

Efficient Checkpointing in Byzantine Fault-Tolerant Systems

November 22, 2019

Michael Eischer, Tobias Distler

Friedrich-Alexander Universität Erlangen-Nürnberg (FAU)



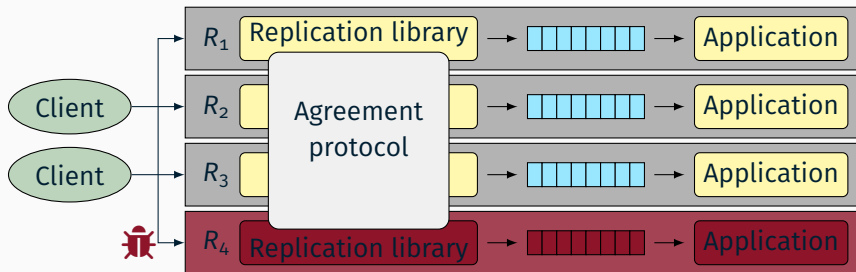
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

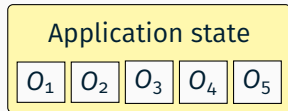
Supported by



grant no. DI 2097/1-2 ("REFIT")



- Replicate service for fault tolerance
- Tolerate Byzantine (arbitrary) faults
- Application state consists of objects O_i with unique object IDs





- Keep requests to tolerate faults



- Keep requests to tolerate faults
- Collect garbage after checkpoint
- Create checkpoints at fixed interval

| | Request execution | | | | Check-pointing | Request execution | | | | Check-pointing | |
|-------|-------------------|------------|-------|-------|-------------------|-------------------|-------|-------|------------|-------------------|---|
| R_1 | O_1 | O_1, O_3 | O_3 | O_1 | O_1, \dots, O_5 | O_2, O_3 | O_1 | O_5 | O_1, O_5 | O_1, \dots, O_5 | ✓ |
| R_2 | O_1 | O_1, O_3 | O_3 | O_1 | O_1, \dots, O_5 | O_2, O_3 | O_1 | O_5 | O_1, O_5 | O_1, \dots, O_5 | ✓ |
| R_3 | O_1 | O_1, O_3 | O_3 | O_1 | O_1, \dots, O_5 | O_2, O_3 | O_1 | O_5 | O_1, O_5 | O_1, \dots, O_5 | ✓ |
| R_4 | O_1 | O_1, O_3 | O_3 | O_1 | O_1, \dots, O_5 | O_2, O_3 | O_1 | O_5 | O_1, O_5 | O_1, \dots, O_5 | ✗ |
| | <i>time</i> → | | | | | | | | | | |

- Keep requests to tolerate faults
- Collect garbage after checkpoint
- Create checkpoints at fixed interval
- Unverified checkpoint might be corrupted
 - Check for $f + 1$ identical checkpoint hashes

Full Checkpointing

| | Request execution | | | | Checkpointing | Request execution | | | | Checkpointing |
|-------|-------------------|------------|-------|-------|-------------------|-------------------|-------|-------|------------|-------------------|
| R_1 | O_1 | O_1, O_3 | O_3 | O_1 | O_1, \dots, O_5 | O_2, O_3 | O_1 | O_5 | O_1, O_5 | O_1, \dots, O_5 |
| R_2 | O_1 | O_1, O_3 | O_3 | O_1 | O_1, \dots, O_5 | O_2, O_3 | O_1 | O_5 | O_1, O_5 | O_1, \dots, O_5 |
| R_3 | O_1 | O_1, O_3 | O_3 | O_1 | O_1, \dots, O_5 | O_2, O_3 | O_1 | O_5 | O_1, O_5 | O_1, \dots, O_5 |
| R_4 | O_1 | O_1, O_3 | O_3 | O_1 | O_1, \dots, O_5 | O_2, O_3 | O_1 | O_5 | O_1, O_5 | O_1, \dots, O_5 |
| | <i>time</i> → | | | | | | | | | |

- Copy every object
- Stop the world: Identical checkpoints
- Service not available during checkpointing

