

GemDisk: a GEOM Class to Emulate Disk Drives

Fabio Checconi and Luigi Rizzo

September 19, 2009

Outline

- ① Introduction
- ② Modeling and Simulation
- ③ Implementation
- ④ Usage/Evaluation

Main Goal

Allow the creation of virtual devices, on a system running on bare metal, with well-defined configurable timing behavior.

Motivation: GEOM Schedulers

GemDisk is a tool born to support the geom_sched project.

Testing schedulers requires access to a lot of hardware: where we cannot get with the money we try to get with imagination...

NOTE: emulation cannot replace testing/benchmarking on real hardware.

What Needs to be Tested

- ▶ No data accesses (microbenchmarks);
- ▶ command queueing;
- ▶ SSDs;
- ▶ robustness of the heuristics;
- ▶ HW/SW RAID.

Other Usage Scenarios

- ▶ Firmware modifications;
- ▶ RAID balancing algorithms or layouts;
- ▶ Media internal parameters.

What Others Do

- ▶ Full system emulators;
- ▶ simulators;
- ▶ Memulator.

Our Solution

Use `geom_gate` to forward requests to userspace, and use `DiskSim` to control the timing.

Geom Gates

- ▶ Receive requests like any regular provider;
- ▶ pass them to userspace with a chardev interface;
- ▶ wait for userspace to reinject them into the kernel;
- ▶ deliver them to their consumers.

Usage Scenarios

Among others:

- ▶ GEOM-ify non-geom devices;
- ▶ access remote block devices;
- ▶ our case, virtual devices controlled in userspace.

DiskSim

- ▶ De-facto research standard;
- ▶ detailed disk modeling;
- ▶ accurate;
- ▶ everything is configurable;
- ▶ SSD and MEMS models available.

Usage Scenarios

Too many to list, among others:

- ▶ Prototyping;
- ▶ validation of algorithms;
- ▶ evaluation of firmware features;
- ▶ evaluation of hardware features.

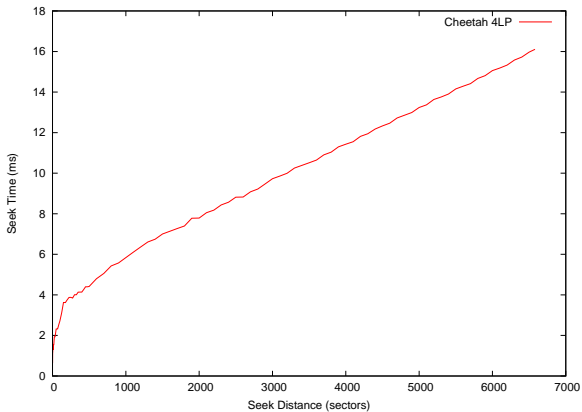
Disk Modeling

- ▶ Seek times;
- ▶ rotational delays;
- ▶ transfer times;
- ▶ caching effects;
- ▶ controller/bus effects.

Seek Times

$$\text{seekTime}(d) = \begin{cases} T_0 & \text{if } d = 1 \\ T_1 + T_2\sqrt{d} & \text{if } d < D \\ T_3 + T_4d & \text{if } d \geq D. \end{cases}$$

An Ancient Seek Profile



Rotational and Transfer Delays

- ▶ The position of the disk is tracked throughout simulation;
- ▶ transfer times grow linearly with the request size.

Caches

Too many parameters to list, among others:

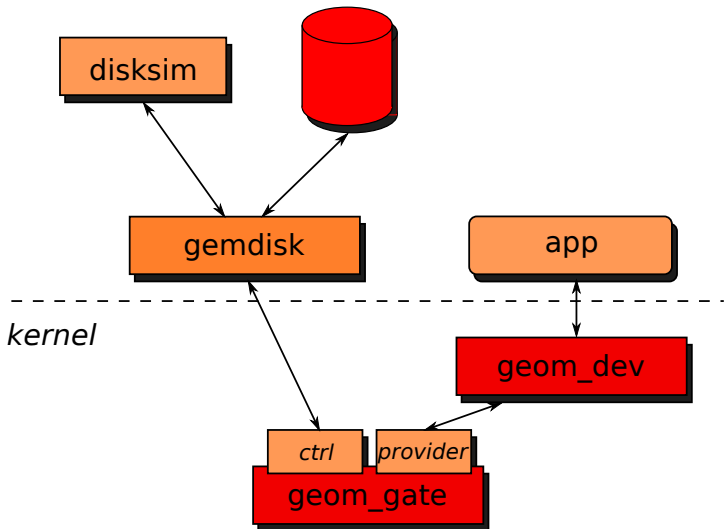
- ▶ Amount of prefetching/caching;
- ▶ prefetch/refill policies;
- ▶ cache partitioning (segments, read vs. write);
- ▶ write coalescing;
- ▶ reads/writes on arrival.

