TUPV033

Distributed Transactions in CERN's Accelerator Control System

F. Hoguin, S. Deghaye, P. Mantion, J. Lauener, R. Gorbonosov, CERN, Geneva, Switzerland

ICALEPCS 2021

Distributed transactions in accelerator controls ensure the consistency of settings

A client who wants to change the settings of dipoles and RF cavities simultaneously must...

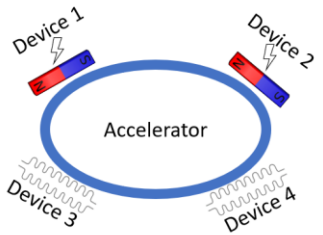
Send all the settings in parallel requests, as fast as possible → Devices reply OK or NOK → If all the requests succeed, the settings are applied. / If any of the request fails, the accelerator is in a bad state.

Without transactions

Open a transaction on all the devices → Send the settings. Devices reply OK or NOK, but settings are not applied yet. → If all the requests are OK, commit the transaction. All the settings are applied. / If any request fails, rollback the transaction. No setting has been changed.

With transactions

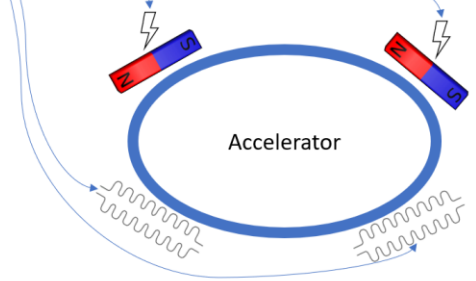
- Any error puts the accelerator in a bad state
- Vulnerable to network errors
- Dependent on the client's machine and network connection



- Errors are detected before any change is made
- Possibility to cancel the transaction and keep the accelerator in a good state
- Commit is done using the timing system, independently of the client

The timing system broadcasts commit or rollback events for simultaneous application

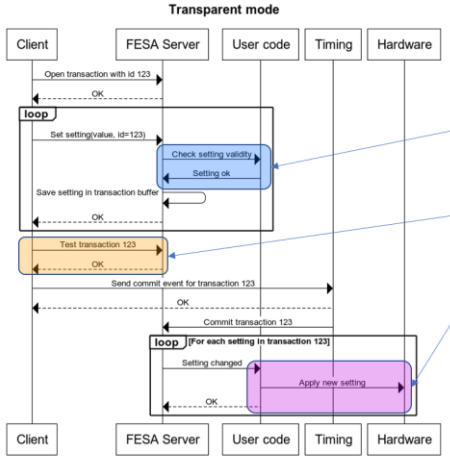
Commit transaction 123 → Timing system → COMMIT 123



- Transaction commit relies on the timing system instead of the client
- Commit event is sent simultaneously to all the devices, and can be used for synchronisation
- Immune to network latency, collisions, or saturation

Real-time frameworks support transactions transparently

In transparent mode, the application of new settings is delayed until the transaction is committed. Users of the framework don't need to change a single line of code.

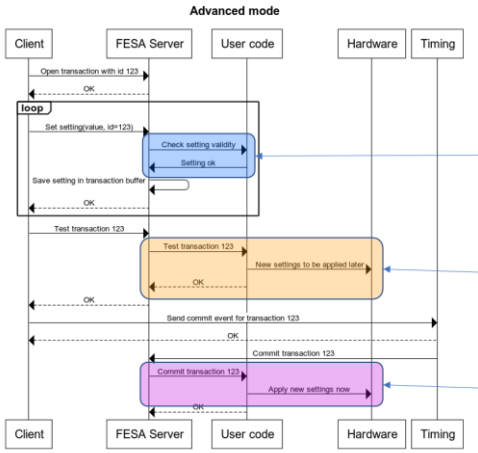


User code checks individual settings validity. Errors are reported to the client before any change is applied.

In transparent mode, the transaction test does nothing and is always successful.

User code applies the changes in the same way it would without a transaction.

With the 2-phase commit, double buffering on the hardware becomes possible



In advanced mode, users can take advantage of hardware capabilities, such as double buffering, to ensure the hardware is ready for the commit event.

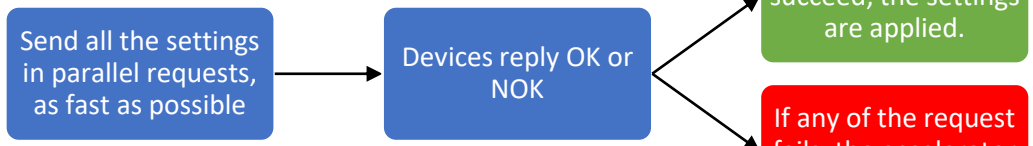
User code checks individual settings validity. Errors are reported to the client before any change is applied.

