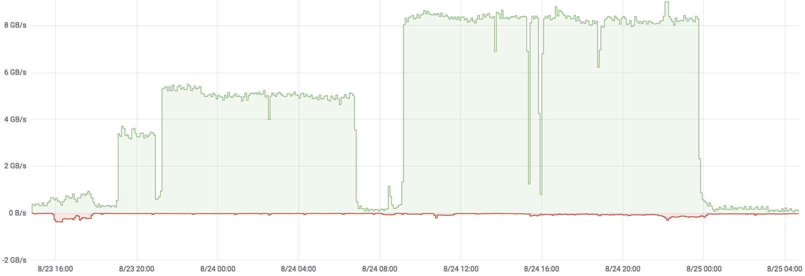
Karlsruhe Institute of Technology

Distributed File Checksumming

Practical Course: Data Management and Data Analysis at the SCC

Steinbuch Centre for Computing (SCC)



Outline

Problem Description

Karlsruhe Institute of Technology

Motivation

- The SCC operates several large file systems (total 44 PB)
- Powered by *IBM Spectrum Scale* (formerly GPFS) and RAID
- **No verification of long-term file integrity**: Silent data corruption?

Goal

- Develop a distributed system which calculates file content checksums
- System runs regularly to maintain database of checksums
- Emits corruption warnings in time to restore files from backup

Problem Description

Motivation

- The SCC operates several large file systems (total 44 PB)
- Powered by *IBM Spectrum Scale* (formerly GPFS) and RAID
- **No verification of long-term file integrity**: Silent data corruption?

Goal

- Develop a distributed system which calculates file content checksums
- System runs regularly to maintain database of checksums
- Emits corruption warnings in time to restore files from backup

Problem Description

Karlsruhe Institute of Technology

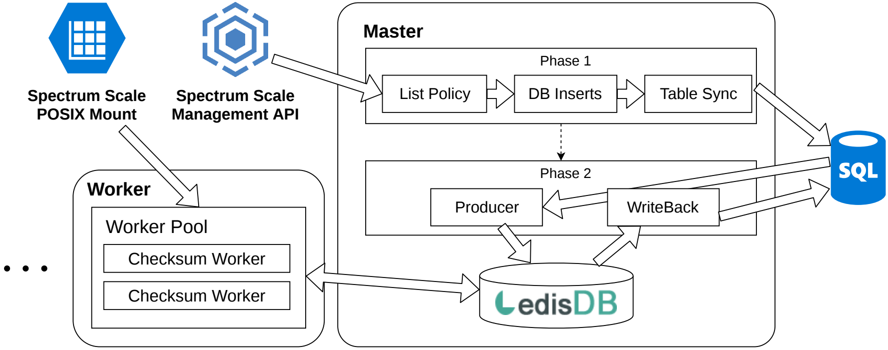
Challenges

- Resilience to node and process failures
- Ability to scale up and down
- Online file systems: Don't impair regular users' work

Design: Key Aspects

- Types of nodes: Master, Worker
- Meta Data Database (SQL): Persistent store of file meta data including checksum
- Files: Identified by path, changes detected via modification time (POSIX)
- Master $\leftrightarrow$ Worker coordination: Central work queue
- Types of runs
  - **Full**: Read all files, emit warnings on checksum mismatch
  - **Incremental**: Read only changed files

Design: Schema



Work Queue

- LedisDB with `gocraft/work`
- Jobs must be queued explicitly
- Queue length can be queried

Scheduler: Objectives

- Queue rarely exhausted (queue length == 0)
- Small queue length
- Low frequency scheduling

Work Queue

- LedisDB with `gocraft/work`
- Jobs must be queued explicitly
- Queue length can be queried

Scheduler: Objectives

- Queue rarely exhausted (queue length == 0)
- Small queue length
- Low frequency scheduling

Work Queue: EWMA Scheduler

- Idea: Enqueue matching current consumption
- Perform scheduling operation at regular interval *intv*
- Track consumption C, deviation D from expected consumption

Scheduler Phases

- Start up
  - High-frequent scheduling, $intv = 10ms$
  - Establish values for $\text{EWMA}(C)$, $\text{EWMA}(D)$
  - Min queue length: $WorkerNum \times N_{WorkerNum}$
- Maintaining
  - Scheduling at greater interval, $intv = 10s$
  - Min queue length: $\mathbb{E}(C \text{ during } intv) + N_{Deviation} \times \mathbb{E}(D \text{ during } intv)$