Differential Checkpointing¹

| | Request execution | | | | Checkp. | Request execution | | | | Checkpointing |
|-------|-------------------|------------|-------|-------|------------|-------------------|-------|-------|------------|----------------------|
| R_1 | O_1 | O_1, O_3 | O_3 | O_1 | O_1, O_3 | O_2, O_3 | O_1 | O_5 | O_1, O_5 | O_1, O_2, O_3, O_5 |
| R_2 | O_1 | O_1, O_3 | O_3 | O_1 | O_1, O_3 | O_2, O_3 | O_1 | O_5 | O_1, O_5 | O_1, O_2, O_3, O_5 |
| R_3 | O_1 | O_1, O_3 | O_3 | O_1 | O_1, O_3 | O_2, O_3 | O_1 | O_5 | O_1, O_5 | O_1, O_2, O_3, O_5 |
| R_4 | O_1 | O_1, O_3 | O_3 | O_1 | O_1, O_3 | O_2, O_3 | O_1 | O_5 | O_1, O_5 | O_1, O_2, O_3, O_5 |
| | time → | | | | | | | | | |

- Only copy changed objects
- Merge with full checkpoint afterwards
- Large objects / expensive state-retrieval still problematic

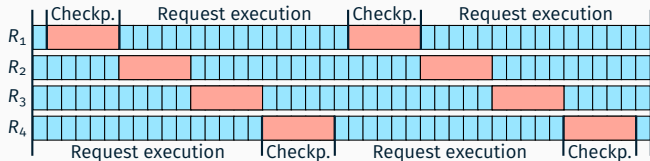
¹Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance and Proactive Recovery". In: *ACM Trans. on Computer Systems* 20.4 (2002), pp. 398–461.

Hybrid Checkpointing²

- Infrequent full checkpoints
- Combine with log of requests
- Reexecution of request log for checkpoint application
 - Requests causing failures could trigger these again

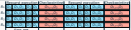
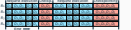
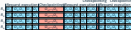
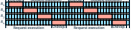
²Allen Clement et al. "UpRight Cluster Services". In: *Proc. of the 22nd Symp. on Operating Systems Principles*. 2009, pp. 277–290.

Sequential Checkpointing³



- Full checkpoints at different sequence numbers
- Not directly comparable
- Application of request log to recreate checkpoints
- **Verification of checkpoint after state application**

³Alysson Bessani et al. "On the Efficiency of Durable State Machine Replication".
In: *Proc. of the 2013 USENIX Annual Technical Conf.* 2013, pp. 169–180.

| Checkpointing Method | Resilience | Efficiency |
|--|------------|------------|
|  Full | ✓ | ✗ |
|  Differential | ✓ | ○ |
|  Hybrid | ○ | ○ |
|  Sequential | ✗ | ✓ |

