Notes on fork-join-causal consistency

P. Mahajan, S. Setty, S. Lee, A. Seehra, A. Clement, L. Alvisi, M. Dahlin, and M. Walsh

UTCS TR-10-19
The University of Texas at Austin
fuss@cs.utexas.edu

We express fork join causal (FJC) consistency semantics in terms of a set of conditions that must hold for the observer graph that we associate with each execution of a system.

The observer graph of an execution captures how information flows during the execution: the graph’s vertices represent the read and write operations executed by the nodes, and the edges encode dependencies among these operations. The graph is not an actual data structure that our protocol maintains, but it is useful for presentation purposes.

**Definition 1.** An observer graph is an execution and an edge assignment.

**Definition 2.** An execution is a set of read and write vertices, with one vertex for each read or write operation by any node.

1. **Write vertices** are tuples of the form \((n, s, \text{old}, \text{val})\), where \(n\) is the node issuing the write operation, \(s\) is a per-node sequence number that monotonically increases with every operation issued by \(n\), \(\text{old}\) is the identifier of the object being written, and \(\text{val}\) is the value written to object \(\text{old}\).

2. **Read vertices** are tuples of the form \((n, s, \text{old}, \text{wl})\) where \(n\), \(\text{old}\), and \(s\) define the node issuing the read, the object read, and the sequence number of the operation and where \(\text{wl}\) denotes the list of write vertices whose values are returned by the read. We say a read \(r\) reads from a write \(w\) if \(r.wl\) includes \(w\).

**Definition 3.** An edge assignment for an execution is a set of directed edges connecting vertices of an execution.

An edge assignment is an abstract representation of the data flow in an execution. Notice that the definition does not specify how the edge assignment is produced. A given consistency semantic is defined by a consistency check that determines the set of executions it allows. In particular, showing than an execution is consistent under some semantics simply requires to show that an oracle can produce an edge assignment that passes the consistency check; showing instead that a system enforces some consistency semantics requires presenting an algorithm that for every execution of the system constructs an edge assignment that passes the consistency check.

**Definition 4.** A consistency check for a consistency semantic \(C\) is a set of conditions that an observer graph must satisfy to be called consistent with respect to \(C\).

**Definition 5.** An execution \(\alpha\) is \(C\)-consistent if there exists an edge assignment for \(\alpha\) such that the resulting observer graph satisfies \(C\)'s consistency checks.

A final bit of housekeeping:

**Definition 6.** We say that vertex \(u\) precedes vertex \(v\) in observer graph \(G\) (denoted as \(u \prec_G v\)) if there is a directed path from \(u\) to \(v\) in \(G\). By extension, we say that the operation corresponding to \(u\) precedes the one corresponding to \(v\). If \(u \prec_G v\), then \(v\) depends on \(u\). If \(u \not\prec_G v\) and \(v \not\prec_G u\), then we say that \(u\) and \(v\) are concurrent.

We now define the set of executions admitted by FJC consistency semantics in terms of its consistency checks.

**Fork-join-causal consistency:** An execution \(\alpha\) is said to be fork-join-causal (FJC) consistent if there exists an edge assignment for \(\alpha\) that produces an observer graph \(G\) that satisfies the following consistency check:

1. **Serial ordering at each correct node.** The ordering of operations by any correct node is reflected in the observer graph. Specifically, if \(p\) is a correct node and \(v\) and \(v'\) are vertices corresponding to operations by \(p\), then \(v.s < v'.s \iff v \prec_G v'\).

2. **Reads by correct nodes return the latest preceding concurrent writes.** For any read operation \(r = (p.s, \text{old}, \text{wl})\) issued by a correct node \(p\), and writes \(w\) and \(w'\) to object \(\text{old}\), the following condition holds:

\[
w \in \text{wl} \iff w \prec_G r \land \exists w' : w \prec_G w' \prec_G r
\]

**Comparison with causal consistency.** Causal consistency enforces conditions that are analogous to the one enforced by FJC, but it requires them to hold for operations issued by all nodes—not just correct nodes. Specifically, an execution \(\alpha\) is said to be causally consistent if there exists an edge assignment for \(\alpha\) that produces an observer graph \(G\) that satisfies the following consistency check:

1. **Serial ordering at each node.** The ordering of operations by any correct node is reflected in the observer graph. Specifically, if \(p\) is a node and \(v\) and \(v'\) are vertices corresponding to operations by \(p\), then \(v.s < v'.s \iff v \prec_G v'\).
FIG. 1—An execution with a faulty node $p_2$ and its corresponding observer graph. The observer graph is not causally consistent because $w_1$ and $w_2$ are not ordered according to the history of node $p_2$. The observer graph is, however, FJC consistent because $p_2$ is faulty and therefore FJC consistency doesn’t require total ordering of $p_2$’s operations.

2. **Reads return the latest preceding concurrent writes.**

For any read operation $r = (p, s, old, w)$ issued by a node $p$, and writes $w$ and $w'$ to object old, the following condition holds:

$$w \in wl \iff w \prec_G r \land \not\exists w' : w \prec_G w' \prec_G r$$

Figure 1-(a) shows an execution that is FJC consistent but not causally consistent. In this example, node $p_2$ is faulty and produces two writes $w_1$ and $w_2$. Node $p_1$ observes $w_1$ but not $w_2$, and node $p_3$ observes $w_2$ but not $w_1$. As Figure 1-(b) illustrates, we can produce an edge assignment and observer graph that passes all FJC tests by violating the serial ordering constraint at the faulty node. Conversely, it is impossible to produce an edge assignment to produce an observer graph $G'$ that passes the causal consistency checks.