Detflow: towards deterministic workflows on your favorite OS

Ryan Scott[1] Ryan Newton[1]
Omar Navarro-Leija[2] Joe Devietti[2]

[1]Indiana University [2]University of Pennsylvania

github.com/RyanGlScott
rgscott@indiana.edu

March 24, 2017

Software is nondeterministic.

Software can give different answers

```
ryanglscott at Linux-T450 in ~
$ date
Mon Mar 20 21:03:57 EDT 2017
ryanglscott at Linux-T450 in ~
$ date
Mon Mar 20 21:03:59 EDT 2017
ryanglscott at Linux-T450 in ~
$ grep -m1 -ao '[0-9]' /dev/urandom | sed s/0/10/ | head -n1
9
ryanglscott at Linux-T450 in ~
$ grep -m1 -ao '[0-9]' /dev/urandom | sed s/0/10/ | head -n1
7
```

Software runs differently on different machines

moroshko commented on Jan 2 — Owner

Weird... I can't reproduce the issue on my side.
Could you try to create a minimal Codepen please that demonstrates the issue?

ngarnier commented on Feb 7 • edited — Member

Hi **@Sarah-IFG**, thanks for reporting this issue.

Unfortunately, I can't reproduce the issue with `NaN`, can you provide your MJML markup?

fzaninotto commented on Jan 19 — Member

Can't reproduce the issue, which version of admin-on-rest are you using?

Software is subject to nondeterministic concurrency

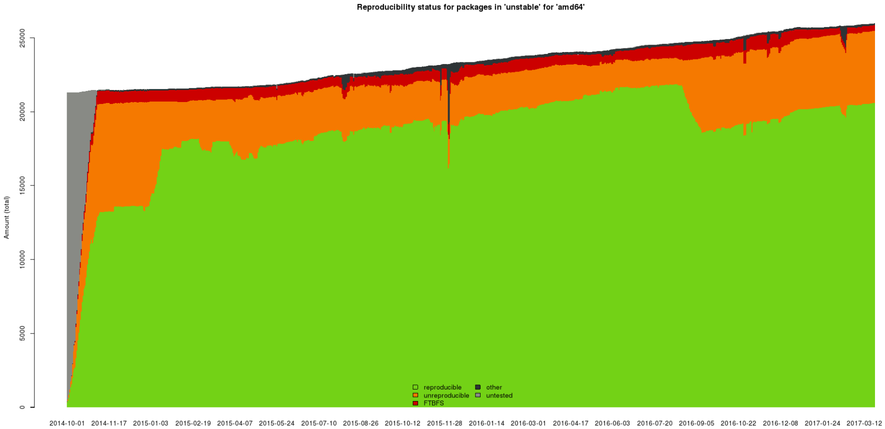
```
ryanglscott at Linux-T450 in ~/Documents/Hacking/Haskell
$ runghc ParHello.hs
HeHHHleHeellellollll,oloo ,o,,W ,  oW WWroWoolrorrdlrll!dldd
!d!!
!


ryanglscott at Linux-T450 in ~/Documents/Hacking/Haskell
$ runghc ParHello.hs
HHeHeHlHelelelllollol,lo,o o, ,W, W o WoWrWorolorlrdrldl!ld!d
d!
!!


ryanglscott at Linux-T450 in ~/Documents/Hacking/Haskell
$ runghc ParHello.hs
HeHHHHleeeelllllollll,oooo ,,,,W    oWWWWroooolrrrrdllll!dddd
!!!!
```

How do we wrangle the nondeterminism?

Debian Reproducible Builds



A fully deterministic OS?

dedis@yale   Determinator   Dissent   EverCloud   Tng

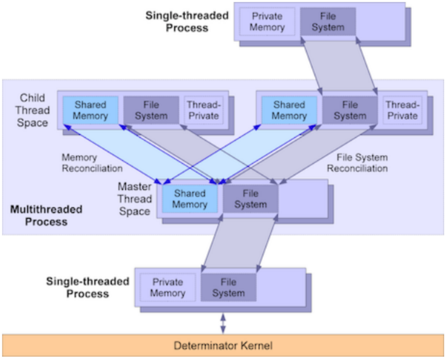
Determinator an operating system for deterministic parallel computing

Background

Determinator is an experimental multiprocessor, distributed OS that creates an environment in which anything an application computes is exactly repeatable. It consists of a microkernel and a set of user-space runtime libraries and applications. The microkernel provides a minimal API and execution environment, supporting a hierarchy of "shared-nothing" address spaces that can execute in parallel, but enforcing the guarantee that these spaces evolve and interact deterministically. Atop this minimal environment, Determinator's user-space runtime library uses distributed systems techniques to emulate familiar shared-state abstractions such as Unix processes, global file systems, and shared memory multithreading.

