From Session Causality to Causal Consistency[*]

Jerzy Brzeziński
Jerzy.Brzeziński@put.poznan.pl

Cezary Sobaniec
Cezary.Sobaniec@cs.put.poznan.pl

Dariusz Wawrzyniak
Dariusz.Wawrzyniak@cs.put.poznan.pl

Institute of Computing Science
Poznań University of Technology
Piotrowo 3a, 60-965 Poznań
Poland

Abstract

In this paper we discuss relationships between client-centric consistency models (known as session guarantees), and data-centric consistency models. The first group includes: read-your-writes guarantee, monotonic-writes guarantee, monotonic-reads guarantee and writes-follow-reads guarantee. The other group includes: atomic consistency, sequential consistency, causal consistency, processor consistency, PRAM consistency, weak consistency, release consistency, scope consistency and entry consistency. We use a consistent notation to present formal definitions of both kinds of consistency models in the context of replicated shared objects. Next, we prove a relationship between causal consistency model and client-centric consistency models. Apparently, causal consistency is similar to writes-follow-reads guarantee. We show that in fact causal consistency requires all common session guarantees, i.e. read-your-writes, monotonic-writes, monotonic-reads and writes-follow-reads to be preserved.

Keywords: *shared objects, remote objects, consistency models, session guarantees*

1. Introduction

Consistency management is a key function of a system supporting highly available data storage by means of replication. Depending on the data usage scenario, applications may have different requirements concerning consistency. Computational algorithms usually require *strong* consistency which tries to follow behaviour of a multiprocessor with shared memory. Other applications like DNS accept *week* consistency, tolerating transient inconsistencies between servers. Replication management aimed at keeping replicas consistent is implemented by the use of a *consistency protocol*. Guarantees provided by the consistency protocol are expressed by a *consistency model* defining the circumstances under which a given replica becomes updated. Therefore, the consistency model is a contract between the application developer that accesses the data, and the system that provides the data.

There are several consistency models that try to find the best trade-off between protocol efficiency and ease of use. In general, existing models may be divided into two groups: *data-centric* and *client-centric* [9]. Data-centric models specify the restrictions imposed on the order in which data is updated on individual servers. There are several data-centric consistency models defined mainly as a result of research in the field of *Distributed Shared Memory* (DSM), e.g.: *atomic consistency* [8], *sequential consistency* [6], *causal consistency* [2], *processor consistency* [4, 1], *PRAM consistency* [7]. The most restrictive data-centric models, such as atomic consistency or sequential consistency, ensure the behaviour of the system as if there was no replication. These models are intuitive for the programmer, but unfortunately their implementations (protocols) tend to be inefficient. Less restrictive consistency models (*causal*, *processor*, *PRAM*), due to consistency relaxation, can be more efficiently implemented, but either they are too weak for a given application, or it is too difficult to prove their applicability.

Client-centric consistency models specify the requirements concerning data consistency that are based only on the history of interaction between individual clients and the system (servers). Client-centric consistency models seem to be more intuitive for the applica-

[*]This work was supported in part by the State Committee for Scientific Research (KBN), Poland, under grant KBN 7 T11C 036 21

tion developer. From the developer point of view, data-centric models are often inappropriate, because he or she prefers to express consistency requirements for a single process, rather than to bother with details of global replication management. Another important distinction between data- and client-centric consistency models lies in separation of clients and servers. In data-centric models it is assumed that the client is bound to a server and forms a single processing node in the sense that everything known to the server is also known to the client. Client-centric consistency models allow issuing different operations at different servers.

Both consistency model families relate to the same problem: consistency maintenance. For this reason a simple intuition says that there should be a way of describing a model from one family by means of consistency conditions defined for another model family, or at least to establish some relationships between client- and data-centric models. Such relationships may be very helpful in understanding consistency maintenance problem in general, and in particular it may help application developer to choose the appropriate model.

In this paper we use a consistent notation to present formal definitions of both consistency model families. Next, we prove a relationship between causal consistency model and client-centric consistency models. Apparently, causal consistency is similar to writes-follow-reads guarantee (also called session causality [3]) from the family of client-centric models. We show that in fact causal consistency requires all session guarantees defined in [10], i.e. read-your-writes, monotonic-writes, monotonic-reads and writes-follow-reads to be preserved.

