Los Alamos NATIONAL LABORATORY

GEORGETOWN UNIVERSITY

Offloading Calculations to Computational Storage Devices: Spark and HDFS

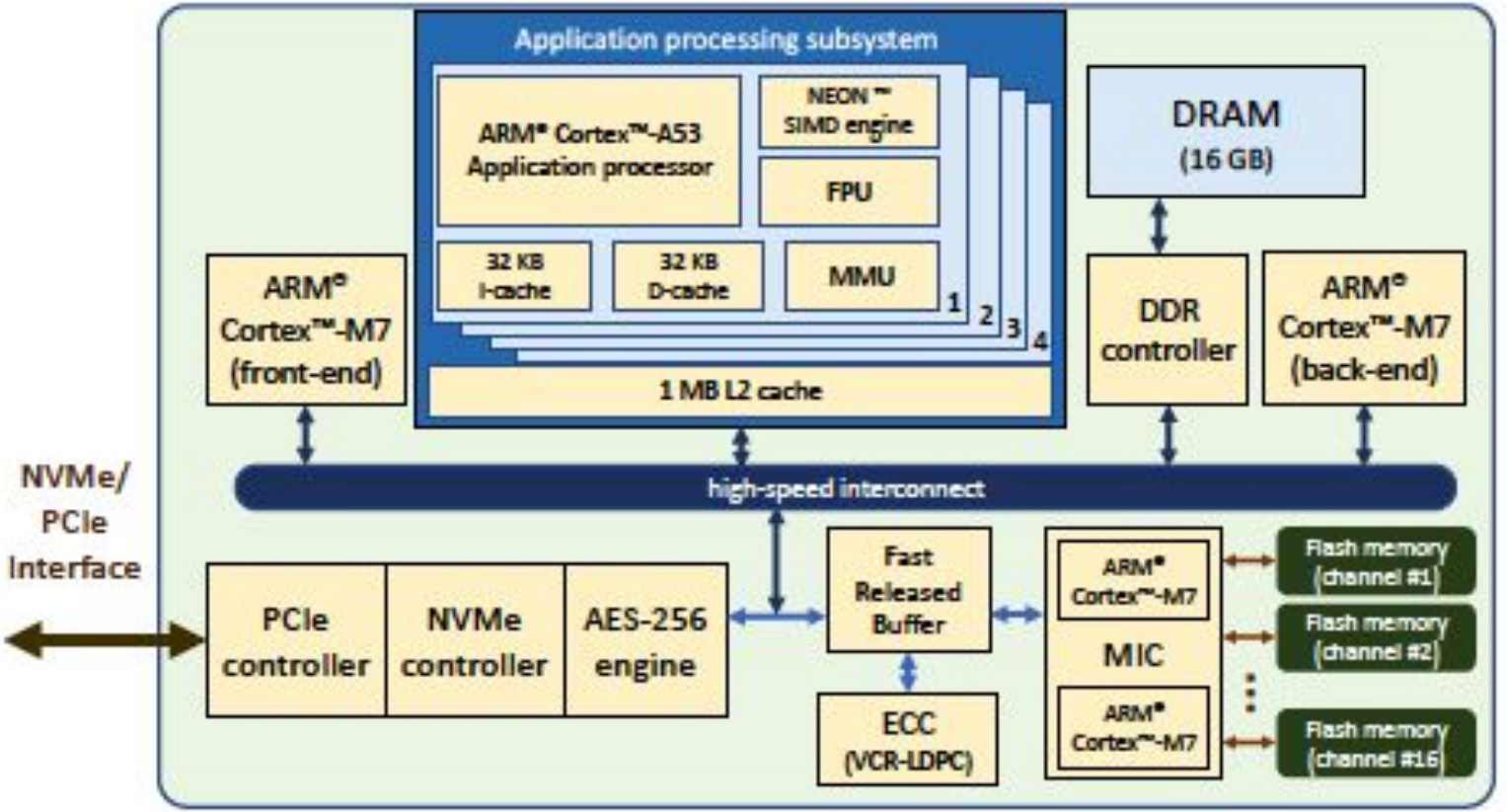
Clyburn Cunningham IV, Justin Goldstein, Warren Hammock (USG), Jacob Janz, Ralph Liu, Mitch Rimerman

