Push-button verification of file systems via crash refinement

Paper presented by: **Manjeet Dahiya**

Scribes compiled by: **Anmol Panda**

1. What is **Verification**?
    a. **Definition**: Formal verification involves proving or disproving the accuracy of a given program. Each program has a formal specification associated with it. The verification process determines if the program satisfies all the conditions set forth by its formal specification i.e if the program is equivalent to its specification.
    b. **Need**: File systems are a very critical service and their corruption can lead to significant data loss. Simulation and testing techniques do not capture all possible scenarios in the running of modern file systems and cannot be relied upon solely to certify that a given software is free of bugs. Formal verification and theorem provers are therefore required to guarantee file system accuracy.
    c. **Complications**: To verify that a program implementation meets its specification requirements, each input and every path of execution must be explored. This is particularly **difficult** and **time** consuming for **file systems** as they operate on large inputs (e.g. entire disks) and have many execution paths. Moreover, arbitrary crashes and reordering of writes by the disk cache add to non-determinism.
    d. **Solution**: Automate the process of verification using Yggdrasil by introducing a new definition of file system correctness: **crash refinement** (see definition on page 6 of paper).

2. Yggdrasil
    a. A toolkit for automated testing of file systems without any manual annotations or proofs about the implementation code. It produces a counterexample if the system has a bug.
    b. Using a new definition of file-system accuracy - crash refinement - Yggdrasil requires that the states in implementation and states obtained after a crash should be a subset of the specification. Thus each possible state, with or without crashes, should be a equivalent to some state in the specification. The crash-refinement based equivalence is formulated as an SMT problem and proved using Z3 solver.

c. The tool requires as inputs the following three things: specification (assumed to be correct), implementation and consistency invariants.
d. If the implementation passes the verifier, Yggdrasil will produce a working file system, else if it finds a bug it will give a counterexample that illustrates how the bug violates the crash refinement principle.

3. Crash refinement
    a. Given a specification S and an implementation F, F is correct with respect S if starting from the same state and invoking the same operations on both systems, any state produced by F is equivalent to some state produced by S.
    b. Main contribution of this paper. How is it modelled?
    c. Consider two variables that model the sequence of actions when a disk write is processed. The **ON** variable indicates if the data has been written to the buffer. The **SYNC** variable shows if the write has been committed to the disk.
    d.

Case	Status at time of crash		Result
	ON	SYNC	
1	0	0	Nothing to be written, no need for commit; State after crash equivalent to state before
2	1	1	Data written to buffer, committed to disk; State after crash equivalent to state before
3	1	0	Data written to buffer, not yet committed to disk; State after crash NOT equivalent to state before
4	0	1	Not relevant as there is no data to write (same as Case 1 [ON = 0, SYNC = 0]

    e. Crash refinement enables a layered structure for file systems, thus isolating proofs to a single layer. As higher layers use only the specifications (not the implementation) of lower layers, cross-layer verification is not needed.

4. Issue with verification of Yxv6 and how is it solved in Yggdrasil?
    a. Yggdrasil needs to obtain an SMT encoding of both the implementation and specification through symbolic execution, before using the Z3 solver to prove the crash refinement theorem.
    b. However, given the size and complexity of Yxv6, encoding the entire file system (specification + implementation) would make it impossible to verify the same using Z3 (or any other state-of-the-art SMT solvers)
    c. To solve this problem, Yggdrasil uses a three pronged approach:
        i. It divides the abstraction into five layers, each with its own specification and implementation, and applies crash refinement to each of them. The layer above uses only the specification (trusted, hence accurate) thus removing the need for cross-layer explosion.
        ii. To gain from locality of writes, the system uses multiple separate disks instead of one unified disk. The resulting file system is named Yxv6-sync. This way it can infer that a write to one disk does not affect the remaining portions of the file system, thus reducing the proof burden. This system is then proven to be a crash refinement of a system that uses a single disk.
        iii. To improve runtime performance of Yxv6-sync, multiple system calls are grouped into a single transaction and committed only when the log is full or upon fsync. The tool then proves the resulting filesystem, Yxv6-group_commit to be a crash refinement of Yxv6-sync.
    d. This way, by dividing the file system into layers it reduces the number of path explorations needed and isolates the reasoning to one layer. By using crash refinements of the original system, the proof burden is further reduces and performance is improved.

5. What are the difference between FSCQ and Yggdrasil?
    a. While their on-disk layouts are similar, Yxv6 uses uses an orphan inodes partition to manage files that are still open but unlinked to guarantee atomicity of *unlink* and *rename*. FSCQ does not have this guarantee.
    b. Unlike FSCQ, Yxv6 uses validation, as opposed to verification, when dealing with block or inode allocation, thus creating an allocator that is safe but does not guarantee block or inode allocation will succeed when there is enough space.

c. Using crash refinement to reduce proof burden and scale up verification and by harnessing the efficient decision procedures of Z3, theorems of Yxv6-sync could be proven in less than a minute. Coq takes 11 hours to do the same for FSCQ's proofs.
d. Crash refinement does not need advanced knowledge of program logistics, unlike FSCQ, making it suitable for SMT reasoning.
e. Yxv6-sync and Yxv6-group_commit both outperform FSCQ on a RAM disk due to the benefit of Yggdrasil's efficient Python-to-C compiler as opposed to FSCQ, which uses Haskell code extracted from Coq.

