SAND2018-13437C

Bytes are Bytes, Right?

Files, objects, and key/values in HPC

Matthew L. Curry, Ph.D.
Center for Computation Research
Sandia National Laboratories

CHPC National Meeting 2018

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

POSIX File I/O is Awesome*

awesome [aw-suh m], *adj.*

2. *Informal* -- extremely good; excellent.

Available for every operating system, with excellent language support across the board.

Easy-to-use, orthogonal foundational interface

Operating system support gives rich features

- OS-managed victim cache

POSIX File I/O is Awesome*

awesome [aw-suh m], *adj.*

1. extremely impressive or daunting; inspiring great admiration, **apprehension, or fear.**
2. *Informal* -- extremely good; excellent.

The POSIX file abstraction was present in the first version of UNIX (released in 1971), and remains largely unchanged at its core

- Extensions abound, and are unevenly supported

Low level interface: Flat address space within a file – Demands:

- File formats
- Serialization
- Care – Durability is a whole other talk.

And yet, underspecified.

Parallel File Systems are Awesome*

Tough to map semantics to performance

- Applications often get less than half of expected performance
- Analytic workloads get even less performance

POSIX was designed for serial processing

Files are physically associated with shared resources

- Weather is important, but no forecasting
- Good strategy requires knowledge of machine

Alternative I/O Facilities Gives Powerful Tools

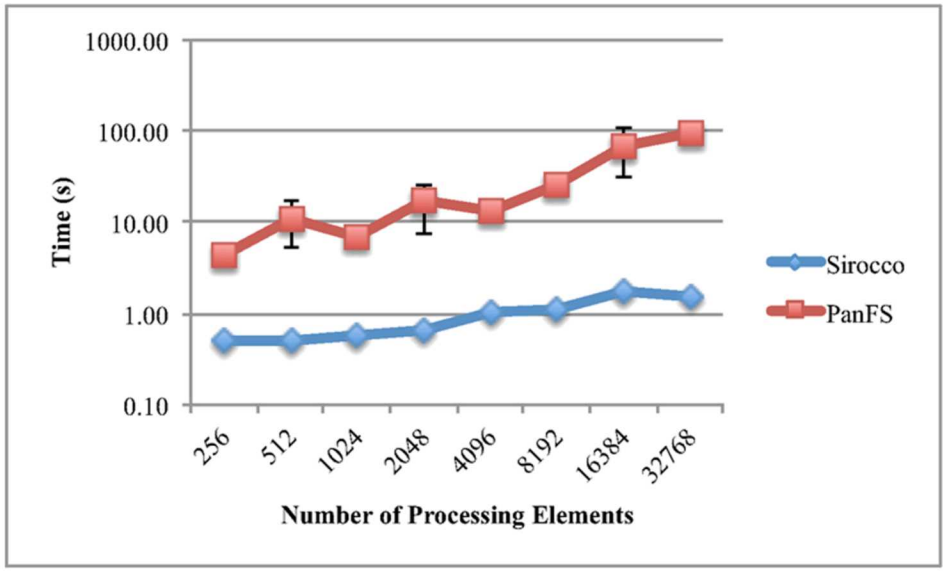
Buffering allows your storage to weather storms more easily

Transactional semantics gives you an easy way to reason about failures

- Hardware or software failure

Using compute nodes to provide or proxy storage can give extremely good performance!