Work Queue: EWMA Scheduler

- Idea: Enqueue matching current consumption
- Perform scheduling operation at regular interval *intv*
- Track consumption C, deviation D from expected consumption

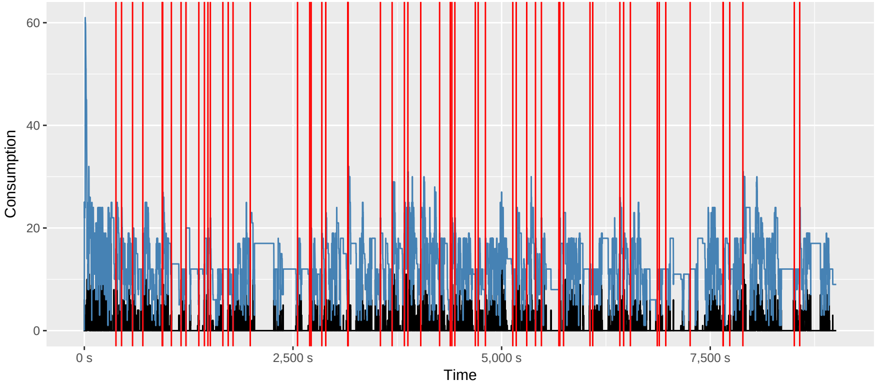
Scheduler Phases

- Start up
  - High-frequent scheduling, $intv = 10ms$
  - Establish values for EWMA (C), EWMA (D)
  - Min queue length: $WorkerNum \times N_{WorkerNum}$
- Maintaining
  - Scheduling at greater interval, $intv = 10s$
  - Min queue length: $\mathbb{E}\left(C \text{ during } intv\right) + N_{Deviation} \times \mathbb{E}\left(D \text{ during } intv\right)$

Work Queue: Simulation

Parameters

- $WorkerNum = 5$
- $SchedulingSteps = 10000$
- $intv = 1s$ (Maintaining)
- $N_{Deviation} = 5$



Work Queue: Metrics

Efficiency

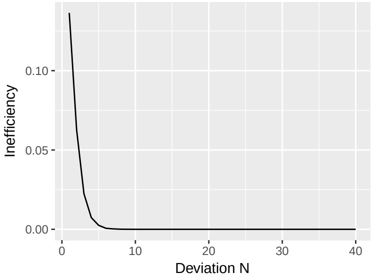
- Upper bound on time lost due to empty queue
- Queue exhausted during scheduling interval? $\rightarrow$ Regard interval as idle
- **Efficiency** $E = \frac{\text{Non-Idle Time}}{\text{Total Time}}$, **Inefficiency** $1 - E$

Work Queue: Metrics

Efficiency

- Upper bound on time lost due to empty queue
- Queue exhausted during scheduling interval? $\rightarrow$ Regard interval as idle
- **Efficiency** $E = \frac{\text{Non-Idle Time}}{\text{Total Time}}$, **Inefficiency** $1 - E$

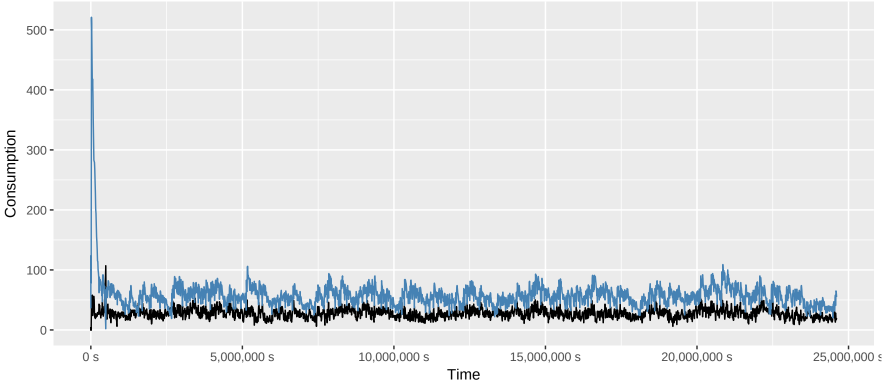
Evaluation



- $intv = 1s$ (Maintaining)
- $WorkerNum = 5$

Work Queue: Full Test Run

- File tree generated using Lognormal
- 16 Workers: 3 TiB of file data, 600 k files



