CMPSCI 691ST Systems **Fall 2011**

Lecture 16

Lecturer: November 22 *Scribe: Hardeep Uppal*

16.1 Software Bugs

16.1.1 Heisenbugs

A Heisenbug is a computer bug that disappears or alters its characteristics when we try to apply standard debugging techniques to catch the error. One example is a bug that occurs in a program compiled for optimized execution (non-debugging mode), but not in the same program compiled to run under debug-mode. Executing program in debug-mode often cleans memory before the program starts, and can effect the condition of the bug.

16.1.2 Bohrbug

A Bohrbug is a bug that is caused consistently under a well-defined (but possibly unknown) set of operations and conditions. If a Bohrbug is present in a program, then retrying the operation that caused the bug will always result in the same bug. In case of a Heisenbug, the error could vanish on a retry. Hence Bohrbug is detectable by standard debugging techniques.

Bohrbug include bugs that are easy to detect and fix, but also includes bugs that are very hard to detect and only occur under certain conditions of the software. These kinds of bugs often live in parts of program that are invokes less often, which makes them remain undetected.

16.2 Rx

Rx is a technique for surviving software failures that can quickly recover programs from many types of software bugs that are both deterministic and nondeterministic. A challenge for all applications is to have high availability, even under failures. Bugs in a program can cause unexpected failures and can reduce availability.

The technique proposed by Rx is to create checkpoints as the program makes progress. When a failure occurs, the program is rollback to a recent checkpoint, and re-executed in a modified environment. The main idea behind this approach is an observation that many bugs are correlated to the environment in which they are executed and therefore running the program on a different environment could make the bug disappear. For example, if there is a race bug, then there is a good chance that the error disappears after re-executing the program.Rx does not guarantee than the error will occur or not and if a failure does occur, then the module is moved to a different environment.

How does Rx interact in a multi-threaded environment? Availability will be destroyed because of all the checkpoints generated by threads. Another issue is the size of the file generated by Rx to maintain check-points. It is hard to preserve the integrity of the file.

A simple solution to avoid checkpoints would be to restart the program whenever a failure occurs. Services are always replicated on multiple machine. If the service fails on a machine, then just restate the service on the same machine. In a user facing application this is not a case. For example, Microsoft Office uses a ad-hoc ways to save as much as of the state of the current environment before crashing.

16.2.1 Tradeoff between Fault Tolerance and Availability

Fail-Stop: As soon as the program detects a failure, the program execution is terminated. The main tradeoff here is the decrease in availability as programs fails. Java implements Fail-Stop technique for fault tolerance. For example, an array out of bound exception causes a program to stop. An idea presented in the class was to catch the array out of bound exception and reassign more heap to the array. This corresponds to the infinite heap model proposed in DieHard.

16.3 DieHard

DieHard is a runtime system that handles memory errors while probabilistically maintaining memory safety. Many application are vulnerable to buffer overflow, memory errors and dangling pointers, that causes programs to crash or behave incorrectly. DieHard tolerates these errors by randomization and replication. The memory manager in DieHard randomizes the location of objects in the heap. The idea here is that randomly placing objects in a large size heap prevents heap corruption and provides probabilistic guarantees of avoiding errors.

To add a second layer of security, DieHard can simultaneously run multiple replicas of the same application. Each replica is executed with a different random seed and consensus is made on output generated by each replica. Since it is unlikely to see the same errors on all replicas, DieHard increases the likelihood of an application executing correctly and without failures. Having replicas of a program allows DieHard to be used as a debugging tool by comparing the heap of correct and incorrect program executions.

DieHard only checks on system boundaries, which makes it fast. There is a tradeoff between memory usage and error tolerance. DieHard is useful for programs where memory footprint is less important than security and fault-tolerance. As memory gets bigger and cheaper, memory usage will become less of an issue. One issue pointed out in class is that there needs to be consensus across the replica on the time of the day. Since the logs use dateTime, different in dateTime across replicas can cause issues in logs.

The replication idea in DieHard is used for executing program written for the Space Shuttle program where a program is replicates on three different systems and consensus is made on each output. The software is written by three different venders. The idea being that errors are very much dependent on the specs of the programs, and programmers often make the bugs in hard functions.

16.4 Fault Tolerance in Windows

Randomization of object placement in heap has been added in Windows 8 for added security. The OS measures the crash velocity of each program. When the crash velocity passes a certain threshold, fault tolerant heap is triggered. The fault tolerant heap is a poor man's DieHard. It has a separate area for maintaining the metadata and also does a number of checks on the heap.

16.5 Finding Errors from Logs

Debugging programs for bugs is really hard since bugs detected in test can be different than bugs detected in the real environment. Hence the idea is that programs generate large amount of logs, and these log entries can be analyzed to find errors. One issue with this approach is the size of the log files can be very large. Also looking at the log entries and correlating it back to the code can be hard.