

Non-Blocking Inter-Partition Communication with Wait-Free Pair Transactions

Ethan Blanton and Lukasz Ziarek
Fiji Systems, Inc.

October 10th, 2013

Wait-Free Pair Transactions

A communication mechanism that is:

- Lightweight
- Transactional
- Non-blocking
- One-way
- Safe
- “Natural”

WFPTs Cont'd

WFPTs are a shared-memory mechanism.

Transactions are explicit:

- Writers commit
- Readers update

Referential integrity is guaranteed.

Access to transaction fields requires no application changes.

Motivation

Real-time systems may contain tasks of differing **priority**.
Communication among such tasks must bound **priority inversion**.
Traditional solutions for this are:

- Priority Inheritance Protocol
- Priority Ceiling Protocol
- Asynchrony (not easy in Java!)

The scene is complicated by tasks of differing **criticality**.

Mixed-Criticality Systems

The Fiji Multi-VM allows multiple **independent** Java VMs to execute, isolated in **time** and **space**.

This complicates **safe**, **low-overhead** communication.

WFPTs are ideal for crossing VM partitions!

Wait-Free Pair Transaction Semantics

WFPT objects provide two contexts, **reader** and **writer**.

Both contexts:

- Never block
- Never lose coherence
- Have $O(1)$ access to transaction fields

Writers:

- Can **commit** their changes
- Never share changes without committing

Readers:

- Can **update** to the writer's committed changes
- Never share their changes

Java Language WFPT Semantics

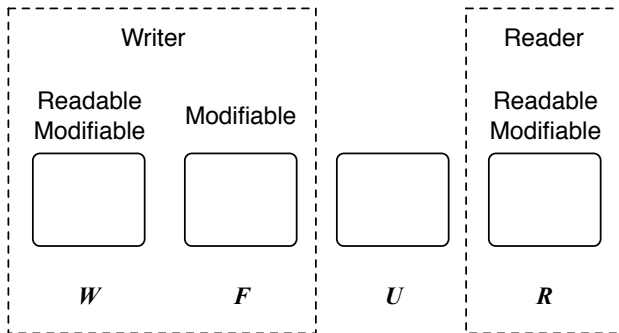
- WFPT objects subclass `WaitFreePairTransaction`
- Static fields are not subject to WFPT semantics.
- Static fields are not shared between partitions for inter-partition WFPT.
- WFPT object field accesses behave like “regular” objects.
- The `commit` and `update` operations are provided by final methods on `WaitFreePairTransaction`.

WFPT Object Model

WFPT Objects don't use the "normal" Fiji VM Java object model

- WFPT object fields are **replicated four times**.
 - These replicas are named by mutable **indexes** W , F , U , and R .
 - Each index has a different role in the transaction.
 - The replica at W is currently being written by the writer.
 - F will receive the writer's next **commit**.
 - U will become the read index on the next **update**.
 - R is currently being read by the reader.
- An **update flag** indicates whether there has been a **commit** since the last **update**.
- A tracking structure is used for each WFPT object to store WFPT state and facilitate transactions.

WFPT Replicas

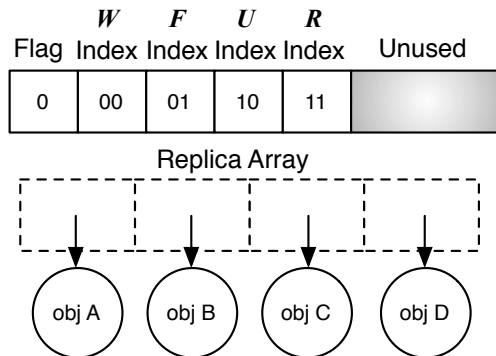


Commit

- 1 Copy the contents of the replica at W to the replica at F .
- 2 Atomically swap the indexes U and F while setting the update flag.

Note that this operation modifies only the replica at F (before the swap) and affects only the indexes F and U .