I/O Performance

- Goal: Restrict impact on other file system users during checksumming
- Idea: Rate limit I/O throughput on the syscall level
- Every call to `read()` is guarded by a rate limit request

Limits

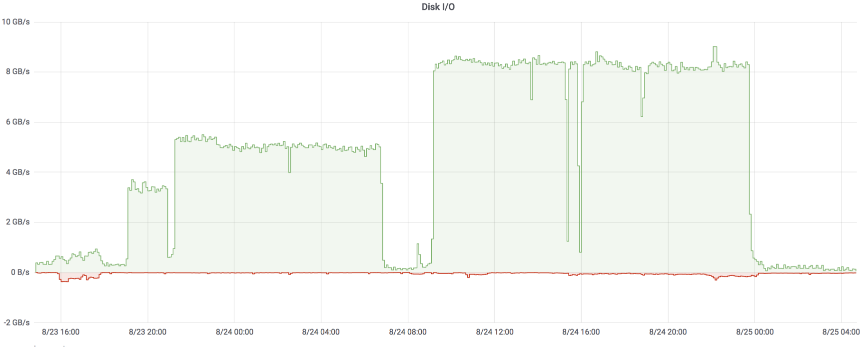
- Master: Global I/O throughput limit
- Worker: Local I/O throughput limit

Evaluation

- Migration of file system subtree
- 382 TiB of file data, 99 M files
- Deployed lsdf-checksum to verify file integrity

Evaluation: System Performance

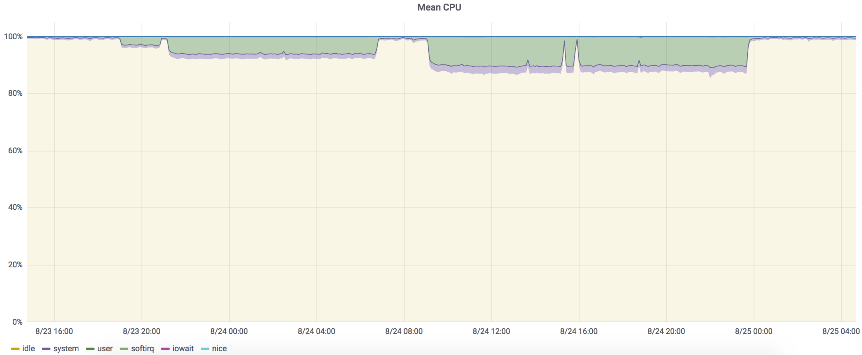
Total Disk I/O of the Worker Cluster (13 Nodes)



- Initial concurrency (per node): 2
- Initial max_node_throughput: 500 MiB/s
- Final concurrency (per node): 10
- Final max_node_throughput: 800 MiB/s

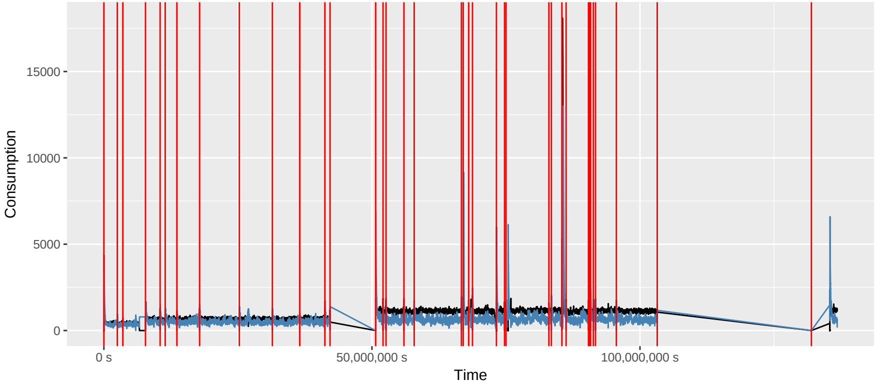
Evaluation: System Performance

Mean CPU of the Worker Cluster (13 Nodes)



- Initial concurrency (per node): 2
- Initial max_node_throughput: 500 MiB/s
- Final concurrency (per node): 10
- Final max_node_throughput: 800 MiB/s

Evaluation: Queue



Lessons Learned

Karlsruhe Institute of Technology

- Testing vs production environments and data
  - Volume: Orders of magnitude more data
  - Variety: Edge cases in real-world data
  - Reduced observability
- Complex tools introduce complex problems
  - Was SQL a good choice?

