δ. Firstly, for a zero value of Δ it is obtained linearizability. This is clear, as *LIN* requires that any effect of some **write** operation is seen immediately in the rest of sites of the system. Any other value for Δ will produce a different *TSC* protocol.

However, the assignment $\Delta = \infty$ will produce *SC* as the effects of some **write** are not needed to be perceived by the rest of sites even for a large value of Δ. So, it can be assumed that the time elapsed since the execution of some event to the receipt of that change could be infinity. This last observation is related with some *convergence* notion explored in [13]. Intuitively, *SC* doesn't offer convergence in the sense that *SC* doesn't guarantee that at some specified

time t all processor will agree in the value for some distributed shared object. Such time t cannot be stablished by *SC*.

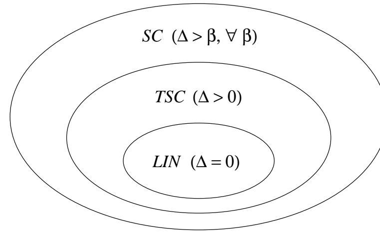


Fig. 6. Varying the value of Δ

5 Conclusions

In this paper, we have presented a consistency model from the distributed computing perspective to be considered as unifying principle for consistency protocols in distributed systems.

Timed consistency models examine interesting temporal relationships between objects and sites that form a distributed system, and are able to capture requirements that are not easily expressed by standard consistency models such as **SC** and **CC**.

A timed consistency model defines a maximum acceptable threshold of time (i.e., parameter Δ) after which the effects of a **write** operation must be available to the entire distributed system. By combining the requirements of timed consistency and those of a consistency criteria such as **SC** and **CC**, we propose **TSC** and **TCC**.

The value of Δ is the result of a trade-off between the need of perceiving changes to shared objects in a timely fashion and the availability of resources in the system. Small values of Δ require more communication overhead and may decrease the scalability of the system (e.g., in extreme cases, local caches become useless), while large values of Δ require less expensive methods but reduce the timeliness of the information and the actual sharing of information by the sites.

References

1. Adve, S. and Gharachorloo, K. *Shared Memory Consistency Models: A Tutorial*. Western Research Laboratory, Research Report 95/7, 1995.
2. Ahamad, M. et al. *Causal memory: definitions, implementation and programming.* Distributed Computing. September, 1995.
3. Ahamad, M. and Raynal, M. *Ordering and Timeliness: Two Facets of Consistency?*, Future Directions in Distributed Computing, 2003.

4. Attiya H. and Welch J.L.. *Sequential Consistency versus Linearizability*. ACM TOCS, 1994.
5. Herlihy, M. and Wing, J. *Linearizability: A Correctness Condition for Concurrent Objects*. ACM Transactions on Programming Languages and Systems. Vol 12(3), July 1990.
6. Kohli, Prince, Neiger, G. and Ahamad, M. *A Characterization of Scalable Shared Memories*. Technical Report GIT-CC-93/04, College of Computing, Georgia Institute of Technology, 1993.
7. Lamport, L. *Time, Clocks and the Ordering of Events is a Distributed System*. Communications of the ACM, vol 21, July, 1978.
8. Lamport, L. *How to make a Multiprocessor Computer that correctly executes Multiprocess Programs*. IEEE Transactions on Computer Systems, C-28(9), 1979.
9. Misra J. *Axioms for Memory Access in Asynchronous Hardware Systems*. ACM TOPLAS, 8(1), 1986.
10. Raynal, M. *Sequential Consistency as Lazy Linearizability*. SPAA'02, Winnipeg, Manitoba, Canada, 2002.
11. Torres-Rojas, F. J., Ahamad, M. and Raynal, M. *Lifetime Based Consistency Protocols for Distributed Objects*. Proc. 12th International Symposium on Distributed Computing, DISC'98, Andros, Greece, September 1998.
12. Torres-Rojas, F. J., Ahamad, M. and Raynal, M. *Timed Consistency for Shared Distributed Objects*. Annual ACM Symposium on Principles of Distributed Computing PODC'99, Atlanta, Georgia, 1999.
13. Torres-Rojas, F. J. and Meneses, E. *Analyzing Convergence in Consistency Protocols for Distributed Systems*. Technical Report. Center for Research in Computing, Costa Rica Institute of Technology, 2004.