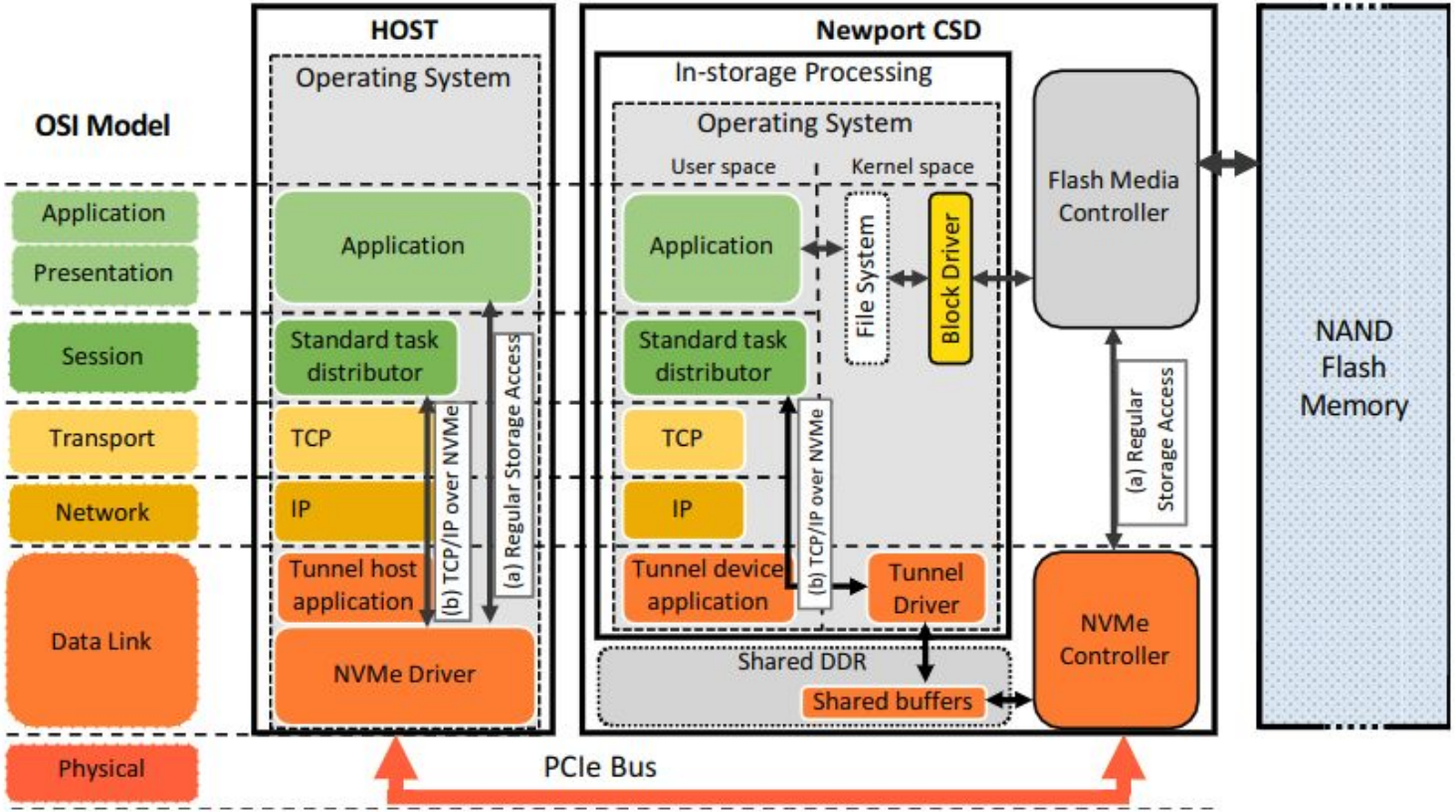
Mentors: Shane Goff, Steve Poole, Kevin Bryant (USG)