A subset of Determinator comprises PIOS ("Parallel Instructional Operating System"), a teaching OS derived from and providing a course framework similar to JOS, where students fill in missing pieces of a reference skeleton. Determinator/PIOS represents a complete redesign and rewrite of the core components of JOS. To our knowledge PIOS is the first instructional OS to include and emphasize increasingly important parallel/multicore and distributed OS programming practices in an undergraduate-level OS course. It was used to teach CS422: Operating Systems at Yale in Spring 2010, and is freely available for use and adaptation by others.

Determinator will also provide a starting point for a certified OS kernel project in collaboration with the FLINT research group.



A multithreaded process built from one space per thread, with a master space managing synchronization and memory reconciliation

A fully deterministic OS?

Statically enforced determinism

Idea: enforce determinism *statically* through your language.

Statically deterministic parallelism

LVars: Lattice-based Data Structures for Deterministic Parallelism

Lindsey Kuper   Ryan R. Newton
Indiana University
{lkuper, rrnewton}@cs.indiana.edu

Freeze After Writing

Quasi-Deterministic Parallel Programming with LVars

Lindsey Kuper, Indiana University, lkuper@cs.indiana.edu
Aaron Turon, MPI-SWS, turon@mpi-sws.org
Neelakantan R. Krishnaswami, University of Birmingham, N.Krishnaswami@cs.bham.ac.uk
Ryan R. Newton, Indiana University, rrnewton@cs.indiana.edu

Taming the Parallel Effect Zoo

Extensible Deterministic Parallelism with LVish

Lindsey Kuper   Aaron Todd   Sam Tobin-Hochstadt   Ryan R. Newton
Indiana University
{lkuper, toddaaro, samth, rrnewton}@cs.indiana.edu

Don't allow users to shoot themselves in the foot

- Restricted IO (RIO)

```haskell
newtype DetIO a = DetIO (IO a) -- exported abstractly
readFile :: FilePath -> DetIO Text
writeFile :: FilePath -> Text -> DetIO ()
-- etc.
```

- All programs must live in `DetIO`

```haskell
main :: DetIO ()
main = ...
```

Example usage

`detflow in/ out/ Main.hs`

- Run in environment with fixed dependencies
- Use `hashdeep` to verify determinism

Why Haskell?

- Most of these techniques could be ported to any language
- A purely functional language that controls side effects is far easier to manage, though!
  - We need only worry about the determinism for `DetIO`—pure computations are always deterministic

API design questions

- What would a function that returns time look like?

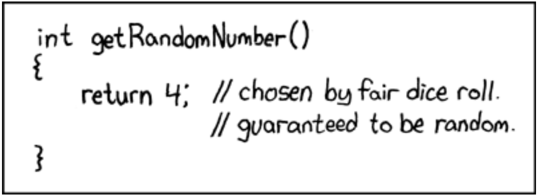
  ```
  getTime :: DetIO Time
  ```
- Can't rely on system clock!
- Could use deterministic, logical clock
  - Progress is counted by number of stores retired
- Could also return the same `Time` every time, but...

API design questions

- What would a function that returns a “random” number look like?

  ```
  getRandomNumber :: DetIO Int
  ```

- One option...



- Watch out for entropy!

What don't we allow?

- Arbitrary IO effects

  `liftIOToDetIO :: IO a -> DetIO a`
- Workaround: Don't allow them in `{-# LANGUAGE Safe #-}` code

What don't we allow?