2. Client-Server Interaction via Replicated Remote Objects

Objects are instances of abstract data types that encapsulate some private instance variables accessed by public methods. Operation on an object means execution of one of its methods. An operation is executed as a result of method invocation issued by a client. Every client is a sequential process in the sense that it invokes a method after the execution of the previously invoked method is completed. In other words, from the viewpoint of a given client, at most one operation on one object is pending at a time.

Objects exist on servers, so the operations on objects are executed by servers (see Figure 1). Consequently, there is an interaction between a server and a client during the operation performance, which proceeds as follows:

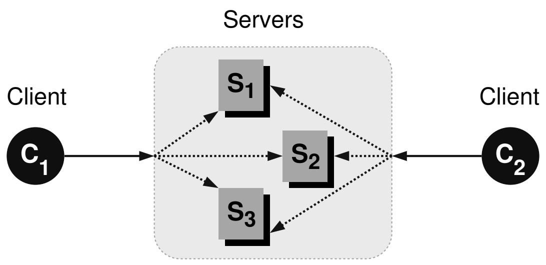

Figure 1: Remote objects method invocations

1. as a result of an invocation the client sends a request message to the server and blocks until results of the execution are received,
2. the server receives the message, executes the requested method and sends back a response with the results of execution,
3. the client receives the response and continues the execution of its program (possibly invokes another method).

As for the methods themselves, they consist of a number of *elementary* read and write operations on instance variables (of a simple data type) encapsulated within objects. Thus the execution of a method can be reduced to a sequence of read and write operations. If a client invokes a method, it indirectly issues the execution of some elementary operations that implement the method. These elementary operations are said to be *issued by the client* although the client may not be aware of what operations are included in the implementation.

Operations issued by a given client, say C_i, are totally ordered. Let O_{C_i} denote the set of operations issued be the client. The issue order of operations is defined as follows:

DEFINITION 1. *Let $o1$ and $o2$ be operations issued by a client C_i, i.e. $o1 \in O_{C_i}$ and $o2 \in O_{C_i}$. Operation $o1$ precedes $o2$ in issue order ($o1 \overset{C_i}{\rightarrow} o2$) if one of the following conditions holds:*

1) $o1$ and $o2$ belong to the implementation of the same method and $o1$ precedes $o2$ in the program order (thereby $o1$ is performed at the server side before $o2$), or

2) $o1$ and $o2$ belong to the implementations of different methods, say $m1$ and $m2$, respectively, and $m1$ is issued by the client C_i before $m2$. □

To execute a method, a client may send the request to any server keeping a replica of the referenced object. When the invoked method changes the state of the object, the servers are obliged to interact between one another to achieve a consistent state of object replicas. Consequently, subsequent elementary write operations are to be executed by each server that keeps any replica of the object. In order to simplify the system model it is assumed that each server keeps a replica of every shared object available to the clients. Therefore, write operations must be performed by each server.

The order in which the elementary operations are executed by the servers or observed by the clients is a question of the consistency model. For the sake of formal specification, let *OW* denote a set of all elementary write operations on shared objects and let O_{S_i} denote a set of operations executed locally by a server S_i as a result of direct invocations of methods by clients. In order to distinguish different replicas of the same instance variable let x_i mean a replica of x kept by the server S_i. The operations themselves are denoted as follows:

$w_i(x_j)v$ — write operation of a value v issued by C_i, performed on the replica of x kept by S_j,

$r_i(x_j)v$ — read operation on the replica of x kept by S_j, issued by C_i, returning a value v,

$o_i(x_j)v$ — any operation on the replica of x kept by S_j, issued by C_i, concerning a value v,

The client index, replica index, or variable value will be omitted when it is either not important or results from the current context.

Definition 2 (Server's view). *For a given server S_i the view of elementary operations on shared objects is the set $OW \cup O_{S_i}$ totally ordered by a relation $\overset{S_i}{\rightarrowtail}$ that satisfies the following condition, called legality*[2]*:*

$$\forall_{\substack{w(x)v \in OW \cup O_{S_i} \\ r(x)v \in OW \cup O_{S_i}}} \left[w(x)v \overset{S_i}{\rightarrowtail} r(x)v \wedge \nexists_{w(x)u \in OW \cup O_{S_i}} \left(u \neq v \wedge w(x)v \overset{S_i}{\rightarrowtail} w(x)u \overset{S_i}{\rightarrowtail} r(x)v \right) \right] \tag{1}$$

□

Let us call the relation $\overset{S_i}{\rightarrowtail}$ *server's view order*.