LA-UR-21-28021

Introducing Computational Storage Devices (CSDs)

- Computational Storage → Near-data processing
- CSD → Runs software where data resides
- Potential performance improvement


Application processing subsystem
ARM® Cortex™-A53 Application processor
NEON™ SIMD engine
FPU
32 KB I-cache
32 KB D-cache
MMU
1
2
3
4
1 MB L2 cache
DRAM (16 GB)
DDR controller
ARM® Cortex™-M7 (back-end)
ARM® Cortex™-M7 (front-end)
high-speed interconnect
NVMe/ PCIe Interface
PCIe controller
NVMe controller
AES-256 engine
Fast Released Buffer
ECC (VCR-LDPC)
ARM® Cortex™-M7
MIC
ARM® Cortex™-M7
Flash memory (channel #1)
Flash memory (channel #2)
Flash memory (channel #16)


OSI Model
Application
Presentation
Session
Transport
Network
Data Link
Physical
HOST
Operating System
Application
Standard task distributor
TCP
IP
Tunnel host application
NVMe Driver
(a) Regular Storage Access
(b) TCP/IP over NVMe
Newport CSD
In-storage Processing
Operating System
User space
Kernel space
Application
Standard task distributor
TCP
IP
Tunnel device application
File System
Block Driver
Tunnel Driver
Shared DDR
Shared buffers
Flash Media Controller
NVMe Controller
(a) Regular Storage Access
(b) TCP/IP over NVMe
NAND Flash Memory
PCIe Bus

Introducing Hadoop and Spark

- Apache Hadoop → Used to store and process large datasets
  - Hadoop Distributed File System (HDFS)
  - Ecosystem includes many useful tools/applications
- Apache Spark → Distributed processing system used for big data
  - Enhances processing performance

Experimental Objective and Design

- Objective → Evaluate the capabilities of multiple CSDs (provided by NDG Systems) using Hadoop Filesystem and Apache Spark
  - Use native Spark libraries, such as SparkSQL and DataFrames, to perform matrix operations on datasets
- Independent Variables
  - # of CSDs → 0, 1, 2, 4, or 6
  - Size of dataset → 1 GB, 5 GB, 10 GB
  - Type of dataset → One large file with all of the data, 10 files, 100 files
- Dependent Variables
  - Job time
  - Execution time
- Constants
  - Operations on dataset