6. Difference between **specification** and **implementation**
    a. **Specification**: More abstract, acts like a blueprint, sets the requirements that the implementation must fulfill and the conditions that should be met.
    b. **Implementation**: More concrete and well defined, must satisfy the requirements and conditions set forth by the specification, can add anything that is not mentioned in the specification.
    c. Compiler guarantees that the implementation is equivalent to the program specification.

7. SMT Solver
    a. Theorem prover, determine whether a theorem is provable or not
    b. Eg. x^2 - y^2 = (x + y)*(x - y)

    How to prove this to be true? Two possible approaches
        i. Trivial approach. Given a finite domain, enumerate on all possible values of x and y. If the equation is satisfied for all x and y, then it holds true. This is called **bit blasting**. It can be very large in terms of number of computations and takes too long to run.
        ii. Better approach (human approach). Use **axioms**, properties of algebra such as associativity, commutativity, transitivity, idempotence, identities, etc. One can also use complex formulae that have been previously proven using SMT solvers
8. What is Hoare logic?
    a. A formal system to test the correctness of logical programs
    b. The specification consists of a pre-condition and a post-condition; the function relies upon the pre-condition to operate correctly and establishes the post-condition when it executes correctly.
    c. Using this model, Hoare defines a set of logical rules that when applied can verify the correctness of a given implementation with regards to its specification.

9. COQ
    a. An interactive theorem prover
    b. Provides a framework to define mathematical assertions, check the validity of those assertions (proof)
    c. Helps in finding formal proofs
    d. Extracts a certified program from the constructive proof of its formal specification

10. What are the meanings of and difference between **functional correctness** and **consistency requirement**?
    a. **Functional correctness**: Given an algorithm, if for each possible input, the algorithm produces the expected output, it is said to be functionally correct.

    Example: Given a set and its equivalent BST, if a new element is inserted in the set, the new set and the new BST should still be equivalent.

    b. **Consistency requirement**: Given any input, the program must not crash or exhibit behavior that goes against its definition.

    Example: Given any insertion into a BST, it must not create a cycle.

    c. In the case of Yggdrasil, the verifier expects a set of consistency invariants as input, along with the specification and implementation. If any of the consistency invariants are not satisfied by the implementation, the verifier will show a '**failed'** result.

11. Log-structured file system
    a. **Definition**: A file system in which data and metadata are written sequentially to a circular buffer.
    b. On the other hand, a UNIX file system maintains a tree-structured hierarchy of its directories with the super-block as the root and subsequent inodes and blocks linked to it. A pre-order traversal of this tree will give a log similar to that of an equivalent log-structured file system

    c. Modifications get appended to the head of the buffer and old data becomes garbage as no log is pointing to them. In a tree-structured file system (UNIX), the writes are made in place and hence the old data gets overwritten. In a log structured FS, the old data is not overwritten.

d. **Advantages**:
    i. **Sequential writes:** As system memories grow, more data can be cached. Consequently, the disk traffic consists mostly of writes as reads are serviced by the caches. Thus the nature of write accesses can have a huge impact on disk performance. Given the huge gap between random I/O performance and sequential I/O performance, the log-structured file system's sequential writes are advantageous.
    ii. **Crash recovery**: Crash recovery is comparatively very simple. When the filesystem is mounted after a crash, it can reconstruct its state from the last consistent state in the log. In the case of a UNIX-like file system, it would need to walk through the entire list of its data structures to fix any inconsistencies that the crash may have caused.
e. **Disadvantages**:
    i. **Garbage generation**: It creates too much garbage and may run out of disk space. A garbage collector should be run to find the garbage and clear that space.
    ii. **Memory fragmentation**: The log structured system, with its many invalidations of old data, creates 'holes' in the memory i.e. it leads to fragmentation of memory, unlike conventional (UNIX) file systems that keep files contiguous with in-place writes. This fragmentation can affect read performance adversely.

12. Why to worry about crashes in the specification?
    a. Bring the specification and implementation closer together without adding too much complexity. This makes **proofs** of equivalence easier.
13. Reordering of writes
    a. How are reorderings modeled? If w1 and w2 are writes in that order and both are in the state [1, 0] (i.e. ON = 1, SYNC = 0), then how does the model guarantee that the writes are committed in that order?
    b. Solution: Keep a convention. Until w1 makes a transition from (1, 0) to (1, 1), w2 cannot reach state (1, 0).

14. Why are file systems layered?
    a. Layer based approach is easier to understand and debug due to its modularity
    b. Much of the code is uniform across a variety of file systems as only certain layers need to be file system specific.
    c. Layer based approach isolates the proof requirement to a single layer.