Because the clients can invoke each method on a different server, it is possible that a server's view of operation execution is different from the client's view. In general, there is no guarantee that an overwritten value cannot be read by a client, because the client can access an object on a server whose state has not been updated yet. Additional guarantees depend on a specific consistency model. Consequently, the definition of client's view does not include the legality condition in the form of the condition (1).

[2] To simplify the identification of write operations, it is assumed that each write operation defines a unique value of a given instance variable.

Definition 3 (Client's view). *For a given client, say C_i, the view of elementary operations on shared objects is the set $OW \cup O_{C_i}$ totally ordered by a relation $\overset{C_i}{\rightarrowtail}$ that satisfies the following condition*[2]*:*

$$\forall_{\substack{w(x)v \in OW \cup O_{C_i} \\ r(x)v \in OW \cup O_{C_i}}} w(x)v \overset{C_i}{\rightarrowtail} r(x)v \tag{2}$$

□

Similarly to server's view order, let us call the relation $\overset{C_i}{\rightarrowtail}$ *clients's view order*.

Consistency models impose additional restrictions on the relation according to which operations are ordered in the view of each processes. The restrictions are specified in the form of consistency conditions that must be preserved in the view order.

When a client issues a read operation, it expects to get the result before issuing another operation. Consequently, we assume as an axiom the following condition:

$$\forall_{\substack{r(x)v \in O_{C_i} \\ o(y)u \in O_{C_i}}} \left(r(x_j)v \overset{C_i}{\rightharpoondown} o(y_j)u \Rightarrow r(x)v \overset{S_j}{\rightarrowtail} o(y)u \right) \tag{3}$$

Similarly, for the client's view:

$$\forall_{\substack{r(x)v \in O_{C_i} \\ o(y)u \in O_{C_i}}} \left(r(x)v \overset{C_i}{\rightharpoondown} o(y)u \Rightarrow r(x)v \overset{C_i}{\rightarrowtail} o(y)u \right) \tag{4}$$

3. Client-centric Consistency Models

The idea of client-centric consistency models has been proposed in [10], and called *session guarantees*. A formal specification from the viewpoint of a client process is presented in [3]. Kermarrec et al. [5] analyse the application of these models to the replication of Web documents.

In the following subsections we formally define the models from the viewpoint of client-server interaction, assuming that one client process corresponds to one session in terms of the definitions in [10].

Generally, in the client-centric consistency models it is assumed that non-commutative write operations are observed by every server in the same order. We assume in this case that non-commutative writes are the write operations that concern the same variable. This means formally that:

$$\underset{x}{\forall}\left(\underset{S_i}{\forall} w(x)v \overset{S_i}{\rightarrowtail} w(x)u \vee \underset{S_i}{\forall} w(x)u \overset{S_i}{\rightarrowtail} w(x)v\right) \tag{5}$$

3.1. Read Your Writes

Read your writes (RYW) guarantee states that a read operation requested by a client can be executed on the server that has performed all write operations previously issued by the requesting client. This is formally expressed by the following condition:

$$\underset{C_i}{\forall}\underset{S_j}{\forall}\left(w(x)v \overset{C_i}{\rightharpoondown} r(y_j)u \Rightarrow w(x)v \overset{S_j}{\rightarrowtail} r(y)u\right) \tag{6}$$

3.2. Monotonic Writes

Monotonic writes (MW) guarantee states that write operations are observed by the servers in issue order. In other words, when a write operation issued by a given client is performed by a server, all write operations issued previously by the client must have been performed by the server before. Consequently, the consistency condition is formulated as follows:

$$\underset{C_i}{\exists}\, w(x)v \overset{C_i}{\rightharpoondown} w(y)u \Rightarrow \underset{S_j}{\forall}\, w(x)v \overset{S_j}{\rightarrowtail} w(y)u \tag{7}$$

3.3. Writes Follow Reads

Writes follow reads (WFR) guarantee has been also called *session causality* in [3]. This model can be informally defined as follows: a write operation by a client process on data item y following a previous read operation on x by the same process, is guaranteed to follow write operation on x of the value that was read. In other words, causal precedence established by one client must be preserved also by all servers (in their views).

A formal definition states that:

$$\underset{C_i}{\exists}\, r(x)v \overset{C_i}{\rightharpoondown} w(y)u \Rightarrow \underset{S_j}{\forall}\, w(x)v \overset{S_j}{\rightarrowtail} w(y)u \tag{8}$$

3.4. Monotonic Reads

Monotonic reads (MR) requires that read operation can be executed by the server that has executed all write operations whose effect has been already read by the client.