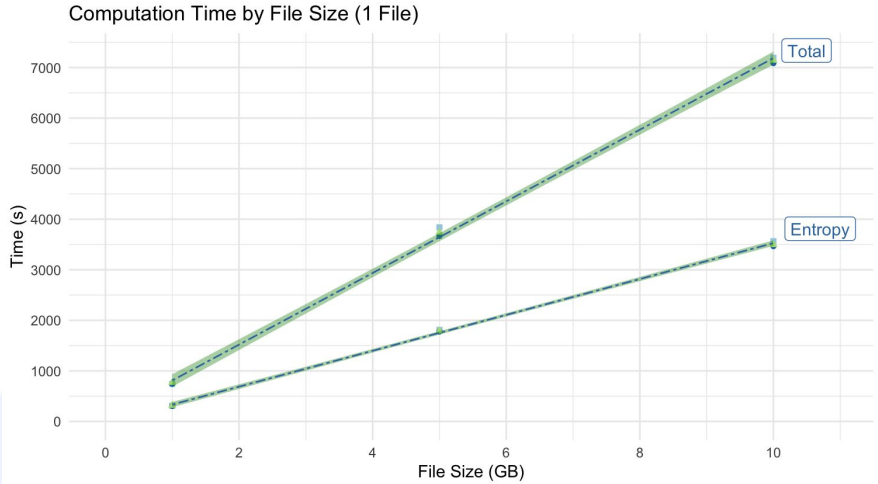
Experimental Design Continued

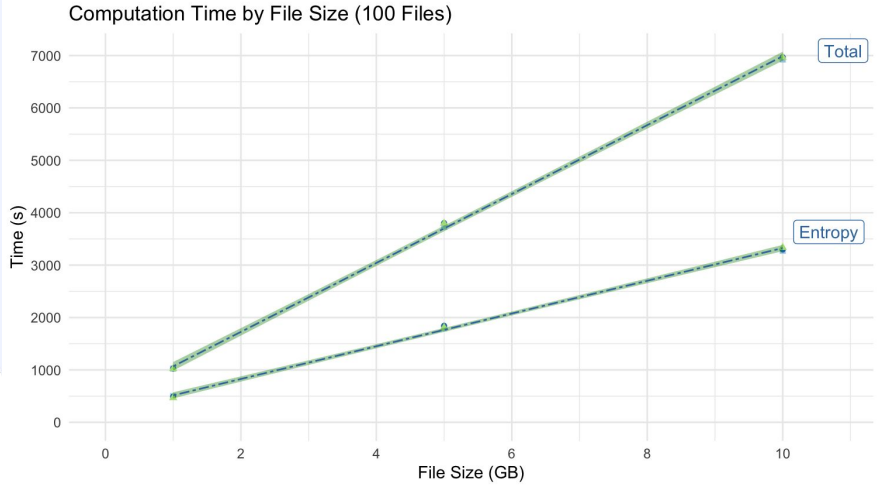
- Ran the experiment 3 times
- Used Trinity sensor data
- Operations
  - Count lines
  - Column operations
    - Sum and average
    - Multiplication and modular arithmetic
    - Mean and standard deviation
    - Compute gram matrix and determinant
  - Measure entropy

Experiment Results

File Size

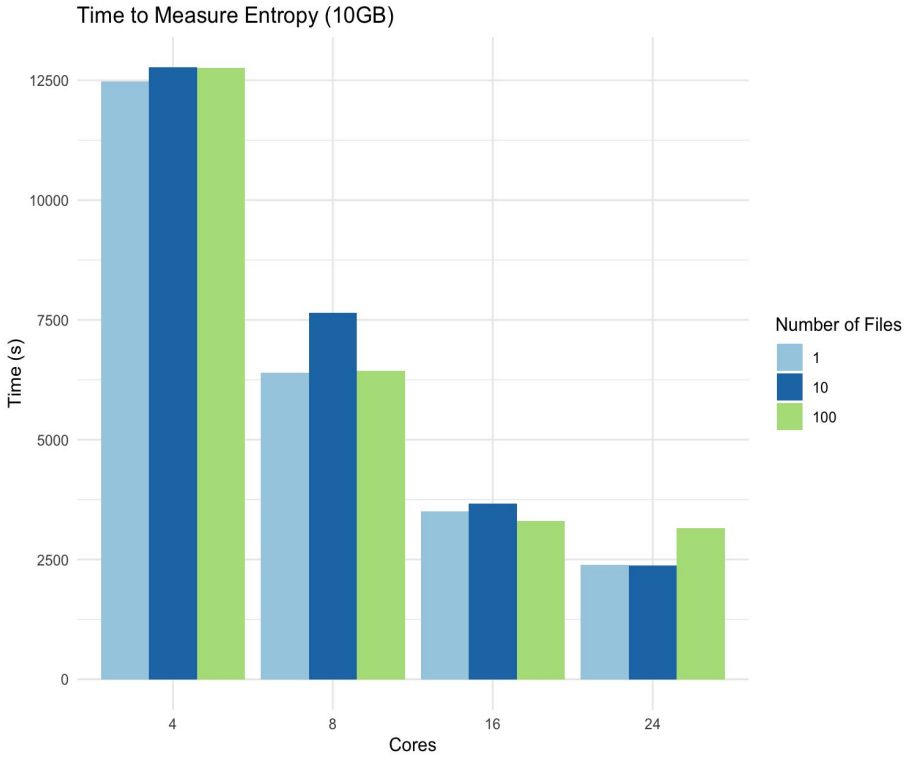
- Linear Scaling with increased file size holding number of CSDs constant