Trade performance of uncommon operations for common ones



https://github.com/matthewcurry/sirocco-release

Objects

Examples: S3, Swift (from OpenStack), Sirocco

- Easy to move into cloud providers

Generally immutable, but appendable

Not files, but can be file-like

- Can be quite large
- Allows iteration
- Not often divorced from underlying storage

Typical use: "Documents"

- Images, videos, HTML

Extremely cacheable

Keys and Values

Can be simpler than objects at every level
- Atomic set and get

… but it can be tempting to layer extra sugar on top
- Searches/queries

… or leave out very obvious features
- Iteration

Can often be used in memory only!
- Many flavors of persistence in advanced packages

Highly granular, small chunks of data
- Rule of thumb: 1MB maximum

Often embeddable into application

Examples: memcached, couchbase, leveldb

Databases

Shared reading and writing

Writes are hard, reads are easy

- Only if doing semi-complex queries in a predictable way

Access is almost as trivial as POSIX

- ODBC

Originally designed for KV-style accesses for known datatypes, but BLOBs (Binary Large Objects) are also available

- But these can be object/file URIs instead

Parallel solutions suitable for HPC are rare

Useful KV Patterns

Use as a burst tier

- Lustre hates n-to-n and n-to-1. Use middle layer to reshape to n-to-m, and do it out-of-band

Use as intermediate format

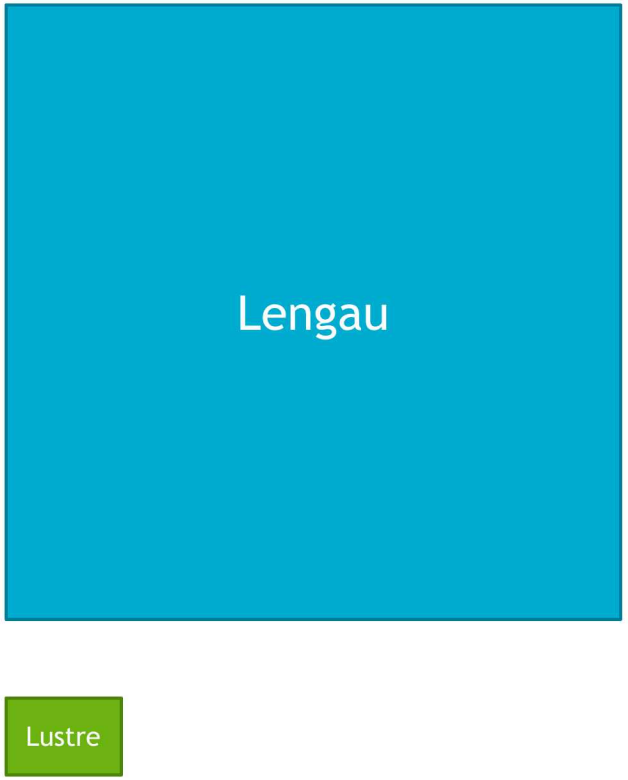
- Reform KV contents as HDF, bulk-load DB, etc.

Use as a query accelerator

- Use memcached as intended
- Use hashes (with chaining) if query IDs are too long

Use as scratch space

- Expected interactions between nodes that have unclear participants or timetables



Useful KV Patterns

Use as a burst buffer

- Lustre hates n-to-n and n-to-1. Use middle layer to reshape to n-to-m, and do it out-of-band
- Note that some systems require predictable keys

Use as intermediate format

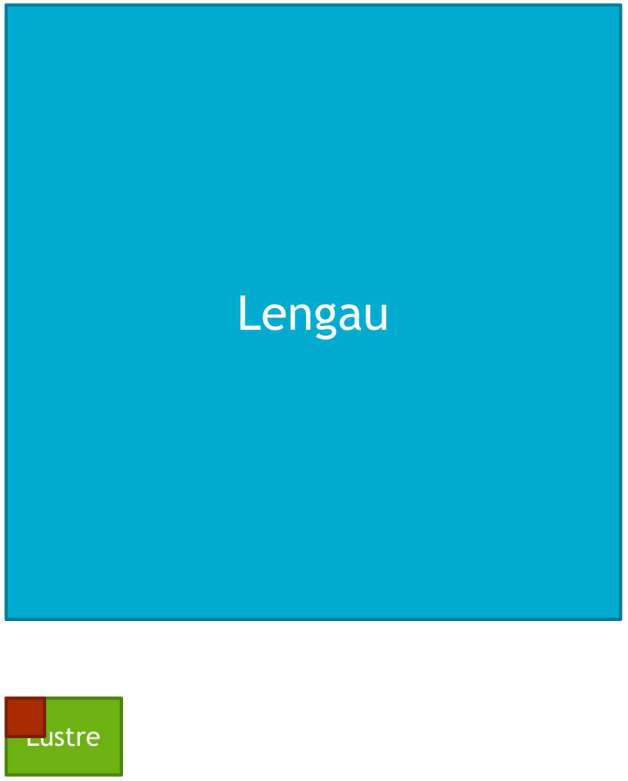
- Manual export from KV as HDF, bulk-load DB, etc.