Parameter Extraction

Send sequences of commands to the drive and analyze how it responds.

DIXTrac is a parameter extraction tool that comes with the DiskSim distribution.

GemDisk Architecture



Data Structures

```
struct request {
    struct g_gate_ctl_io ggio;
    struct aiocb aiocb;
    struct disksim_request dsim;
    int flags;
    int error;
    TAILQ_ENTRY(request) qlink;
};

struct device {
    uint32_t unit;
    char path[MAX_BPATH];
};
```

Threads

- ▶ A main (simulation) thread;
- ▶ one “gatekeeper” thread per virtual device;
- ▶ a “reaper” thread.

Main Thread

- ▶ Initializes everything;
- ▶ enters the simulation loop:
 1. passes all the available requests to DiskSim;
 2. runs DiskSim until simulation time reaches the current time;
 3. waits for a new request or the next simulation event, whatever comes first;
 4. restarts from 1.

Gatekeeper Threads

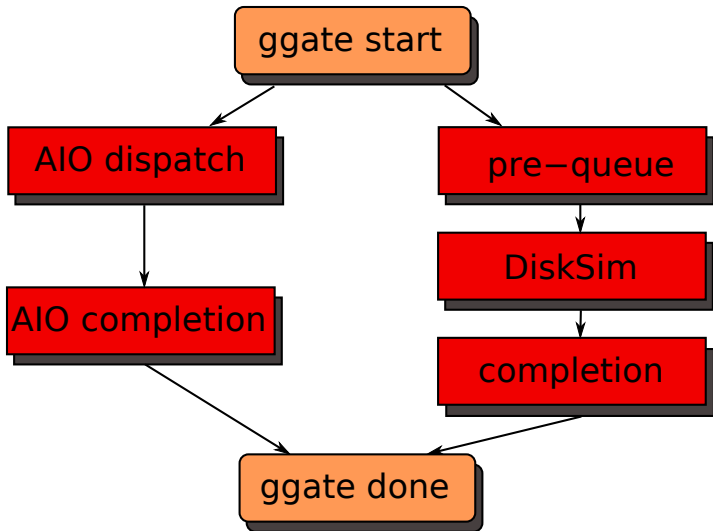
1. Waits for requests to come from the gate device;
2. when a request arrives enqueues it for the simulator and issues the I/O on the backing store (if needed);
3. restarts from 1.

Reaper Thread and Request Completion

Waits for the completion of I/O on any issued request.

When both the reaper and DiskSim have notified the completion of a request, that request is passed down to `geom_gate`.

Life of a Request



Flowing of Time

When there is no I/O or I/O is fast enough simulation time grows with real time.

Otherwise simulation time grows slower than real time, and applications may see incorrect completion times.

Our Solution

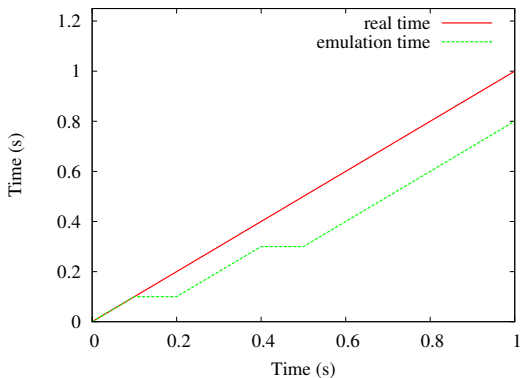
Slow down the time perceived by applications using virtualized devices.

Whenever a request is completed by the simulator but its data is not ready simulated time stops.

NOTE: We need to support that, not to limit too much the size of emulated devices.

Implementation

We hijack the timing-related syscalls, controlling them with simulation time.



Implementation (2)

We use `LD_PRELOAD` to “steal” the `libc` calls, and we used a shared memory area to make the clients (the applications under test) read the time from `GemDisk`.