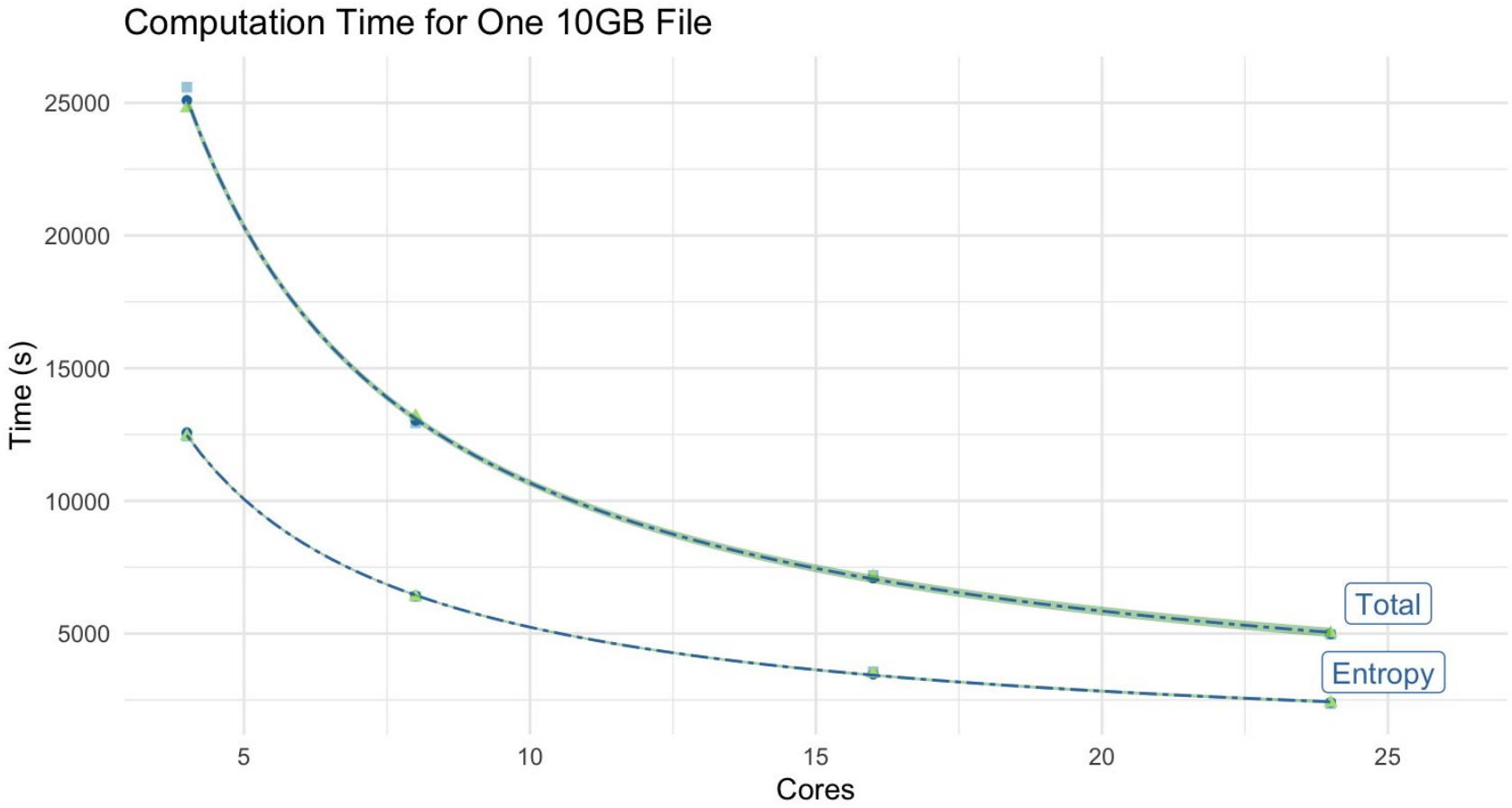


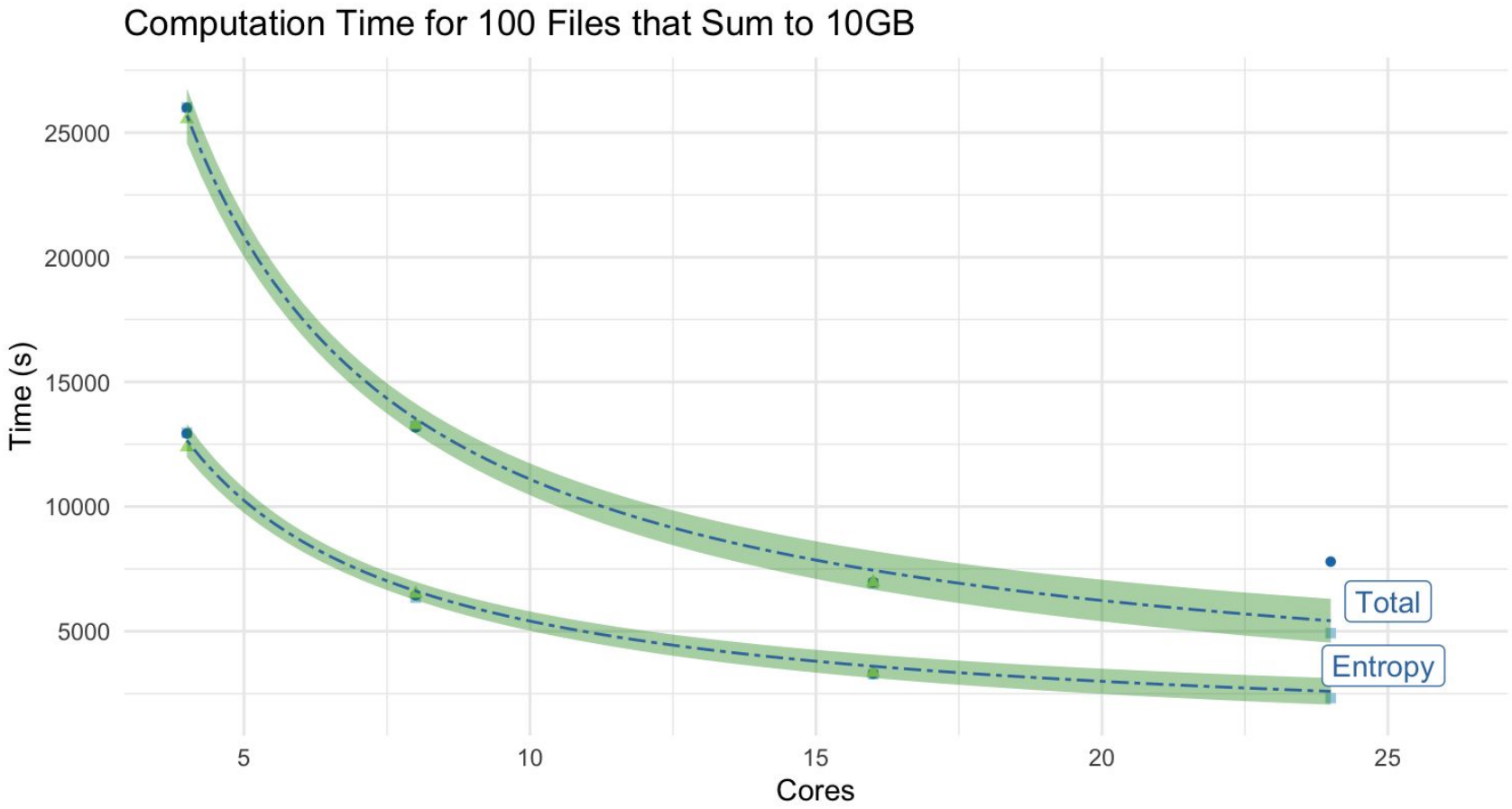


Number of Files

- Increased performance with more CSD’s