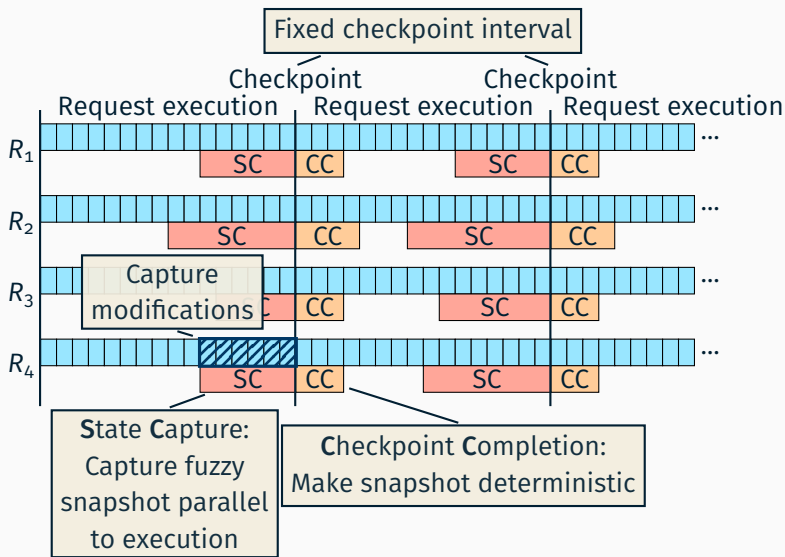
Challenges

Need for Byzantine fault-tolerant checkpointing mechanism that is

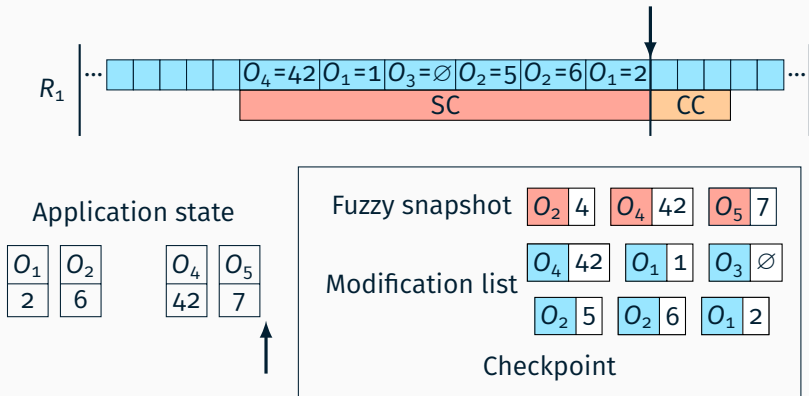
- **Resilient:** Validate checkpoint before applying
- **Efficient:** Low performance impact

1. Motivation
2. Our Approach: Deterministic Fuzzy Checkpoints
3. Evaluation
4. Summary

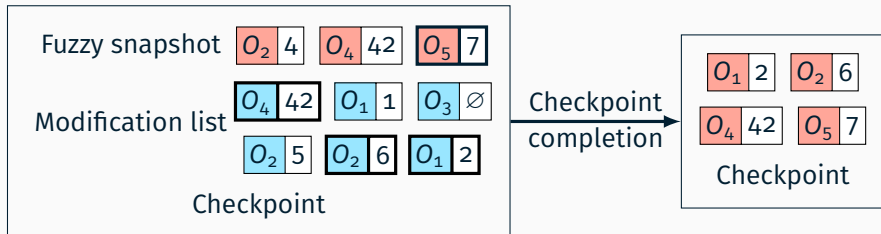
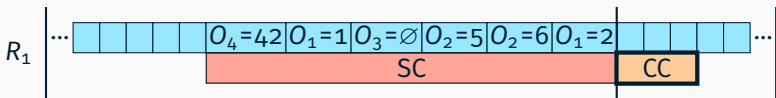
Our Approach:
Deterministic Fuzzy Checkpoints



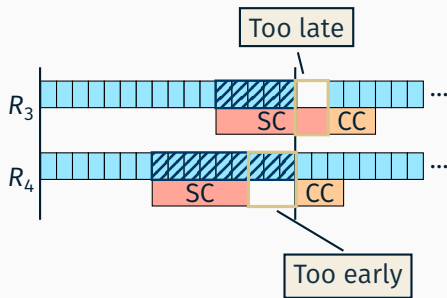
- Capture state parallel to request execution
- Snapshots differ between replicas



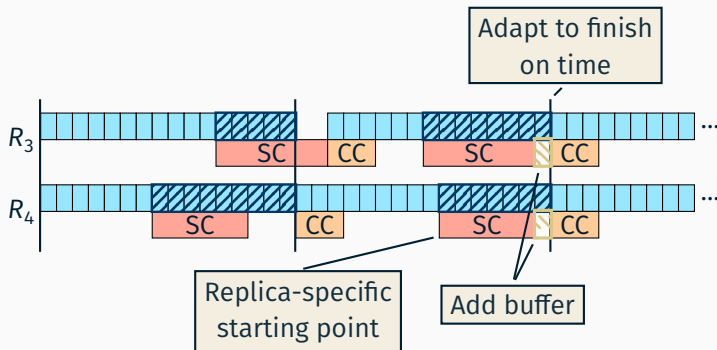
- Apply modifications to snapshot
- Creates an *identical checkpoint on all replicas*
 - Snapshot or modification list contain latest version of each object



- Adapt starting point to finish on time
 - Goal: Minimize overhead
 - Account for capture time in sequence numbers
 - Add buffer time
- Adapt to heterogeneous server performance



- Adapt starting point to finish on time
 - Goal: Minimize overhead
 - Account for capture time in sequence numbers
 - Add buffer time
- Adapt to heterogeneous server performance



Application interface

similar to BASE⁴

```
// Request execution  
RESULT invoke(REQUEST r);  
@Callback  
void modified(OBJECTID oid);
```

```
// Checkpointing  
BYTE[] object(OBJECTID oid);
```

- Replication library has access to individual state objects
- Generic snapshot handling
- State capture
 - Checkpointer thread collects copy of all objects
 - Modification list: At checkpoint sequence number copy final state of objects modified during state capture
- Checkpoint completion
 - Keep latest version of an object