Formally, this requires the following condition to be satisfied:

$$\underset{C_i}{\forall}\underset{S_j}{\forall}\left(r(x)v \overset{C_i}{\rightharpoondown} r(y_j)u \Rightarrow w(x)v \overset{S_j}{\rightarrowtail} r(y)u\right) \tag{9}$$

4. Session guarantees for Causal Consistency

From the viewpoint of client-server application data-centric consistency models can be specified both for the client side and for the server side. The distinction between client side and server side results from the fact that a client process can interact with several servers within its session. If it uses the same server all the time the distinction makes no sense, because the server's view and the client's view are the same.

Data-centric consistency models require legality, which is not the case of client-centric models at the client side. Consequently, the legality condition must also be preserved in client's view, i.e:

$$\underset{\substack{w(x)v\in OW\cup O_{C_i}\\ r(x)v\in OW\cup O_{C_i}}}{\forall}\left[w(x)v \overset{C_i}{\rightarrowtail} r(x)v \wedge \underset{w(x)u\in OW\cup O_{C_i}}{\nexists}\left(u\neq v \wedge w(x)v \overset{C_i}{\rightarrowtail} w(x)u \overset{C_i}{\rightarrowtail} r(x)v\right)\right] \tag{10}$$

Causal consistency [2] requires causal order of operations to be preserved in the views of operations. The causal order relation is defined as follows:

Definition 4 (Causal order). *An operation, say* $o1$*, causally precedes another operation, say* $o2$ *(*$o1 \rightsquigarrow o2$*), if one of the following conditions holds:*

$$(i)\quad \underset{C_i}{\exists}\, o1 \overset{C_i}{\rightharpoondown} o2 \tag{11}$$

$$(ii)\quad o1 = w(x)v \wedge o2 = r(x)v \tag{12}$$

$$(iii)\quad \underset{o\in O}{\exists}\,(o1 \rightsquigarrow o \wedge o \rightsquigarrow o2) \tag{13}$$

□

4.1. Server side

At the server side, causal consistency means that the following condition is preserved:

$$\underset{S_i}{\forall}\ \underset{o1,o2\in OW\cup O_{S_i}}{\forall}\left(o1 \rightsquigarrow o2 \Rightarrow o1 \overset{S_i}{\rightarrowtail} o2\right) \tag{14}$$

Theorem 1. *At the server side, causal consistency is fulfilled if and only if RYW (6), MW (7), WFR (8) and MR (9) conditions are preserved.* □

Proof. Let us consider 2 causally related operations $o1$ and $o2$ ($o1 \rightsquigarrow o2$) executed by a given server, say S_i. According to the definition 4 (causal order) the following 3 cases can be distinguished:

(i) Both $o1$ and $o2$ are issued by the same client. Let us further decompose this case into 4 points:

1) $o1 = r$ and $o2 = r$ — following the axiom (3), $o1 \overset{S_i}{\rightarrowtail} o2$,
2) $o1 = r$ and $o2 = w$ — following the axiom (3), $o1 \overset{S_i}{\rightarrowtail} o2$,
3) $o1 = w$ and $o2 = r$ — following the condition RYW (6), $o1 \overset{S_i}{\rightarrowtail} o2$,
4) $o1 = w$ and $o2 = w$ — following the condition MW (7), $o1 \overset{S_i}{\rightarrowtail} o2$.

Thus, local order is preserved in the view of each server.

(ii) $o1$ is a write operation and $o2$ is a read operation that returns the value written by $o1$. Since both operations are executed by the same server then obviously read operation must follow write operation to return the value written by this write.

(iii) Let us assume that $o1$ and $o2$ are issued by different clients. If the case (ii) is excluded, there must be another operation o causally ordered between $o1$ and $o2$ ($o1 \rightsquigarrow o \rightsquigarrow o2$). The proof of this case will be provided by means of induction.