References I

David SH Rosenthal. "Keeping bits safe: how hard can it be?" In: *Communications of the ACM* 53.11 (2010), pp. 47–55.

- The Spectrum Scale logo is Copyright International Business Machines Corporation. *IBM Spectrum Scale* is a trademark of the International Business Machines Corporation.
- The Go Logo is is Copyright The Go Authors.
- The LedisDB Logo is Copyright siddontang.
- Plots have been created using R.
- Further graphics have been created using `https://www.draw.io/`

Design: Technology

Karlsruhe Institute of Technology

Go

- Explicit data structures (low-level?)
- Lightweight concurrency
- Compiles statically-typed native binaries

SHA-1

- 160 bit (20 byte) hash sums
- Considered not-cryptographically secure
- Performance (`gpfstest-03`, Intel Xeon E5 2640 v2)

437 MiB/s	`sha1sum`
301 MiB/s	Go implementation comparable to Worker

Design: Technology

Karlsruhe Institute of Technology

Go

- Explicit data structures (low-level?)
- Lightweight concurrency
- Compiles statically-typed native binaries

SHA-1

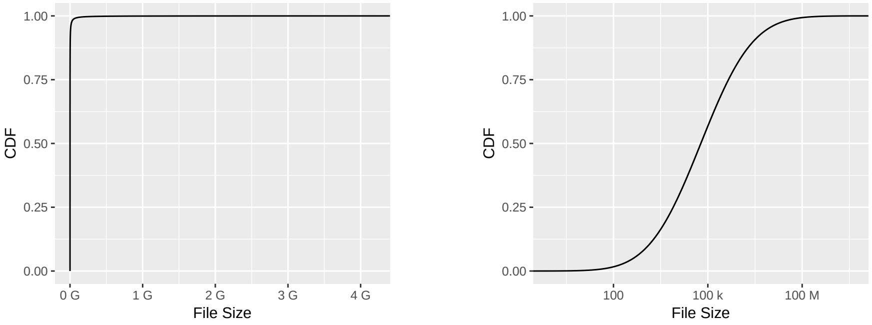
- 160 bit (20 byte) hash sums
- Considered not-cryptographically secure
- Performance (`gpfstest-03`, Intel Xeon E5 2640 v2)

  437 MiB/s `sha1sum`
  301 MiB/s Go implementation comparable to Worker

Work Queue: Simulation

Assumption: File Size Distribution

- Lognormal Distribution: Lognormal (μ, σ^2)
- Parameters: $\sigma = 11, \mu = 3$



Work Queue: In Practice

Work Packs

- Goal: Reduce network and de-queuing overhead
- Each job: Work Pack containing multiple files
- Total file size has to exceed threshold, e.g. 5 MiB

Randomisation

- Goal: Uniform distribution of file sizes over time
- Explicitly order files randomly (SQL: `RAND()`)
- Add files to Work Pack in this order

Work Queue: In Practice

Work Packs

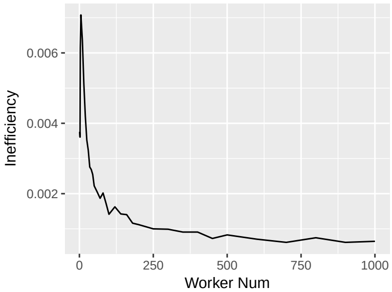
- Goal: Reduce network and de-queuing overhead
- Each job: Work Pack containing multiple files
- Total file size has to exceed threshold, e.g. 5 MiB

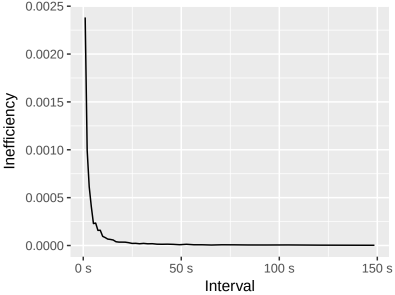
Randomisation

- Goal: Uniform distribution of file sizes over time
- Explicitly order files randomly (SQL: `RAND()`)
- Add files to Work Pack in this order

Work Queue: Metrics

Evaluation





- $intv = 1s$ (Maintaining)
- $N_{Deviation} = 5$

- $WorkerNum = 5$
- $N_{Deviation} = 5$

I/O Performance: Token Bucket

- Bucket containing a number of tokens
- Tokens are replenished at constant rate
- Upper bound on number of tokens → burstiness