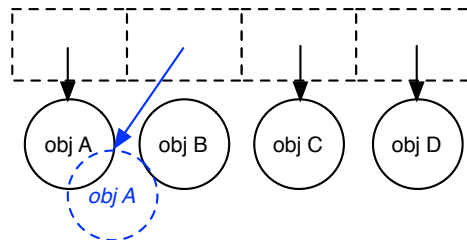
Replica State Prior to Commit



Replica State After Commit Phase I

	<i>W</i>	<i>F</i>	<i>U</i>	<i>R</i>	Unused
Flag	Index	Index	Index	Index	
0	00	01	10	11	

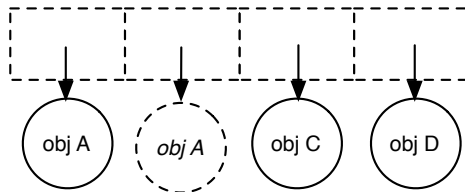
Replica Array



Replica State After Commit Phase II

	<i>W</i>	<i>F</i>	<i>U</i>	<i>R</i>	Unused
Flag	Index	Index	Index	Index	
<i>1</i>	00	<i>10</i>	<i>01</i>	11	

Replica Array

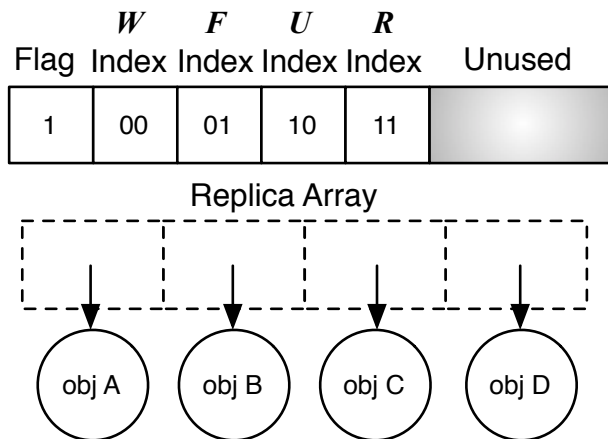


Update

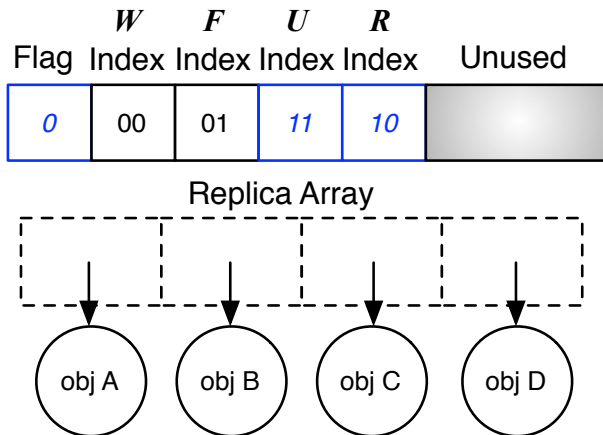
- 1 Check the update flag, return if clear.
- 2 Atomically swap the indexes U and R while clearing the update flag.

Note that this operation modifies no replicas, and affects only the indexes U and R .

Replica State Prior to Update



Replica State After Update



Complexity

For the higher priority context on a WFPT:

- `update` runs in $O(1)$ time.
- `commit` runs in time proportional to the size of the object.
- All operations are **completely independent** of the other context's state.

For the lower-priority context on a WFPT:

- Transaction complexity remains the same.
- Field accesses are **completely independent** of the other context's state.
- Transactions **may be affected** by the other context.

Safety and Correctness

The safety and correctness of WFPT transactions are predicated upon:

- The replica at U is never modified.
- The writer only reads or modifies W and F .
- The reader only reads or modifies R .
- F and U are swapped only when all of the following are true:
 - The update flag is set,
 - F contains the newest commit, and
 - U contains the most recent previous commit, if it exists.