Assumption

$$\forall_{S_j} \left(o1, o \in O_{S_j} \Rightarrow o1 \overset{S_j}{\rightarrowtail} o \right) \wedge \forall_{S_i} \left(o, o2 \in O_{S_i} \Rightarrow o \overset{S_i}{\rightarrowtail} o2 \right) \quad (15)$$

Proposition

$$o1 \overset{S_i}{\rightarrowtail} o2 \quad (16)$$

Proof for the initial case. Two initial cases should be considered:

1) $o1 = w(x)v$. In the simplest case this means that $\exists_{C_j} r(x)v \overset{C_j}{\rightharpoonup} o2$. If $o2$ is also a write operation, then according to WFR guarantee (8) $o1$ must appear before $o2$ in the view of each server. If $o2$ is a read operation, then according to MR guarantee (9) $o1$ must appear before $o2$ in the view of each server that observes these operations.
2) $o2 = r(x)v$. In the simplest case this means that $\exists_{C_j} o1 \overset{C_j}{\rightharpoonup} w(x)v$. $w(x)v$ is observed by each server, including S_i. According to point (i) of this proof $o1$ appears before $w(x)v$, and according to point (ii) $o2$ must follow $w(x)v$. Consequently $o1 \overset{S_i}{\rightarrowtail} o2$.

Proof for the general case. Let us assume that $o2$ is issued by C_j. Causal relation between $o1$ and $o2$ means that $\exists_{w(y)t, r(y)t} o1 \rightsquigarrow w(y)t \wedge r(y)t \overset{C_j}{\rightharpoonup} o2$, and from the assumption $o1 \overset{S_i}{\rightarrowtail} w(y)t$. Two kinds of $o2$ will be distinguished:

1) $o2 = w(x)v$. This means that according to WFR guarantee (8) $w(y)t \overset{S_i}{\rightarrowtail} o2 \Rightarrow o1 \overset{S_i}{\rightarrowtail} o2$.
2) $o2 = r(x)v$. According to MR guarantee (9) $w(y)t \overset{S_i}{\rightarrowtail} o2 \Rightarrow o1 \overset{S_i}{\rightarrowtail} o2$. □

The following is a proof of opposite implication:

Proof. By contradiction we get the following four false sentences:

1) RYW: $\forall_{S_i} \forall_{o1,o2 \in OW \cup O_{S_i}} \left(o1 \rightsquigarrow o2 \Rightarrow o1 \overset{S_i}{\rightarrowtail} o2 \right) \wedge \exists_{C_j} \exists_{S_i} \left(w(x)v \overset{C_j}{\rightharpoonup} r(x_i)u \wedge w(x)v \overset{S_i}{\not\rightarrowtail} w(x)u \right)$
If we choose $o1 = w(x)v$ and $o2 = r(x_i)u$ (causaly ordered by client C_j) we get $w(x)v \overset{S_i}{\rightarrowtail} r(x)u$. $w(x)u$ must follow $w(x)v$ because of legality requirement, which leads to contradiction in the sentence.

2) MW: $\forall_{S_i} \forall_{o1,o2 \in OW \cup O_{S_i}} \left(o1 \rightsquigarrow o2 \Rightarrow o1 \overset{S_i}{\rightarrowtail} o2 \right) \wedge \exists_{C_j} \exists_{S_i} \left(w(x)v \overset{C_j}{\rightharpoonup} w(y)u \wedge w(x)v \overset{S_i}{\not\rightarrowtail} w(y)u \right)$
Assuming $o1 = w(x)v$ and $o2 = w(y)u$ issued by the same client we get obvious contradiction.

3) WFR: $\forall_{S_i} \forall_{o1,o2 \in OW \cup O_{S_i}} \left(o1 \rightsquigarrow o2 \Rightarrow o1 \overset{S_i}{\rightarrowtail} o2 \right) \wedge \exists_{C_j} \exists_{S_i} \left(r(x)v \overset{C_j}{\rightharpoonup} w(y)u \wedge w(x)v \overset{S_i}{\not\rightarrowtail} w(y)u \right)$
$w(x)v$ must occur before $r(x)v$ on a servers, and causally proceeds the read, so we can choose $o1 = w(x)v$ and $o2 = w(y)u$ which leads to contradiction.

