JWormBench

About

JWormBench is a port of the WormBench benchmark from C# to Java. This port extends the original benchmark in several ways, making it more useful as a testbed for evaluating STMs. Moreover, our port, which we called JWormBench, was designed to be easily extensible and to allow easy integration with different STMs
- JWormBench quickstart

WormBench

The WormBench benchmark was built to research new workloads (creating, testing, and running) for TM systems' evaluation. The idea behind WormBench is inspired in the Snake game. In this case the snakes are *worms* formed by a *body* and a *head*, moving in a shared *world* --- matrix of *nodes*. Each *node* has an integer *value* and the worms' *group id* that is over that node (worms belonging to other groups should not cross through each other).

One of the advantages of WormBench is the ability to create new configurations of the *world*, *worms*, and *worm operations*, producing new workloads with complex contention characteristics and different transaction durations and sizes, without modifying its source-code.

JWormBench Remarks

The JWormBench adds two new features important for the research of new workloads and evaluation of STM scalability: (1) the ability to specify the proportion between different kinds of *worm operations*, and (2) the ability to set the number of worms independently of the number of threads.

Furthermore, the JWormBench provides a simple API, easy to integrate with any STM implementation in Java. So, anyone may add a new synchronization mechanism (based on STM or other), implementing the appropriate abstract types and providing those implementations to JWormBench via a configuration *Guice module*. In the same way you can also extend JWomBench with new kinds of *worm operations* without modifying the core JWormBench library.

We also provide JWormBenchApp that is a Java console application that extends the JWormBench framework with some built-in *modules* for several synchronization strategies. The running strategy can be specified by the command line argument –sync, which receives one of the following values:

- `none` - default *module* that provides no synchronization;
- `lock` - a coarse-grain locking synchronization strategy;
- `finelock` - a fined-grain locking synchronization strategy;
- `jvstm` - Java Versioned STM (requires jvstm.jar or any other that implements the JVSTM API);
- `deuce` - DeuceSTM (requires one of the available versions of Deuce STM: e.g. deuceAgent-1.3.0.jar);
- `artof−free` - (requires artof.jar);
- `artof−lock` - (requires artof.jar);
- `boost` - an highly-concurrent transactional version of a linearizable implementation of *node* (requires artof.jar).

Finally, despite being a port of WormBench, the JWormbench has some key differences from the former:

1. The STM integration on WormBench is based on macros and requires pre-compilation specific for each STM API. The JWormBench uses an approach that is based on *inversion of control*, *abstract factory* and *factory method* design patterns.

2. The core engine of the JWormBench benchmark is deployed in a separate and independent library, whose features can be extended with other libraries. For instance, in WormBench is impossible to add new kinds of *worm operations* without modifying its source-code, whereas in JWormBench we can do it by implementing the new *operations* as separate classes and the corresponding *factory*.

3. Unlike JWormBench, the WormBench distribution does not implement the correctness test (i.e. sanity check for the STM system) based on the results accumulated on each thread’s private buffer.

4. In WormBench it is not easy to maintain the same contention scenario when varying the number of threads. In JWormBench the number of threads is totally decoupled from the environment specification and we can maintain the same conditions along different numbers of worker threads.

5. The operations generator tool in JWormBench allows us to specify the proportion between each kind of operation. This feature is essential to produce workloads with different ratios of update operations.