Persistent Scripting

Zi Fan Tan Jianan Li Haris Volos Terence Kelly

hvolos01@ucy.ac.cy tpkelly@eecs.umich.edu

NVM Workshop
UC San Diego

9 May 2022

NVM & Low-Level Languages

- Mainstream NVM Programming: C/C++
  - machine efficiency
  - hardware control (cache line flushes)
- Downside: programmer efficiency

Scripting

- Convenient
- Concise
- Productive

- Persistence?

Persistence for Scripting

- Last stronghold of undersimplification
- Several options in Python, Perl
  - manual
  - per-variable fuss
- External checkpoint-restore (CRIU, DMTCP)
  - wrong transparency

Example: Log File Processing

```
{           # executes once per input line
  if ( ! ($0 in id) )   # assign numeric IDs to
      id[$0] = ++n;     #   unique strings
  freq[$0]++;           # count string frequencies
}
END {       # executes after all input processed
  print n;          # number of unique strings
  for (s in id)   # print table of IDs & frequencies
      print id[s], s, freq[s];
}
```

Incremental processing $\Rightarrow$ persistence

Manual Persistence

```
BEGIN {  # executes before first input line is read
  getline n < "summary";
  while (0 < (getline < "summary")) {
      id[$2] = $1;  freq[$2] = $3;  }
}
```

Persistent Scripting Done Right

- Interface: Interpreter remembers variables across runs
  - new "`--persist=heapfile`" flag
- Implementation: Slide persistent heap beneath interpreter
- Benefits
  - effortless persistence: scripts remain oblivious
  - share persistent variables between unrelated scripts
  - Big Data

pm-gawk: Persistent Memory gawk

- Slide persistent heap beneath gawk interpreter
- Under 100 LOC added/changed out of 91,000 LOC
  - add new `--persist` flag (easy)
  - `#define malloc pma_malloc` etc. (easy)
  - gawk symbol table $\Longleftrightarrow$ pma root pointer (not too hard)
- https://github.com/ucy-coast/pmgawk
- https://coast.cs.ucy.ac.cy/projects/pmgawk/

pm-gawk in Action

```
$ truncate -s 409600 heap.pma

$ gawk --persist=heap.pma 'BEGIN{myvar = 47}'

$ gawk --persist=heap.pma 'BEGIN{print myvar}'

47
```

Digression: Why `gawk`?

- Relatively simple
- Lightly guarded
- Innovations permitted in interpreter
- Maintainer answers e-mail

Foundation: `pma`

- Least-imaginatively-named persistent memory allocator
- Runs on conventional hardware; NVM not required
- `malloc`, `calloc`, `realloc`, `free`
- `init`
- `get_root`/`set_root`
- https://queue.acm.org/DrillBits7/

Crash Tolerance

- Usual commonsense precautions for scripting
- Make backups of important files
  - "`cp --reflink heap.pma heap.bak ; sync`"
- Distinguish successful completion vs. interruption
- Re-run jobs interrupted by failures

Performance: Hardware

- Cascade Lake 2.1 GHz
- 20 cores, 40 threads (irrelevant; `gawk` is serial)
- DRAM: 64 GB
- NVM: 256 GB Optane PM Series 100
- SSD: 960 GB SATA, 6 GB/sec

Performance: Workload

- Incremental log processing w/ AWK script
- 100 simulated days, measure performance on last day
- total 1 billion random strings
- non-stationary distribution, mimics "hot set drift"
- report `write` and `sync` times separately
  - `sync` off critical path of data analysis

Performance: Contestants

- (N) Naïvely read all 100 logs on day 100
- (B) BEGIN block implements manual incremental processing
- (P) `pm-gawk`, varying media beneath `pma` persistent heap:
  - DRAM (`/dev/shm`)
  - Optane configured as block storage
  - SSD block storage
  - Optane DAX mode
- All outputs (daily summary reports) written to SSD