- Similar observations for different file amounts
- Lesser improvement for more nodes with large amount of files

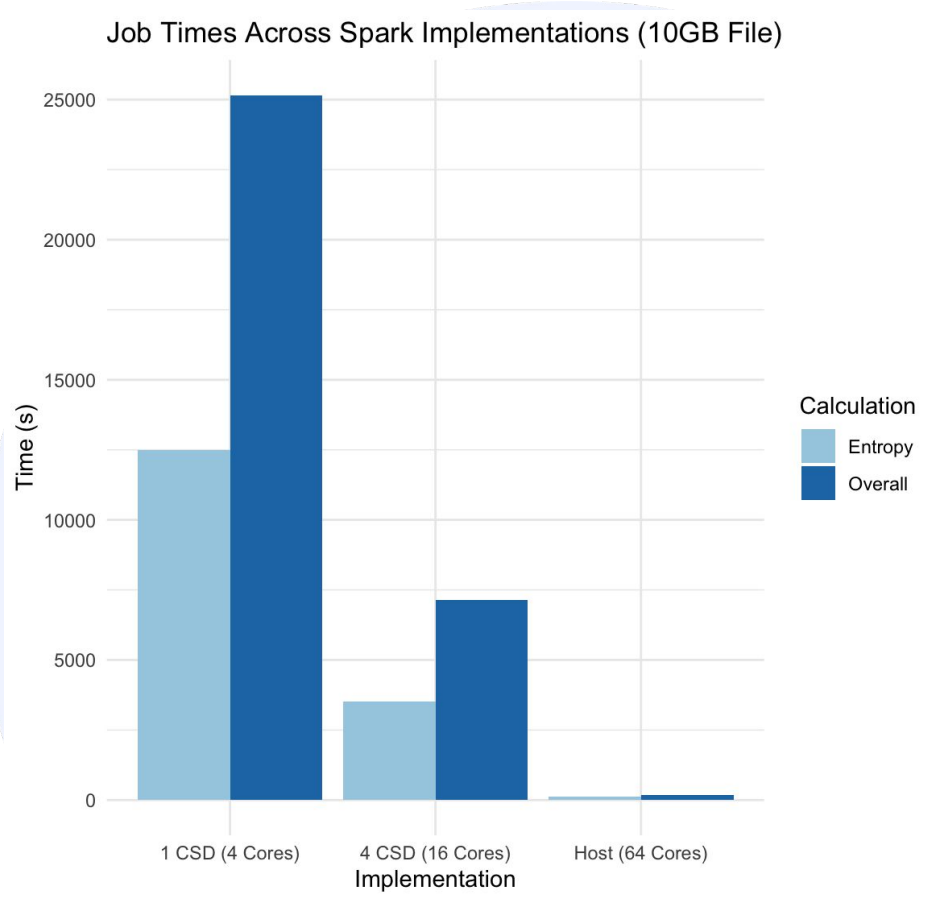



Computation Time for One 10GB File
25000
20000
15000
10000
5000
Time (s)
5
10
15
20
25
Cores
Total
Entropy