We pass only a delta value to allow time to grow at the same rate as real time.

Running Microbenchmarks

- ▶ Usually do not access the data they read/write (no I/O required);
- ▶ arbitrary device sizes without timing tricks.

Filesystems and SSDs

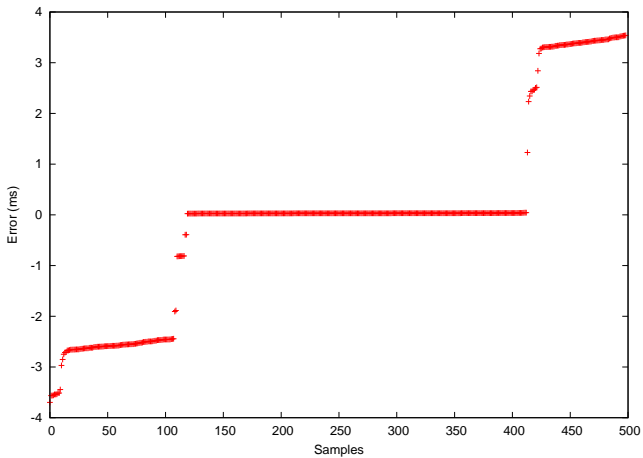
They need to access data, so GemDisk needs RAM-backing store or real disk storage, depending on device size.

Emulating SSDs requires “time stealing,” as the backing store is unlikely to be as fast as them.

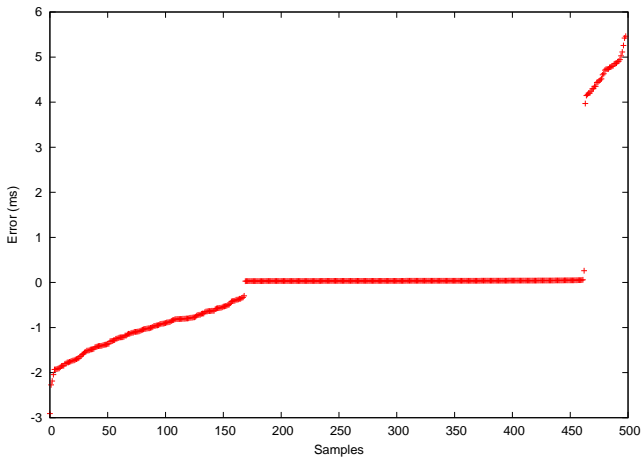
Evaluation

- ▶ Atom-based EEEPC;
- ▶ SATA WD3200 disk (16MB cache, 7200RPM);
- ▶ 500 synthesized requests, size in $[2KiB, 130KiB]$,
 - ▶ random: no sequential accesses;
 - ▶ mixed: mix of sequential, local and random accesses.

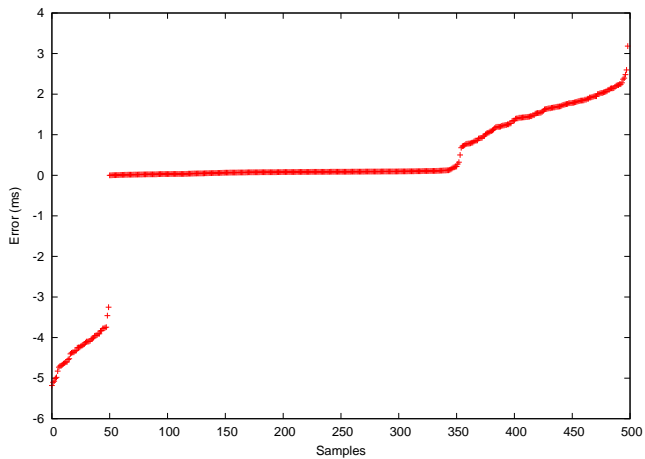
Results: Emulation Error



Results: "Stolen Time" Error



Results: User Perceived Error



Strengths

- ▶ Can emulate arbitrary hardware, either existent or non-existent;
- ▶ accurate simulation;
- ▶ arbitrary device sizes;
- ▶ simple to use (just follow DiskSim documentation and run a command).

Weaknesses

- ▶ The “time stealing” mechanism loses some of the emulation advantages and may produce inaccuracies with deep submit queues;
- ▶ impossible to use in distributed environments;
- ▶ the CPU load of the emulator may require to run it on a separate machine.

Thank You!

Questions?