4) MR: $\forall_{S_i} \forall_{o1,o2 \in OW \cup O_{S_i}} \left(o1 \rightsquigarrow o2 \Rightarrow o1 \overset{S_i}{\rightarrowtail} o2 \right) \wedge \exists_{C_j} \exists_{S_i} \left(r(x)v \overset{C_j}{\rightharpoonup} r(y_i)u \wedge w(x)v \overset{S_i}{\not\rightarrowtail} r(y)u \right)$

As in the previous case: $w(x)v$ must occur before $r(x)v$ on servers, and causally proceeds the read, so we can choose $o1 = w(x)v$ and $o2 = r(y)u$ which leads to contradiction. $\square$

4.2. Client side

At the client side, causal consistency means that the following condition is preserved:

$$\underset{C_i}{\forall} \underset{o1,o2 \in OW \cup O_{C_i}}{\forall} \left(o1 \rightsquigarrow o2 \Rightarrow o1 \overset{C_i}{\rightarrowtail} o2 \right) \tag{17}$$

LEMMA 1. *At the client side, causal order of operation issue is preserved if RYW (6), MW (7), WFR (8) and MR (9) conditions hold.* $\square$

Proof. If the operations that appear in the client's view are executed by the same server, the view preserves causal order according Theorem 1. Notice also that two reads appear in the client's view in the issue order according to the condition 4.

Let us assume without loss of generality that $o1$ and $o2$ are non-commutative operations. Let us also assume for the sake of contradiction that for two causally related operations executed by different servers, client's view violates causal order, i.e. $o1 \rightsquigarrow o2$ and $o2 \overset{C_i}{\rightarrowtail} o1$. Let S_j be the server executing $o1$. Consequently, either $o2 \notin O_{S_j}$ or (from Theorem 1) $o1 \overset{S_j}{\rightarrowtail} o2$.

The first case means that $o2$ is a read operation executed by a different server. Consequently, $o1$ is a write operation. Since $o1$ is observed by the client after $o2$, the server has not seen $o1$ when it is executing $o2$. On the other hand, following Theorem 1 the server must already have seen $o1$, which means contradiction.

For the second case it is assumed that $o2$ is a write operation. Since $o2$ is observed by the client before $o1$, to execute $o1$, the client must interact with a server that has not seen the result of $o2$ yet (see example in Figure 2). This in turn means violation of MR guarantee (9). $\square$

To complete the proof that causal consistency is fullfield at the client side, it is necessary to show that the client's view is legal.

LEMMA 2. *If RYW (6), MW (7) and MR (9) guarantees are preserved, client's view is legal, i.e. satisfies the condition (10).* $\square$

Proof. Let us assume by contradiction that the conditions (6), (7) and (9) are preserved and there is a client

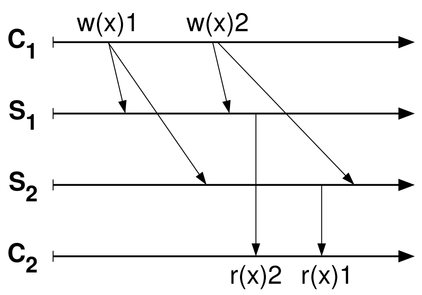

Figure 2: Violation of causal order at the client side

whose view is not legal. This means that for a given client, say C_i, the following condition holds:

$$\underset{\substack{w(x)v \in OW \cup O_{C_i} \\ w(x)u \in OW \cup O_{C_i} \\ r(x_j)v \in OW \cup O_{C_i}}}{\exists} w(x)v \overset{C_i}{\rightarrowtail} w(x)u \overset{C_i}{\rightarrowtail} r(x_j)v \tag{18}$$

Consequently, at the server S_j either $w(x)v \overset{S_j}{\rightarrowtail} r_i(x)v \overset{S_j}{\rightarrowtail} w(x)u$ or $w(x)u \overset{S_j}{\rightarrowtail} w(x)v \overset{S_j}{\rightarrowtail} r_i(x)v$ holds. Let us consider the following four cases:

1) $w(x)v \in O_{C_i}$ and $w(x)u \in O_{C_i}$
Following Lemma 1 and the fact that both client's view order and issue order are linear, the client's view order corresponds to the issue order of these operations, i.e.: $w(x)v \overset{C_i}{\rightharpoondown} w(x)u \overset{C_i}{\rightharpoondown} r(x_j)v$. Consequently, the first case of the S_j's view order violates RYW guarantee (6), and the second case violates MW guarantee (7).

2) $w(x)v \notin O_{C_i}$ and $w(x)u \in O_{C_i}$
The view order defined by the condition (18) is forced by the following issue order: $r(x_k)v \overset{C_i}{\rightharpoondown} w(x)u \overset{C_i}{\rightharpoondown} r(x_j)v$. Consequently, the first case of the S_j's view order violates RYW guarantee (6), and the second case violates the condition (5), because — according to RYW guarantee (6) — $w(x)u \overset{S_j}{\rightarrowtail} r(x)v \Rightarrow w(x)u \overset{S_j}{\rightarrowtail} w(x)v)$, while following the conditions (3) and (6) $w(x)v \overset{S_k}{\rightarrowtail} w(x)u$.

