Task-Level Checkpointing for Nested Fork-Join Programs

Lukas Reitz
Advisor: Prof. Dr. Claudia Fohry
Research Group Programming Languages / Methodologies
University of Kassel, Germany
lukas.reitz@uni-kassel.de

UNIKASSEL VERSITÄT

Motivation

- Checkpointing is an established technique for tolerating fail-stop failures.
- Task-level checkpointing can be efficient, because only task descriptors are saved.
- We consider Nested Fork-Join (*NFJ*) programs (see below).
- Recently, a task-level checkpointing scheme (*CP*) for NFJ programs has been sketched [1].
- *Problem:* CP has been designed for programs that restrict the number of worker threads per process to one.

Contribution

- **We devise a new backup writing scheme:**
  - It supports multiple worker threads per process.
  - Checkpoints are saved per process to reduce the overall number of checkpoints.
  - Workers independently contribute to a process checkpoint.

Nested Fork-Join Programs

- Tasks may spawn child tasks, wait for their results, and return a result to their parent. An example of a naive implementation of Fibonacci:

```
1 func fib(n) {
2   if (n <= 2) return 1;
3   a = spawn fib(n - 1);
4   b = spawn fib(n - 2);
5   sync;
6   return a + b;
7 }
```

- Tasks do not have side effects.
- The NFJ task model is suitable for, e.g., divide-and-conquer algorithms.
- Each worker maintains an own task queue.
- When a process runs out of tasks, it steals tasks from another process. Local workers additionally share some tasks.
- We consider the work-first policy: When a task spawns another task, the worker branches into the child task and puts the continuation of the parent task into its queue.
- Each worker alternates between task processing phases and communication phases.
- Local workers store certain data in a *shared data* location, e.g., intermediate results.

Task-Level Checkpointing Scheme

Original Scheme (CP)

- The original checkpointing scheme saves checkpoints at:
  - regular time intervals,
  - in the event of work stealing (*steal backups*), and
  - during recovery.
- The scheme writes a checkpoint per worker thread.
- Upon failure, it recovers locally confining the failure handling to a small subset of processes.
- It deploys shrinking recovery, i.e., the program execution continues on a reduced set of processes.

Extension

- A naive approach to backup writing might pause all workers of a process from the beginning until the end of backup writing.
- Our approach avoids stalling the workers:
  1. Backup writing is initiated by an arbitrary worker, e.g., when a time interval is over.
  2. Each worker checks in its communication phase, whether a backup writing has begun, and if so, it copies its task queue.
  3. The last worker additionally writes the task queue copies and the shared data to a resilient store.
- During backup writing, operations that move frames between different parts of the shared state are logged and executed after backup writing.
- Incremental steal backups avoid delaying work stealing due to the independent contributions to the checkpoint.
- Victims only save the number of stolen tasks. This number can be updated without the participation of all workers.

Conclusions

- This work removed the restriction of CP to support multiple worker threads per process.
- Future work should implement and experimentally evaluate the scheme.

References

[1] C. Fohry, “Checkpointing and localized recovery for nested fork-join programs,” in *SuperCheck*, 2021.