⁴Miguel Castro, Rodrigo Rodrigues, and Barbara Liskov. "BASE: Using Abstraction to Improve Fault Tolerance". In: *ACM Trans. on Computer Systems* 21.3 (2003)

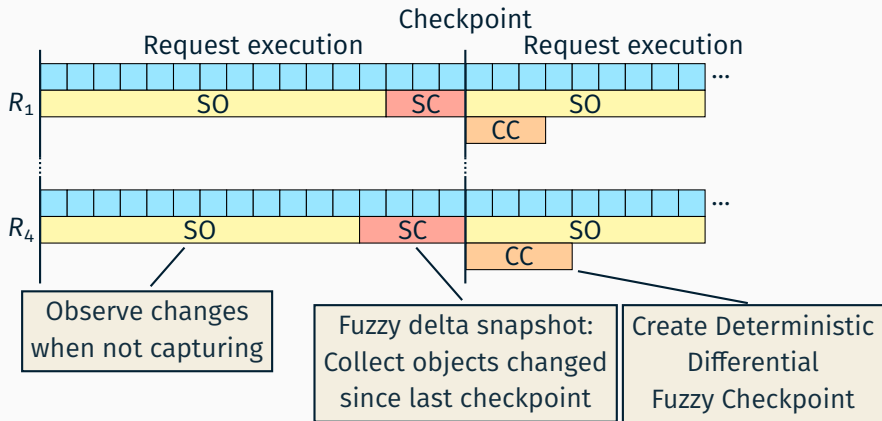
Application interface

```
// Request execution  
[RESULT, UPDATE] invoke(  
    REQUEST r,  
    BOOLEAN createUpd);
```

```
// Checkpointing  
SNAPSHOT fuzzy();
```

```
// Completion  
SNAPSHOT complete(  
    SNAPSHOT s, UPDATE[] u);
```

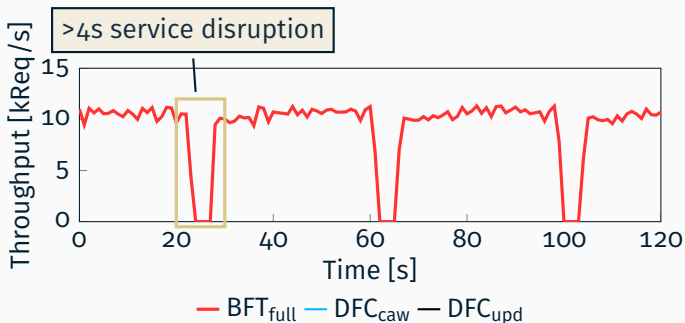
- Content of UPDATES and SNAPSHOT is application-specific
- Fine-grained modification tracking
- State capture
 - Concurrent snapshot creation
 - Modification list: Library collects list of UPDATES
- Checkpoint completion
 - Apply collected UPDATES



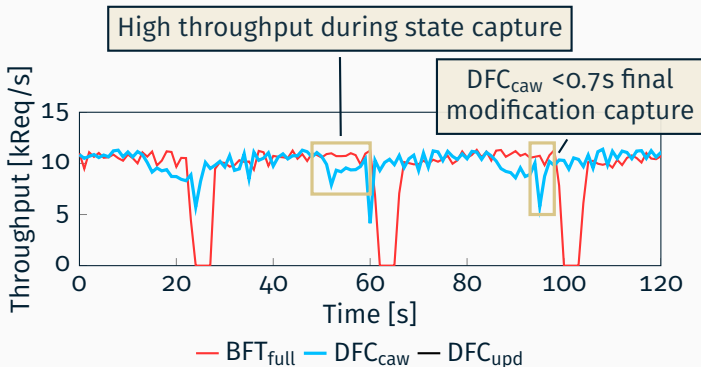
→ Merge with latest full checkpoint for up-to-date full checkpoint