Use as a query accelerator

- Use memcached as intended
- Use hashes (with chaining) if query IDs are too long

Use as scratch space

- Expected interactions between nodes that have unclear participants or timetables



Useful KV Patterns

Use as a burst buffer

- Lustre hates n-to-n and n-to-1. Use middle layer to reshape to n-to-m, and do it out-of-band
- Note that some systems require predictable keys

Use as intermediate format

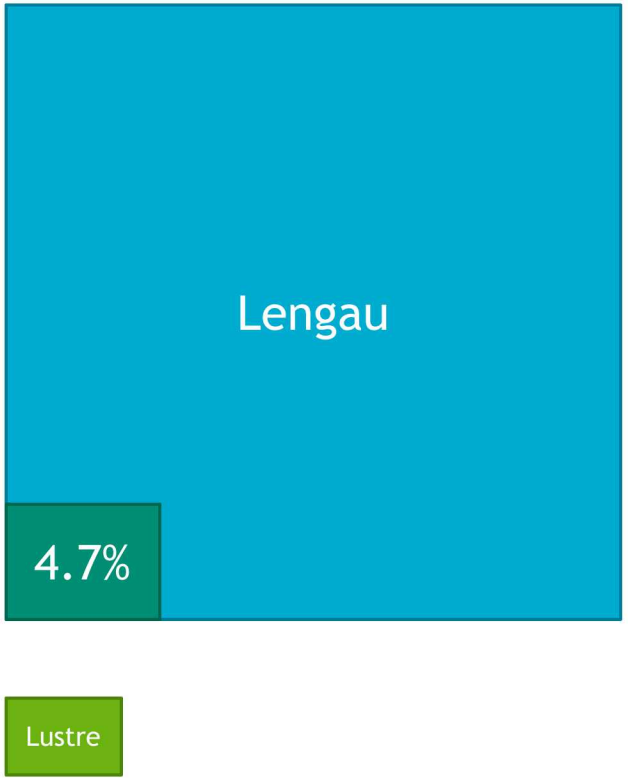
- Manual export from KV as HDF, bulk-load DB, etc.

Use as a query accelerator

- Use memcached as intended
- Use hashes (with chaining) if query IDs are too long

Use as scratch space

- Expected interactions between nodes that have unclear participants or timetables



Conclusion

A wide variety of I/O paradigms are available

Each has its own typical set of limitations

- Some are not inherent to the paradigm, but are conventional

Batteries not included

- Many packages aren't designed for our environment

Currently no easy way to ensure paradigm independence

- Upcoming I/O APIs are designing toward this possibility

Dependencies are challenging. Use Spack.

	Files	Objects	KV	DB
Appendable	No	Yes	Yes	No
Partial Overwrite	Yes	No	No	Maybe
Immutable	No	No	Yes	No
Transactional	No	Yes	Yes	Yes
Memory buffering	No	No	Yes	No
Value size	Large	Large	Small	Small
Elastic	No	Yes	Yes	No
Iteratable	Yes	Yes	No	Yes
Versioning	No	Yes	No	No
Embeddable	No	No	Yes	Yes

Thanks

mlcurry@sandia.gov

http://www.cs.sandia.gov/cr-mlcurry