System-Level Transactions with picotm

Thomas Zimmermann
tdz@users.sourceforge.net

February 12, 2018

1 The Problems with Traditional Code

Our goal is to write software that works reliably under all circumstances. Besides correctness at the algorithmic level, this requires correct handling of concurrency and run-time errors. Both are notoriously hard to test and implement.

```
int fd0, fd1; /* file descriptors */
char obuf[100]; /* output buffer */

write(fd0, obuf, sizeof(obuf));
write(fd1, obuf, sizeof(obuf));
```

This example program writes an output buffer to two files. If the first `write()` operation succeeds but the second fails, only one file will be updated and the program will enter an inconsistent, and probably erroneous, state. Correctly returning to the previous consistent state will become impossible at this point. If there's concurrent access to the files, the program's result will become unpredictable. Similar examples can be constructed for memory access, data structures, or any other non-constant resource.

2 Transactional Execution

A *transaction* is a transition from one consistent state to another consistent state; without the intermediate steps being visible outside of the transaction's context. This is known as the ACID properties *atomicity, consistency, isolation* and *durability.* To achieve ACID properties, a transaction has to handle run-time errors and concurrency of its contained operations; exactly those pieces of the software that we just identified as being problematic.

3 Phases of each Operation

Most operations, such as `write()`, can be split into different phases. We call them *execute*, *apply* and *undo.*

When first called, an operation enters the execute phase, where it allocates resources, acquires locks and checks for potential run-time errors. If successful, the operation will enter the apply phase where its effects become globally visible. If unsuccessful, the operation will enter the undo phase, where all setup from the execute phase is reverted.

For n operations in a row, phases can be rearranged. First we perform all execute phases e_1, e_2, ... e_n. If successful, we will perform all apply phases a_1, a_2, ... a_n. In unsuccessful, we will revert execution with the undo phases u_n, ... u_2, u_1. This scheme allows us to compose transactions from arbitrary operations.

4 System-Level Transactions with picotm

picotm is a system-level transaction manager implemented in highly portable C. It provides a generic framework to formalize the execute-apply-undo scheme and to integrate a large number of different modules and interfaces. All concurrency control and error detection is performed internally, users only have to implement application-specific algorithms and error recovery.

Re-implementing the example program puts all execute phases between `picotm_begin` and `picotm_commit`. Note each operation's `_tx` suffix.

```
picotm_begin
  /* execution phase */
  write_tx(fd0, obuf, sizeof(obuf));
  write_tx(fd1, obuf, sizeof(obuf));
picotm_commit /* apply during commit */
  /* recovery phase */
  put_error_recovery_here();
  picotm_restart();
picotm_end
```

Execute phases perform concurrency control and error detection internally. Each operation's apply phase is performed by `picotm_commit`. Until commit, all changes are local and can be reverted via the corresponding undo phase. After reverting from a detected error, the transaction enters the recovery phase where the application can attempt to repair the error and restart the transaction.

5 picotm Modules

The picotm core library provides the building blocks for generic error handing and concurrency control. On top of these primitives, users can implement modules for their specific use case and application. Out of the box, picotm comes with a large set of available modules, such as

Software Transactional Memory Transactional access to shared memory

Data structures Implements transactional lists, queues, multisets, stacks

Memory management Provides transaction-safe C memory allocation with `malloc()`, `free()`, et al.

C string and memory operations Provides C Standard Library functions that operate on transactional memory, such as `memset()`, `strcpy()`, et al.

File-descriptor I/O Offers transaction-safe access to files and file-like structures on Posix systems, with functions such as `open()`, `close()`, `read()`, `write()`, et al.

C math functions Transactional `tan()`, `sqrt()`, et al.

All available modules co-operate and can be mixed freely. More modules are planned and will be added over time.

6 Obtaining picotm

picotm is available at `picotm.org`. It's implemented in plain C and currently supports Linux, MacOS X, Windows (Cygwin) and FreeBSD. More systems are planned. The source code is distributed under the terms of the MIT license.