Performance: Results

test	time (sec) run	time (sec) `sync`	time (sec) total	speedup vs. N run	speedup vs. N total
N (naïve)	669.43	1.50	670.93	1.00	1.00
B (`BEGIN`)	49.17	1.51	50.68	13.62	13.24
P `/dev/shm/`	53.58	1.51	55.09	12.49	12.18
P Optane block	58.68	23.54	82.22	11.41	8.16
P SSD block	58.77	43.93	102.71	11.39	6.53
P Optane DAX	174.81	3.15	177.96	3.83	3.77

Performance: Results

test	time (sec) run	time (sec) `sync`	time (sec) total	speedup vs. N run	speedup vs. N total
N (naïve)	669.43	1.50	670.93	1.00	1.00
B (`BEGIN`)	49.17	1.51	50.68	13.62	13.24
P `/dev/shm/`	53.58	1.51	55.09	12.49	12.18
P Optane block	58.68	23.54	82.22	11.41	8.16
P SSD block	58.77	43.93	102.71	11.39	6.53
P Optane DAX	174.81	3.15	177.96	3.83	3.77

Performance: Results

test	time (sec) run	time (sec) `sync`	time (sec) total	speedup vs. N run	speedup vs. N total
N (naïve)	669.43	1.50	670.93	1.00	1.00
B (`BEGIN`)	49.17	1.51	50.68	13.62	13.24
P `/dev/shm/`	53.58	1.51	55.09	12.49	12.18
P Optane block	58.68	23.54	82.22	11.41	8.16
P SSD block	58.77	43.93	102.71	11.39	6.53
P Optane DAX	174.81	3.15	177.96	3.83	3.77

Performance: Results

test	time (sec) run	time (sec) `sync`	time (sec) total	speedup vs. N run	speedup vs. N total
N (naïve)	669.43	1.50	670.93	1.00	1.00
B (`BEGIN`)	49.17	1.51	50.68	13.62	13.24
P `/dev/shm/`	53.58	1.51	55.09	12.49	12.18
P Optane block	58.68	23.54	82.22	11.41	8.16
P SSD block	58.77	43.93	102.71	11.39	6.53
P Optane DAX	174.81	3.15	177.96	3.83	3.77

Performance: Results

test	time (sec) run	time (sec) `sync`	time (sec) total	speedup vs. N run	speedup vs. N total
N (naïve)	669.43	1.50	670.93	1.00	1.00
B (`BEGIN`)	49.17	1.51	50.68	13.62	13.24
P `/dev/shm/`	53.58	1.51	55.09	12.49	12.18
P Optane block	58.68	23.54	82.22	11.41	8.16
P SSD block	58.77	43.93	102.71	11.39	6.53
P Optane DAX	174.81	3.15	177.96	3.83	3.77

Performance: Mystery

test	time (sec) run	time (sec) `sync`	time (sec) total	speedup vs. N run	speedup vs. N total
N (naïve)	669.43	1.50	670.93	1.00	1.00
B (`BEGIN`)	49.17	1.51	50.68	13.62	13.24
P `/dev/shm/`	53.58	1.51	55.09	12.49	12.18
P Optane block	58.68	23.54	82.22	11.41	8.16
P SSD block	58.77	43.93	102.71	11.39	6.53
P Optane DAX	174.81	3.15	177.96	3.83	3.77

Dirty Pages *Not* Dirt Cheap

```
{           # executes once per input line
  if ( ! ($0 in id) )   # assign numeric IDs to
      id[$0] = ++n;     #   unique strings
  freq[$0]++;           # count string frequencies
}
END {       # executes after all input processed
  print n;          # number of unique strings
  for (s in id)     # print table of IDs & frequencies
      print id[s], s, freq[s];
}
```

Summary

- Scripting should be easy and productive, but isn't
- Solution: interpreter aware, scripts oblivious
- `malloc`-compatible persistent heap makes it easy
- Reducing gawk STORE`s would reduce overheads
- https://github.com/ucy-coast/pmgawk
- https://queue.acm.org/DrillBits7/