Persistent Scripting

Zi Fan Tan Jianan Li Haris Volos Terence Kelly
hvolos01@ucy.ac.cy tpkelly@eecs.umich.edu

SNIA SDC 14 September 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

- Interpreter remembers script variables across runs
- Interface: specify *heap file* where variables live
- Implementation: Slide persistent heap beneath interpreter
- Benefits
  - effortless persistence: scripts remain oblivious
  - share persistent variables between unrelated scripts
  - de-couple data ingestion from analytic queries
  - $O(1)$ associative array lookups

Persistent Scripting Done Right: Two Implementations

- Prototype based on fork of GNU AWK (`gawk`) 5.1
- Re-implementation in official `gawk` 5.2
- Same persistent memory allocator, `pma`

pm-gawk: Persistent Memory gawk [prototype]

- 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 [prototype]

```
$ truncate -s 409600 heap.pma

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

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

47
```

Why gawk?

- Relatively simple
- Lightly guarded
- Innovations permitted in interpreter
- Maintainer answers e-mail
- It worked! `pm-gawk` ships in `gawk` 5.2

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/

https://queue.acm.org/detail.cfm?id=3534855

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
- *Persistent Memory `gawk` User Manual*

Performance: Hardware [prototype]

- 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 [prototype]

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

Performance: Contestants [prototype]

- (N) Naïvely read all 100 logs on day 100
  - non-incremental
  - appallingly inefficient
- (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

[prototype]

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

[prototype]

test	time (sec) run	sync	total	speedup vs. N run	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

[prototype]

test	time (sec) run	sync	total	speedup vs. N run	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

[prototype]

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

[prototype]

	time (sec)			speedup vs. N	
test	run	sync	total	run	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

[prototype]

test	time (sec) run	sync	total	speedup vs. N run	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];
}
```

pm-gawk in Official gawk 5.2

- Re-implementation by gawk maintainer
- New interface
- Same persistent variables/data as prototype
- Also persistent *functions*
- Same persistent memory allocator (pma)
- Additional performance evaluations
- *Persistent Memory gawk User Manual*

New Interface


```
$ truncate -s 4096000 heap.pma              # create heap file

$ export GAWK_PERSIST_FILE=heap.pma         # envar

$ gawk 'BEGIN{myvar = 47}'                    # script #1

$ gawk 'BEGIN{myvar += 7; print myvar}'       # script #2
54
```

Ergonomics


```
$ alias pm='GAWK_PERSIST_FILE=heap.pma'

$ pm gawk 'BEGIN{print ++myvar}'
55

$ pm gawk 'BEGIN{print ++myvar}'
56
```

Persistent Functions

[official gawk]

```
$ truncate -s 10M funcs.pma
$ export GAWK_PERSIST_FILE=funcs.pma

$ gawk 'function count(A,t) { for(i in A) t++; return 0+t }

$ gawk 'BEGIN { for (i=0; i<47; i++) a[i]=i }'

$ gawk 'BEGIN { print count(a) }'
47
```

Performance Again [official `gawk`]

- Text analysis
  - W unique words, N words in corpus
  - $N >> W$
- One script reads words into associative array, saves array
- Second script serves word-frequency queries
- Compare against manual frequency table, `rwarray` extension
- All require $O(N)$ time to ingest corpus
- But how long to serve queries?

Performance Results

[official `gawk`]

	asymptotic running time	measured time (s)	peak mem (KiB)	storage (KiB)
INGEST				
manual	$O(N)$	288.408	2,400,120	69,112
`rwarray`	$O(N)$	250.288	2,846,868	156,832
`pm-gawk`	$O(N)$	251.946	2,079,520	2,076,608
QUERY				
manual	$O(W)$	11.624	2,336,616	
`rwarray`	$O(W)$	11.653	2,081,444	
`pm-gawk`	$O(1)$	0.026	3,252	

Summary

- Scripting is easy & productive, except for persistence
- Solution: interpreter aware, scripts oblivious
- `malloc`-compatible persistent heap makes it easy
- `pm-gawk` is transparent & no-fuss
- De-couple data ingest from data analysis (learning/inference)
- Fast: $O(1)$ persistent array lookups
- https://queue.acm.org/detail.cfm?id=3534855
- https://ftp.gnu.org/gnu/gawk/gawk-5.2.0.tar.xz
- *Persistent Memory `gawk` User Manual*