Computation Time for 100 Files that Sum to 10GB
Time (s)
25000
20000
15000
10000
5000
5
10
15
20
25
Cores
Total
Entropy

Comparison to Host

- Much faster on Host
- Assuming uniform scaling, achieving host performance would not be possible with any amount of CSDs

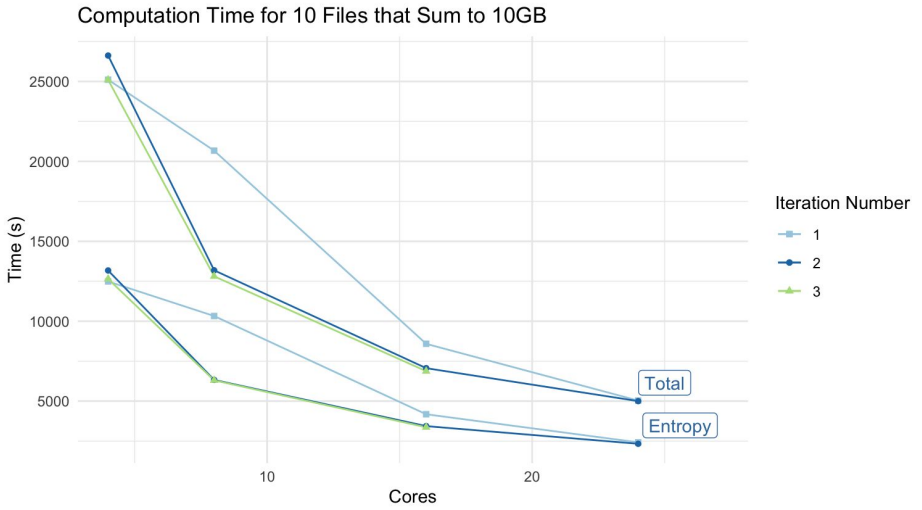


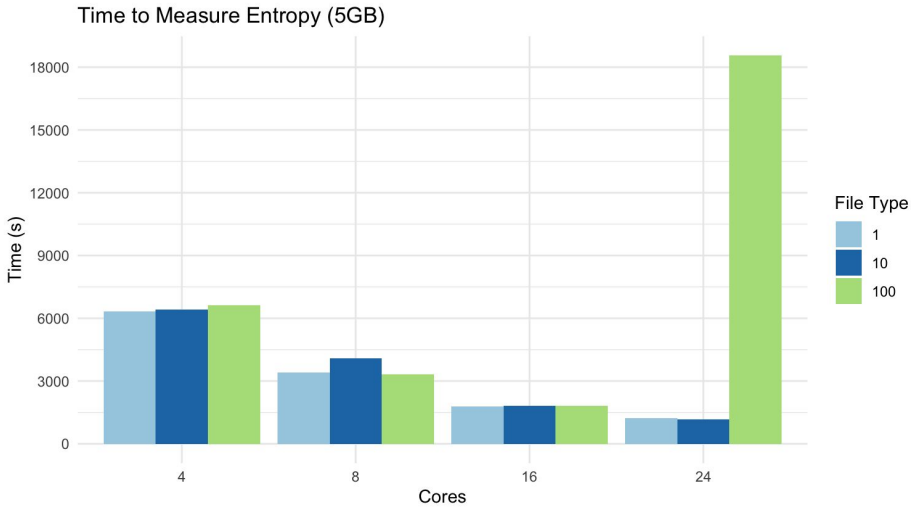
Conclusion

We Observed That

- Ineffective at offloading our operations
- Time v. Size scales linearly
- Time v. Cores scales inversely
- File size v. quantity matters

Important Observations





Future Work

Scalable, but CSDs are not fast and are unstable

- Drives break often
- If one breaks, all must halt
  - Erase and reinstall Linux

Moving past Spark?

- Removing overhead