Correctness Criteria for Concurrent Systems

Andrew P. Black

Introduction

In a distributed system with replicated resources (files, memory, name-value associations, or whatever), we need to take care that the presence of replication does not lead to system behaviour that is “incorrect”, in the sense that it could not happen were the resource not replicated.

But how do we define “correct” behavior? It turns out that there are several “reasonable” ways to do this, with differing strengths and weaknesses.

The Model

Before we can formally define the consistency criteria, we need to rigorously define the kind of system in which computation takes place.

The system is composed of a set of processes \( P, Q, \ldots \) and a set of shared resources \( x, y, \ldots \). The processes can communicate only through the resources. (How do things change if they can also communicate directly?)

The events of interest in this system are operations on the resources. Without loss of generality, we will assume that each resource is a simple memory location with read and write operations, and the obvious semantics. \( w(x)a \) means that value \( a \) is written to resource \( x \); \( r(x)b \) means that value \( b \) is read from resource \( x \).

These read and write operations are not instantaneous; event \( e \) occupies the time interval between \( \text{start}(e) \) and \( \text{end}(e) \). Thus, even if we did have a global clock, some operations would still be concurrent. In Fig. 1, \( e \) and \( f \) are concurrent, because they overlap in real time.

A number of different orderings on the events in a system are of interest.

Linearizability

The ordering that is of interest for linearizability is the real-time ordering of events in the execution \( \sigma \). In this ordering, \( e \rightarrow f \) if operation \( e \) completes before operation \( f \) starts, that is if \( \text{start}(f) > \text{end}(e) \).

An execution \( \sigma \) is linearizable if

1. There exists a total order \( < \) on the operations in \( \sigma \) such that \( (e \rightarrow f) \Rightarrow (e < f) \). In words: all of the operations that are ordered in real time are also similarly ordered by \( < \).

2. The values read by the read operations correctly model the semantics of the memory assuming that the write and read operation occur in the order specified by \( < \).

In Fig. 2, the partial order that captures the real-time execution order of the operations is
Sequential Consistency

The ordering that is of interest for Sequential Consistency is the sequencing of events in the execution of each process. In this ordering, \( e \rightarrow f \) iff operations \( e \) and \( f \) are both executed by the same process, and \( e \) is executed before \( f \). [Since we disallow parallelism within a process, this is equivalent to saying that \( e \) completes before operation \( f \) starts, that is if \( \text{start}(f) > \text{end}(e) \)]. In other words, the ordering of interest for Sequential Consistency is that used for Linearizability minus the real-time order on events in different processes.

An execution \( \sigma \) is Sequentially Consistent if

1. There exists a total order \( \prec \) on the operations in \( \sigma \) such that \( (e \rightarrow f) \Rightarrow (e \prec f) \). In words: all of the operations that are ordered by being executed by the same process in sequence are also similarly ordered by \( \prec \).
2. The values read by the read operations correctly model the semantics of the memory, assuming that the write and read operation occur in the order specified by \( \prec \).

It should be clear that linearizability is strictly stronger than Sequential Consistency. Both orderings
require that all the participants agree on a legal total order of operations. However, with Sequential Consistency, this order may be different from what actually occurred.

Fig. 3 was not linearizable, but it is Sequentially Consistent. The execution sequence that worked for Fig. 4 can be used for Fig. 3 as well: \( w_1 w_3 w_2 r_1 r_2 \) is a valid execution sequence. \( w_3 \) is serialized before \( r_1 \), but that is ok, because \( w_3 \) and \( r_1 \) are executed by different processors.

---

**Causal Consistency**

Causal Consistency is based on Lamport’s causality rules. Causal order is a partial order defined as follows:

1. If \( e \) and \( f \) are both invoked by process \( P \) and \( e \) occurred before \( f \), then \( e \rightarrow f \).
2. \( w_i(x) \rightarrow r_j(x) \), under the assumption that all values written are distinct. In other words, if a process ever reads the value \( v \) from \( x \), there must have been a preceding write of \( v \) to \( x \).
3. If \( e \rightarrow f \) and \( f \rightarrow g \) then \( e \rightarrow g \).

Clause 2 is the equivalent of the causality rule in a message passing system that says that \( \text{send}(m) \rightarrow \text{receive}(m) \).

Causal consistency does not require that there be a total order consistent with each process’s view of the execution and into which the causal ordering can be embedded. Instead, each process can have its own total order, consistent with causal order, but perhaps different from that of other processes. For example, consider the execution \( \sigma \):

\[
P: \quad w_1(x) a \quad r_1(x) b \\
Q: \quad w_2(x) b \quad r_2(x) a
\]

Each of the read operations is preceded by the write operation in the other process. \( P \) perceives that there must have been a sequence of operations \( w_j(x) a \quad w_2(x) b \quad r_j(x) b \), and \( Q \) perceives that there must have been a sequence of operations \( w_2(x) b \quad w_j(x) a \quad r_j(x) a \). So, \( P \) sees the partial order \( \{ w_1 \rightarrow w_2, w_2 \rightarrow r_1 \} \) and \( Q \) sees the partial order \( \{ w_2 \rightarrow w_j, w_j \rightarrow r_2 \} \).

There is no total order consistent with both of these partial orders, so this example is not sequentially consistent. This means that it could never occur in practice if there were only a single copy of the memory. But it is causally consistent from the individual points of view of all of the participants. Allowing it to occur (when the memory is replicated) reduces the amount of synchronization.

Clearly, causal consistency is weaker than sequential consistency.

---

**Further Reading**

The definitions given here are based on a short conference paper by Raynal and Mizuno[4]. Attiya and Welch undertook a seminal analysis of the differences between Linearizability and Sequential Consistency[3]. Causal consistency was introduced by Ahamed et al. in 1991[1] and explored more fully in 1994[2].

---

**References**