3) $w(x)v \in O_{C_i}$ and $w(x)u \notin O_{C_i}$
The view order defined by the condition (18) is forced by the following issue order: $w(x)v \overset{C_i}{\rightharpoondown} r(x_k)u \overset{C_i}{\rightharpoondown} r(x_j)v$. Consequently, the first case of the S_j's view order violates MR guarantee (9), and the second case violates RYW guarantee (6), because

— according to the condition (5) — $w(x)u \overset{S_k}{\rightarrowtail} w(x)v$.

4) $w(x)v \notin O_{C_i}$ and $w(x)u \notin O_{C_i}$
The view order defined by the condition (18) is forced by the following issue order: $r(x)v \overset{C_i}{\rightharpoondown} r(x_k)u \overset{C_i}{\rightharpoondown} r(x_j)v$. Consequently, the first case of the S_j's view order violates MR guarantee (9), and the second case violates the condition (5) because of MR guarantee (9) at S_k, according to which $r(x)v \overset{C_i}{\rightharpoondown} r(x_k)u \Rightarrow w(x)v \overset{S_k}{\rightarrowtail} r(x)u \Rightarrow w(x)v \overset{S_k}{\rightarrowtail} w(x)u$. □

THEOREM 2. *At the client side, causal consistency is fulfilled if RYW (6), MW (7), WFR (8) and MR (9) conditions are preserved.* □

Proof. As results from Lemma 1, client's view preserves causal order, and, following Lemma 2 it is legal. □

5. Conclusions

In this paper we have presented formal specifications of client-centric and one data-centric consistency models (causal consistency). Proofs have been conducted claiming that causal consistency may be provided as the combination of different session guarantees. This shows that only when the session guarantees are applied together, the system ensures a model from the class of data-centric consistency models. This means that session guarantees provide flexibility of consistency maintenance in comparison to data-centric models.

We differentiate two views of consistency: client view and server view, resulting from the system model which allows clients to switch between servers. To ensure causal consistency at the server side all session guarantees, i.e. *read your writes*, *monotonic writes*, *write follow reads*, and *monotonic reads*, are necessary and sufficient guarantees. The same guarantees are sufficient to achieve causal consistency at the client side.

References

[1] M. Ahamad, R. A. Bazzi, R. John, P. Kohli, and G. Neiger. The power of processor consistency (extended abstract). In *Proc. of the 5th ACM Annual Symp. on Parallel Algorithms and Architectures (SPAA'93)*, pages 251–260, June 1993.

[2] M. Ahamad, G. Neiger, P. Kohli, J. E. Burns, and P. W. Hutto. Casual memory: Definitions, implementation and programming. *Distributed Computing*, 9:37–49, 1995.

[3] G. Chockler, R. Friedman, and R. Vitenberg. Consistency conditions for a CORBA caching service. *Lecture Notes in Computer Science*, 1914:374–388, 2000.

[4] J. R. Goodman. Cache consistency and sequential consistency. Technical Report 61, IEEE Scalable Coherence Interface Working Group, Mar. 1989.

[5] A.-M. Kermarrec, I. Kuz, M. van Steen, and A. S. Tanenbaum. A framework for consistent, replicated Web objects. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS)*, May 1998.

[6] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Comput.*, C-28(9):690–691, Sept. 1979.

[7] R. J. Lipton and J. S. Sandberg. PRAM: A scalable shared memory. Technical Report CS-TR-180-88, Dept. of Computer Science, Princeton University, Sept. 1988.

[8] J. Misra. Axioms for memory access in asynchronous hardware systems. *ACM Trans. Prog. Lang. Syst.*, 8(1):142–153, Jan. 1986.

[9] A. S. Tanenbaum and M. van Steen. *Distributed Systems — Principles and Paradigms*. Prentice Hall, New Jersey, 2002.

[10] D. B. Terry, A. J. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. W. Welch. Session guarantees for weakly consistent replicated data. In *Proceedings of the Third International Conference on Parallel and Distributed Information Systems (PDIS 94), Austin, Texas, September 28-30, 1994*, pages 140–149. IEEE Computer Society, 1994.