User code checks new settings, and if the hardware supports double buffering, loads them on a free buffer. If a hardware error occurs, it is reported, and the transaction can be rolled back.

When the timing system broadcasts a commit event, the user code tells the hardware to swap the buffers.

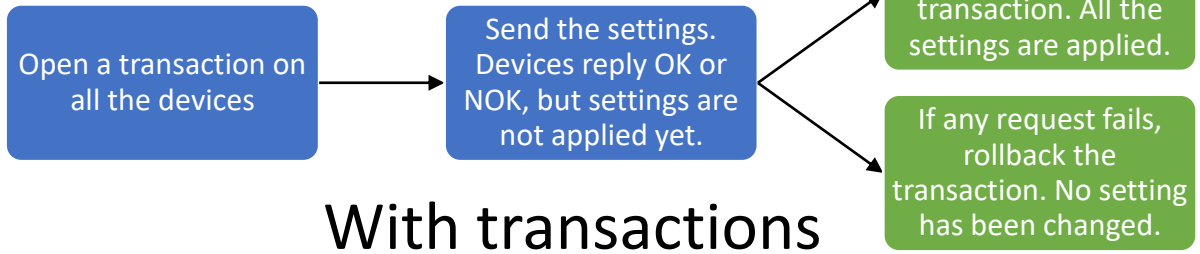
Distributed transactions in accelerator controls ensure the consistency of settings

A client who wants to change the settings of dipoles and RF cavities simultaneously must…

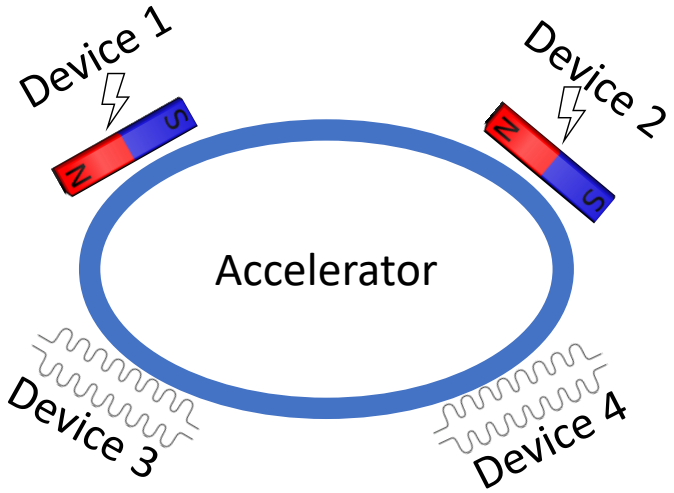


Without transactions

- Any error puts the accelerator in a bad state
- Vulnerable to network errors
- Dependent on the client's machine and network connection