Safety and Correctness, cont'd

Thus:

- Neither context can “see” the other context’s current replica.
- The reader is always guaranteed to update to either:
 - The **most recent** commit, or
 - The **immediately previous** commit.

Implementation Overview

WFPT involves changes to both the runtime and compiler.

Compiler:

- Four new classes
- Almost entirely self-contained
- < 500 lines

Runtime:

- WFPT objects replaced with WFPTProxy tracking objects
- One new exception
- Trivial change to `newInstance()`

Compiler Changes

- Tracking structure allocation at `new` time
- Method lookasides
- Cast replacements to hide `WFPTProxy`
- Access barriers to perform context lookups

Proxy Objects

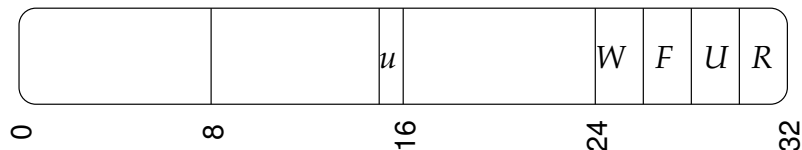
The proxy object contains four items:

- One word for index values and update flag
- Array of four WFPT objects
- Reader context object
- Writer context object

The context objects are entirely uninteresting.

Index Field

All four index values and the update flag are packed into a **single 32-bit word**:



Index updates are then performed by **strong CAS**.

Inter-partition Communication

Some restrictions must be levied for inter-partition WFPT:

- IP-WFPT objects must be negotiated *a priori* and are allocated in “immortal” memory.
- Reference fields are forbidden.
- WFPT object structure must be identical in size and layout across partitions.
- Concurrent access to IP-WFPT objects does not guarantee coherency **within a partition**.

WFPT Overhead

Worst-case micro-benchmarks:

```
public synchronized void set(int value) {  
    this.value = value;  
}
```

```
public void set(int value) {  
    this.value = value;  
}
```

Micro-benchmark Results

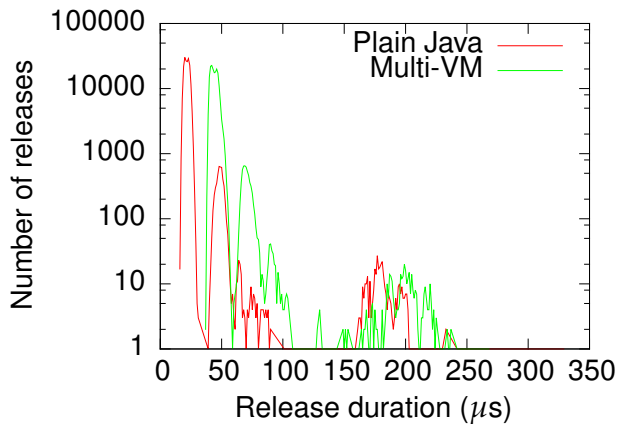
Configuration	ARM	LEON3
Monitors	123	5107
WFPT	260	6557
WFPT w/ Commit	653	14107

All values are in nanoseconds.

Multi-VM jPapaBench

- jPapaBench consists of three major modules:
 - Autopilot
 - Simulator
 - Fly-by-Wire (FBW)
- We placed the FBW module in its own VM.
- Communication between VMs is via three WFPT objects.

Stabilization Release Durations



Conclusions

- WFPT is a viable mechanism for both inter- and intra-partition communication.
- Optimization for elision of context checks is wanted for performance, to bring overheads closer to monitors.
- Inter-partition WFPT is viable in a Multi-VM, but safety requires restrictions that should be carefully examined.

Questions?

Non-Blocking Inter-Partition Communication with Wait-Free Pair Transactions

Ethan Blanton and Lukasz Ziarek
Fiji Systems, Inc.

<http://fiji-systems.com/academia/>