User Experiences with MPI RMA and ULFM in a Resilient Key-Value Store Implementation

Claudia Fohry and Rainer Fink
University of Kassel (Germany), Programming Languages/Methodologies, fohry@uni-kassel.de

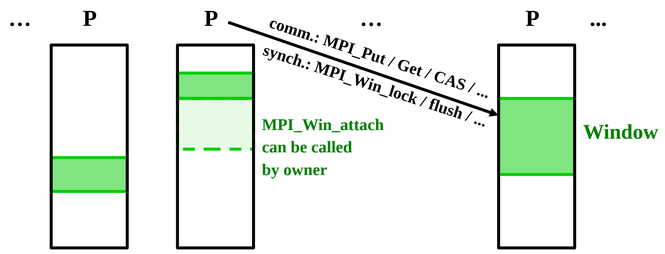
Motivation

- Consider task-level checkpointing for task-based runtime system Itoyori
  - Itoyori is written in MPI, tasks communicate via futures
  - checkpoints (task descriptors, futures, ...) are saved regularly and during work stealing, independently by each worker
  - shrinking, localized recovery after process aborts
- Resilient store needed to save checkpoints
  - in-memory, distributed, replication-based key-value store, such as Hazelcast IMap, Redis, etcd
- MPI-based store preferable, for better integration and adaptation to application needs:
  - frequent writes, few reads
  - fixed number of keys per worker, size of values unknown
  - no concurrent accesses to same key, except after failures
  - transactions required for failure-during-recovery scenarios
  - lost replicas should be replaced

⇒ We developed such a store.

MPI Technologies

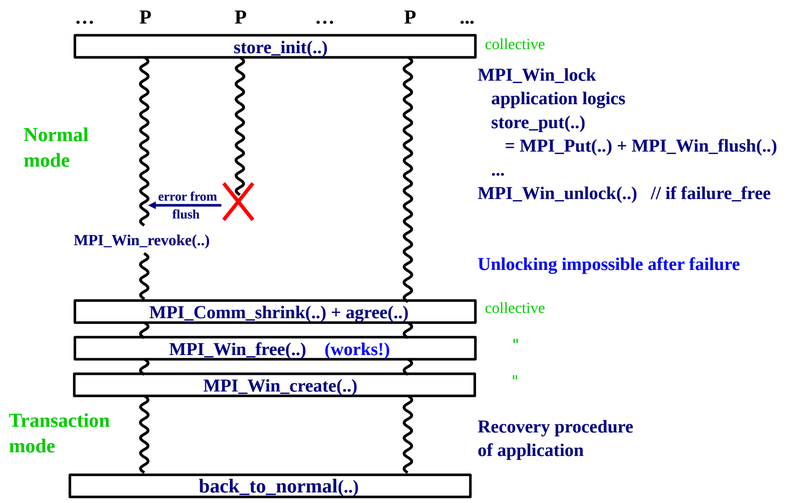
- Passive target MPI RMA is natural communication mechanism for store



- ULFM for failure detection and recovery (implemented in Open MPI)
  - affected processes receive error, e.g., `MPI_ERR_PROC_FAILED`
  - error propagation via `MPI_Comm_revoke` → all receive error
  - collective communicator repair, e.g., `MPI_Comm_shrink`
- ULFM support for RMA specified, but documented as untested
  - error codes from `MPI_Win_flush` / `MPI_Win_lock`, ...
  - `MPI_Win_revoke`

Store Designs and MPI/ULFM Realization

Global Recovery Variant:



- Synchronous: `store_put` writes to master copy and all replicas
- Data copies are evenly spread across the machine
- No concurrent writes in normal mode
- Performance optimization focuses on normal mode

Localized Recovery Variant (not yet implemented)**:**

- Replace MPI locks by application-level locks via `CAS(pid)`
- No distinction into normal / transaction modes
- Single epoch: `MPI_Win_lock_all(MPI_MODE_NOCHECK, ...)`
- Other problems remain: window enlargement / replacement etc.

Missing Features

- Enforced unlocking function or locking function with timeout,
- Signaling mechanism to allow one process to interrupt another,
- Failure propagation to a subset of processes,
- Blocking variant of `MPI_Put`,
- Window creation/deletion functions with implicit epoch definition,
- Function for shrinking a window without shrinking the communicator,
- Error message or warning issued when `MPI_Win_free` is called on a locked window, and
- Query function for revocation state of a communicator that works without prior communication.

Practical Limitations and Workaround

Current Status:

- ULFM support for MPI RMA not yet functional

Workaround:

- "ping" preceding each MPI RMA call to obtain error code
- Realized in helper thread
  - via `MPI_Send` and `MPI_Iprobe(...ANY_SOURCE...)`
  - one helper thread per process, `MPI_THREAD_MULTIPLE`
- Remaining risk of failure between ping and RMA call
  - leads to non-termination
  - error handler `MPI_ERRORS_ARE_FATAL` for RMA communication did not help
- Helper thread also enables window enlargement

Preliminary Experiments:

No failures, but using `with-ft ulfm`:

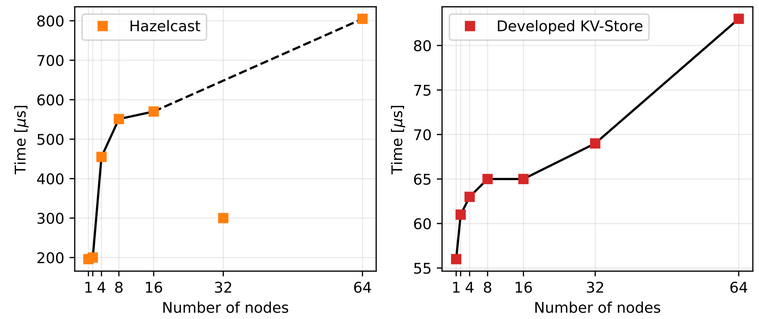


Figure 1: `store_put` latencies of Hazelcast (left) and our store (right), with 6 replicas and 40 processes per node (minor differences in setting)

Conclusions

- Resilient store implementation project was only partially successful
  - remaining risk of non-terminating behavior
  - workaround causes performance overhead
- Implementation of specified ULFM functionality for MPI RMA is urgently needed
- Enhancements of specified ULFM functionality are desirable
- Future work includes implementation of localized recovery variant