- Unrestricted memory accesses

```
readFile :: FilePath -> DetIO Text
writeFile :: FilePath -> Text -> DetIO ()
```

- Easy to end up with race conditions

Thread 1

```
writeFile "foo.txt"
          "Hello, World"
```

Thread 2

```
do foo <- readFile "foo.txt"
   if foo == "Hello, World"
     then ...
     else ...
```

What don't we allow?: unrestricted memory accesses

- Solution: fine-grained, thread-level permissions

```
/abcdefg/hijklmn/opqrstu
```

/abcdefg	/hijklmn	/opqrstu
Thread 1: R	Thread 1: R	Thread 1: RW
Thread 2: R	Thread 2: R	Thread 2:

- Read (R): Ability to read directory contents
- Read-Write (RW): Ability to read/modify directory contents, and delete the directory

Key idea

If a thread has a RW permission on a path, no other thread retains permission on it.

What don't we allow?: unrestricted memory accesses

- Design API around these permissions

  ```
  forkWithPerms :: [PathPerm] -> DetIO a -> DetIO (Child a)
  ```

- If the forked computation requests permission to write a path, the parent must *relinquish* its own permission to do so.

What don't we allow?: unrestricted memory accesses

- What about symbolic links?
  - Not accounted for in our model of paths
  - Treating them properly would require dealing with aliasing
- For now, we disallow symlinks

What about legacy software?

What about legacy software?

- We'd like to be able to shell out to applications not written in `DetIO`
- How do we retain determinism while doing so?

What about legacy software?

Run legacy applications in a deterministic runtime.

Counteracting external sources of nondeterminism

- The deterministic runtime must be resilient against many different things in a *worker process*:
  - Special directories: `/proc`, `/dev/random`
  - Nondeterministic instructions: `rdtsc`, `cpuid`, `rdrand`
  - Reading system time
  - Concurrency (can lead to races!)
  - Address-space layout randomization (ASLR)

Counteracting external sources of nondeterminism

- "Determinizing" OS-level operations requires some way to intercept them
- Possible solutions:
  - `LD_PRELOAD`
  - `ptrace`
  - Hypervisors

Counteracting external sources of nondeterminism

- Obtaining a deterministic runtime for worker processes might include:
  - Disallowing "exotic" process execution (e.g., background processes)
  - Running everything sequentially (i.e., intercept `pthread_create`)
  - Intercepting naughty library calls/system calls whenever possible
  - Passing path permissions from the `DetIO` program to the runtime

Use case: `fread` and `fwrite`

- From the manpage for `fread`:
  "On success, `fread()` and `fwrite()` return the number of items read or written. This number equals the number of bytes transferred only when size is 1. If an error occurs, or the end of the file is reached, the return value is a **short item count** (or zero)."

Use case: fread and fwrite

- Using the LD_PRELOAD trick:

```
size_t fread(void *ptr, size_t size,
             size_t nmemb, FILE *stream) {
  printf("Running deterministic version of fread...\n");
  FILE* (*originalFread)(const char*, const char*);
  originalFopen = dlsym(RTLD_NEXT, "fread");

  ssize_t actual_bytes
    = (*originalFread)(ptr, size, nmemb, stream);
  if (actual_bytes != /* requested bytes */) {
    /* Keep reading... */
  }

  return /* requested bytes */;
}
```

Case study: deterministic make

- The make build tool is known to suffer from race conditions when ran in parallel

```
bin_PROGRAMS = multicall

install-exec-local:
        cd $(DESTDIR)/$(bindir) && \
                $(LN_S) multicall command1 && \
                $(LN_S) multicall command2
```

Case study: deterministic `make`

- Solution: let's make our own `make`!
- Dynamic enforcement of path permissions *forces* us to declare dependencies correctly

Case study: deterministic make

- Pseudocode

```
main :: DetIO ()
main = do
  forkWithPerms [{- Perms -}]
    (detsystem "gcc" [ "file" ++ show n ++ ".c"
                     , "-o"
                     , "file" ++ show n ++ ".o"
                     ])
  wait
  detsystem "gcc" ( ["-o", "main"] ++
                    map (\n -> "file" ++ show n ++ ".o")
                        files )
```

Takeaways

- The first system to use a hybrid approach of static and dynamic determinism enforcement
- Write deterministic code in `DetIO` while still retaining the ability to run legacy code deterministically
- Combine the strengths of Haskell with a deterministic runtime
- Not much extra overhead (hopefully!)

Any questions?