Here are some helpful debugging tips when you encounter with simulation failures.

1. Why I cannot reproduce failures reported by Joshua on my local machine?
    - Simulation tests are only reproducible on the same OS type, and with the same build environment. A binary from a CMake build in docker may not execute the same as a CMake build in the centos7 VM. Libstdc++ vs libc++ makes determinism change.
    - Make sure your test is deterministic by checking the unseed numbers are same across multiple runnings. (The seed initializes the random number generator for the test, and then the unseed is the last random number generated.)
    - Verify the `trace*.xml` is well-formatted. If you know your test is successful on your local machine, check the last `trace*.xml` file using command like `mono ./correctness/bin/TestHarness.exe summarize trace.0.0.0.0.0.1595012319.MJbTZt.0.1.xml summary.xml "" "" false`. See whether summary.xml has something the same as your Joshua report. (Maybe open the XML file in the browser directly also help.)
2. How to use `valgrind` to help diagnose memory problems?
    - Edit `CMakeCache.txt`. Set `USE_VALGRIND=ON` then recompile fdbserver. Then `valgrind --log-file=valg.log ./bin/fdbserver -r simulation -s 366751840 -f ../tests/fast/SwizzledRollbackSideband.toml` can work.
    - If illegal instruction error appears when you run `valgrind` (says `vex amd64->IR: unhandled instruction bytes...`), set `USE_AVX=OFF`.
    - You can also submit `packages/valgrind-7.0.0.tar.gz` to Joshua. It takes more time to finish one assemble.
3. How to choose a subset of tests to run on Joshua?
    - Edit `TEST_INCLUDE:STRING, TEST_EXCLUDE:STRING`. Notice that the regex doesn't have `.toml` suffix, and the path can be related to `${FDB_REPO_ROOT}/tests` (ex. `fast/.*`), or not (ex. `Swizzled.*` means all test files under subfolders started with Swizzled).
    - Only test spec files under `${FDB_REPO_ROOT}/tests/{rare,restarting,slow,fast}` will be choosen. The files under ./tests/ are some tests that used to work or be useful but not anymore.
4. How to get useful information in `trace*.xml`, considering there're too many lines?
    - Start grep Trace events from `fdbserver/tester.actor.cpp` and locate you failed in which test phase.
    - Function `waitForQuietDatabase` in `QuietDatabase.actor.app` also contains many useful trace message in simulation test.

5. Get you to know which line causes the problem.
   - use coredump files. (ex. `gdb ./bin/fdbserver core.17426`)
   - use addr2line. The problematic trace stack is logged in trace files and stdout. (ex. `addr2line -e ./bin/fdbserver -p -C -f -i 0x7fae93006630 0x26b5f98 0x26eff9b 0x26dc132 0x26df565 0x26dfaea 0x25c8798 0x25c4f45 0x25c20d8 0x25c1e0e 0x25c0118 0x25b816c 0x25bf888 0x25bf4ba 0x25bc968 0x25bc27a 0x1eb6148 0xf34538 0xf34316 0x287a431 0x287a1f1 0xfc6c78 0x28edc37 0x28eda96 0x28ed731 0x28edae2 0x28ee16c 0xfc6c78 0x29d1016 0x29c5bfe 0x28df844 0x13f3c18 0x7fae92745555`
6. A restart test failed. How do I reproduce it locally?
   - See https://github.com/apple/foundationdb/wiki/How-to-reproduce-a-restart-test-failure.
7. Try to set `TestConfig.simpleConfig=true` to make the fdb cluster as a small cluster to make debug easier.
8. `MutationTracking.h` and `MutationTracking.cpp` file contains useful functions to track all mutation activities of up to 2 keys in simulation test. Try to set the key you want to track, and `grep MutationTracking`.
9. Use `fdbserver --print-sim-time` to print out the simulation seconds if you experience a long `time_out` failure. Kill the simulation after you think the debug already happened after a certain simulation interval.
10. Use `DEBUG_ERROR` macro to enable backtrace when a specified error is created.
11. Use `tr.debugTransaction()` together with `debugRandom()` to enable the trace for a specified transaction without change the deterministic randomness of a simulation failure.
12. There are several compilation options can enable sanitizer utilities provided by `clang`. For example, `USE_ASAN`, `USE_TSAN`...