Evaluation

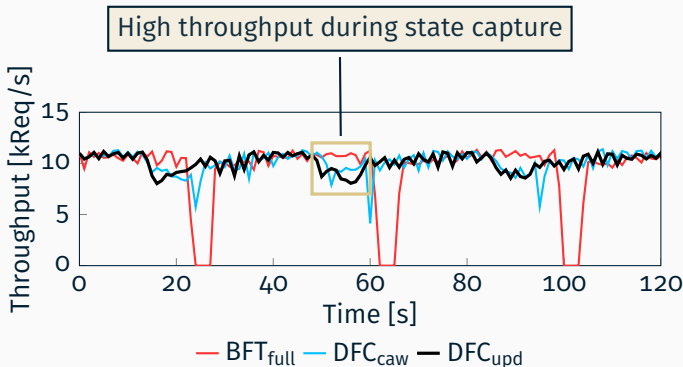
- Application: Key-value store with in-memory SQLite database
 - Application state 3GB (750k objects à 4kb)
 - Mixed read/write request on single entry
 - Checkpoint approximately every 400k requests
- Four replicas (4 cores, 3.6 GHz)
- 100 client instances on one server (12 cores, 2.4 GHz)



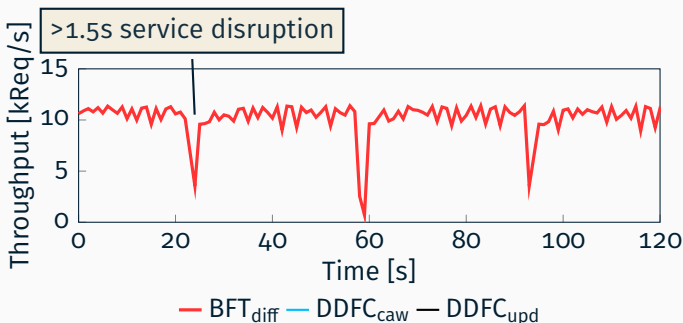
- Application: Key-value store with in-memory SQLite database
 - Application state 3GB (750k objects à 4kb)
 - Mixed read/write request on single entry
 - Checkpoint approximately every 400k requests
- Four replicas (4 cores, 3.6 GHz)
- 100 client instances on one server (12 cores, 2.4 GHz)



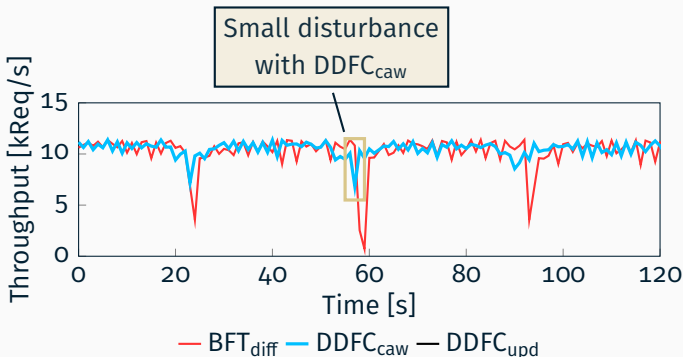
- Application: Key-value store with in-memory SQLite database
 - Application state 3GB (750k objects à 4kb)
 - Mixed read/write request on single entry
 - Checkpoint approximately every 400k requests
- Four replicas (4 cores, 3.6 GHz)
- 100 client instances on one server (12 cores, 2.4 GHz)



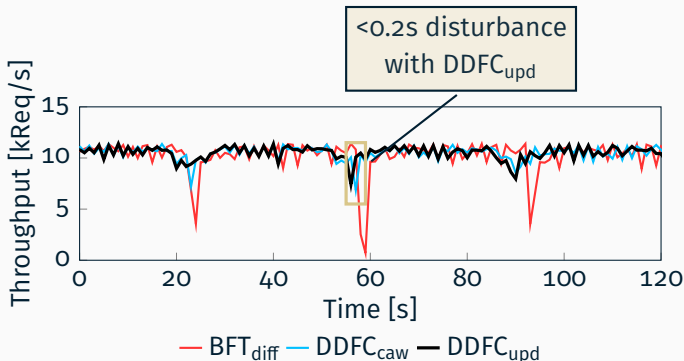
- More than >200k changed objects



- More than >200k changed objects



- More than >200k changed objects



Summary

| Checkpointing Method | Resilience | Efficiency |
|----------------------|------------|------------|
| Full | ✓ | ✗ |
| Differential | ✓ | ○ |
| Hybrid | ○ | ○ |
| Sequential | ✗ | ✓ |
| DFC ¹ | ✓ | ✓ |

Deterministic Fuzzy Checkpoints

- Fuzzy state capture parallel to execution
- Deterministic checkpoint after completion

Thank you for your attention

Questions?

¹Michael Eischer, Markus Büttner, and Tobias Distler. “Deterministic Fuzzy Checkpoints”. In: *Proc. of the 38th Symp. on Reliable Distributed Systems*. 2019.