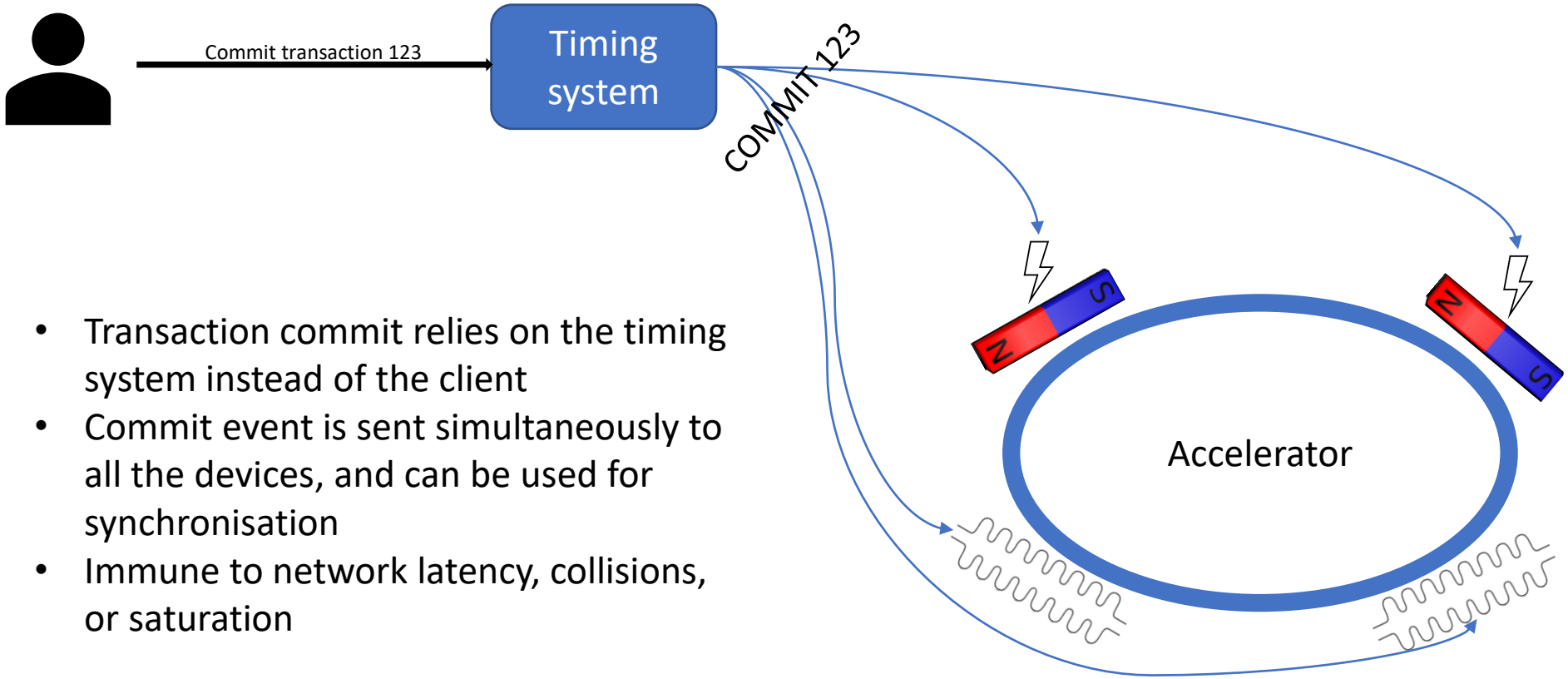


With transactions



- Errors are detected before any change is made
- Possibility to cancel the transaction and keep the accelerator in a good state
- Commit is done using the timing system, independently of the client

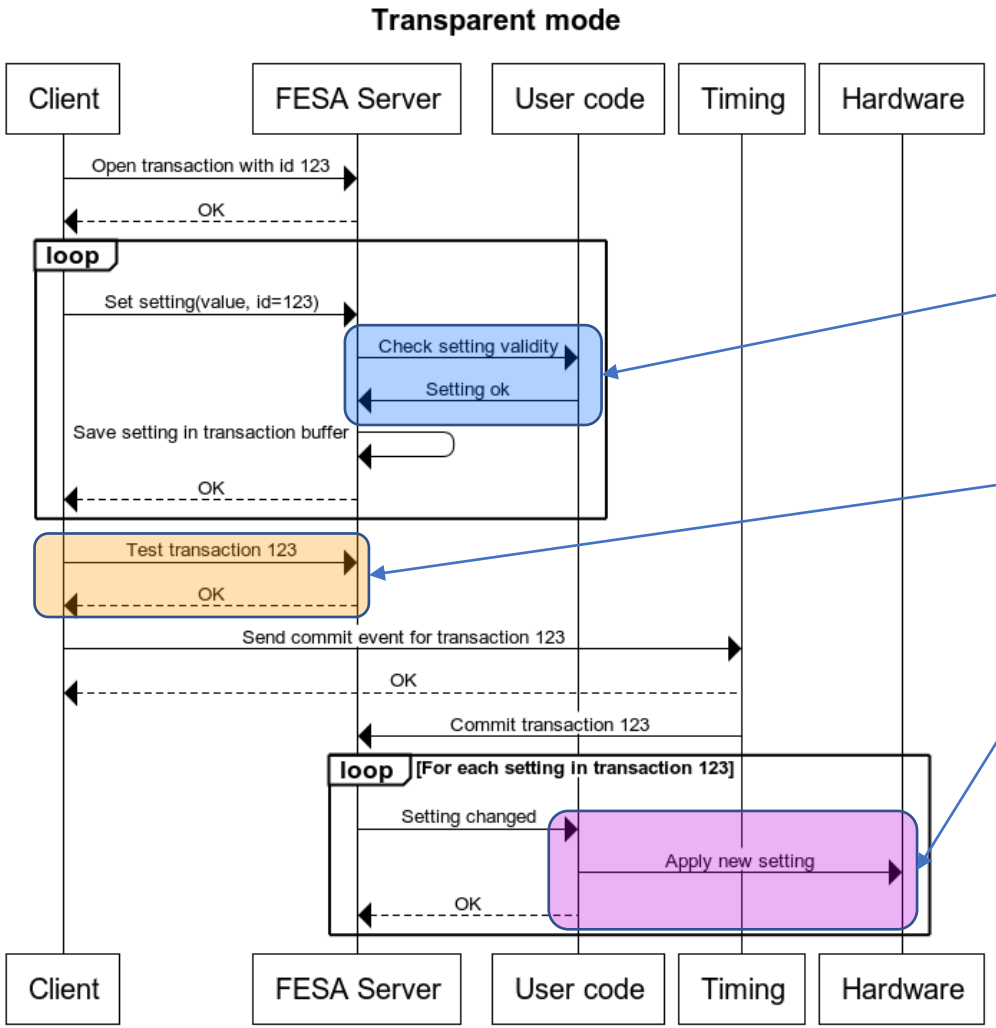
The timing system broadcasts commit or rollback events for simultaneous application



- Transaction commit relies on the timing system instead of the client
- Commit event is sent simultaneously to all the devices, and can be used for synchronisation
- Immune to network latency, collisions, or saturation

Real-time frameworks support transactions transparently

In transparent mode, the application of new settings is delayed until the transaction is committed. Users of the framework don't need to change a single line of code.

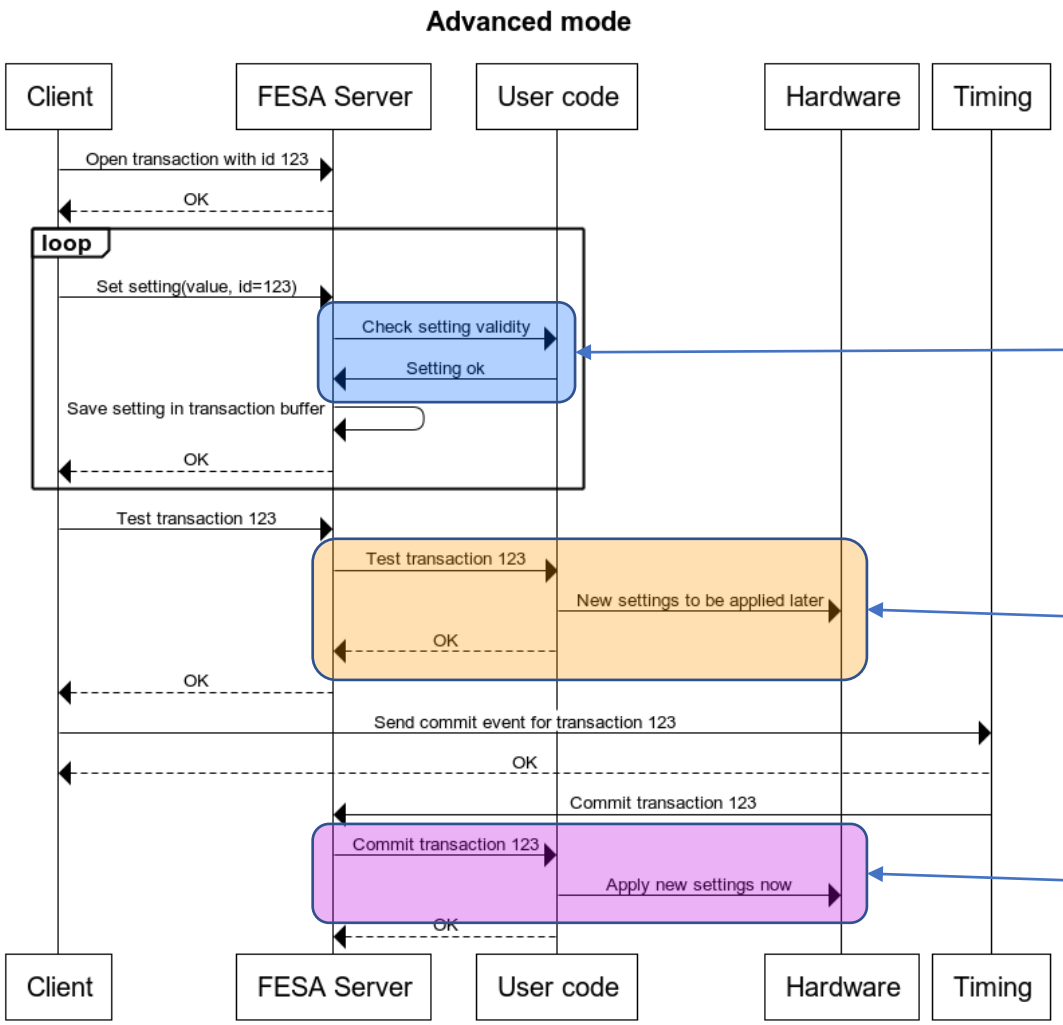


User code checks individual settings validity. Errors are reported to the client before any change is applied.

In transparent mode, the transaction test does nothing and is always successful.

User code applies the changes in the same way it would without a transaction.

With the 2-phase commit, double buffering on the hardware becomes possible



In advanced mode, users can take advantage of hardware capabilities, such as double buffering, to ensure the hardware is ready for the commit event.

User code checks individual settings validity. Errors are reported to the client before any change is applied.

User code checks new settings, and if the hardware supports double buffering, loads them on a free buffer. If a hardware error occurs, it is reported, and the transaction can be rolled back.

When the timing system broadcasts a commit event, the user code tells the hardware to swap the buffers.