Applying selectively parallel I/O compression to parallel storage systems

Rosa Filgueira: University of Edinburgh, Edinburgh, U.K.

Malcolm Atkinson: University of Edinburgh, Edinburgh, U.K

Yusuke Tanimura: AIST, Tsukuba, Japan

Isao Kojima: AIST, Tsukuba, Japan

edinburgh data-intensive research

Outline

- Motivation
- Problem
- Objective
- Proposal
- Papio
- Adding compression to Papio
- Adding decompression to Papio
- Evaluations
- Conclusions
- Future Work

Non-technical

Technical

Motivation

- **Large scale Data-Intensive Computing** plays an important role in many scientific activities and commercial applications:
  - data mining of commercial transactions
  - experimental data analysis and visualization
  - intensive simulation such as climate modelling
- The **challenge** is to develop a **new framework** to support Data- Intensive Computing:
  - **persistent storage for large datasets**
  - balanced computing

Problem

- PFS used in Data-Intensive Computing for high-performance I/O.
- PFS can scale to support a large number of clients and data.
- Inconvenient:
  - Imbalance between I/O throughput and compute power
  - Expensive storage network
  - Limitation of hard disk drive (HDD) throughput
- Problem:
  - The rate at which data can be delivered from disk to compute engine is a limiting factor:
    - **data-transfer channel becomes a bottleneck**

Objective

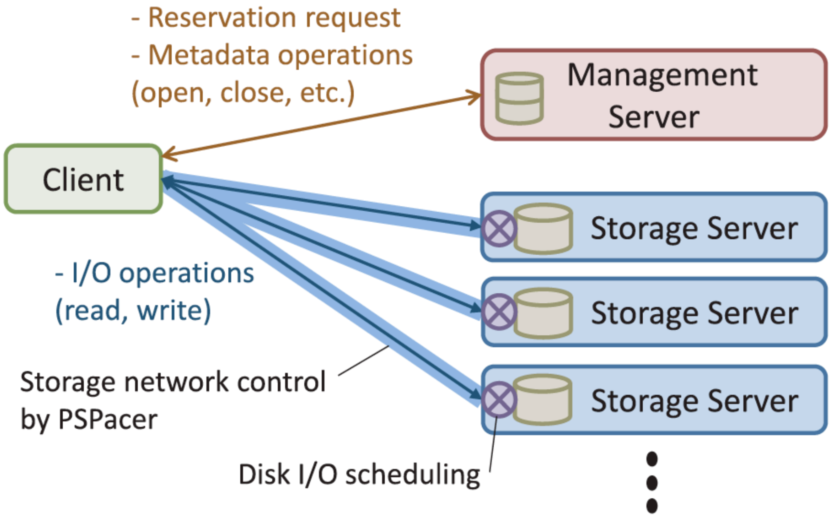
- To reduce the **data-transfer bottleneck:**
  - decreasing the overall I/O time for transferring datasets between the computer and storage system (and vice versa)

Proposal

- New I/O strategies for providing high-speed storage and access to a parallel storage system:
  - *Sequential Compression*
  - *Parallel Compression*
  - *Selectively Parallel I/O Compression* (SPIOC)
- They apply transparent run-time lossless compression between the computing and the storage systems

Proposal applied in Papio

- Papio is a parallel storage system.
- It was designed for large scale cluster computing.
- It provides QoS guarantees by employing a reservation approach.
- **Papio + New I/O Strategies** = To reduce the total I/O time while satisfying the reservation requests from users.



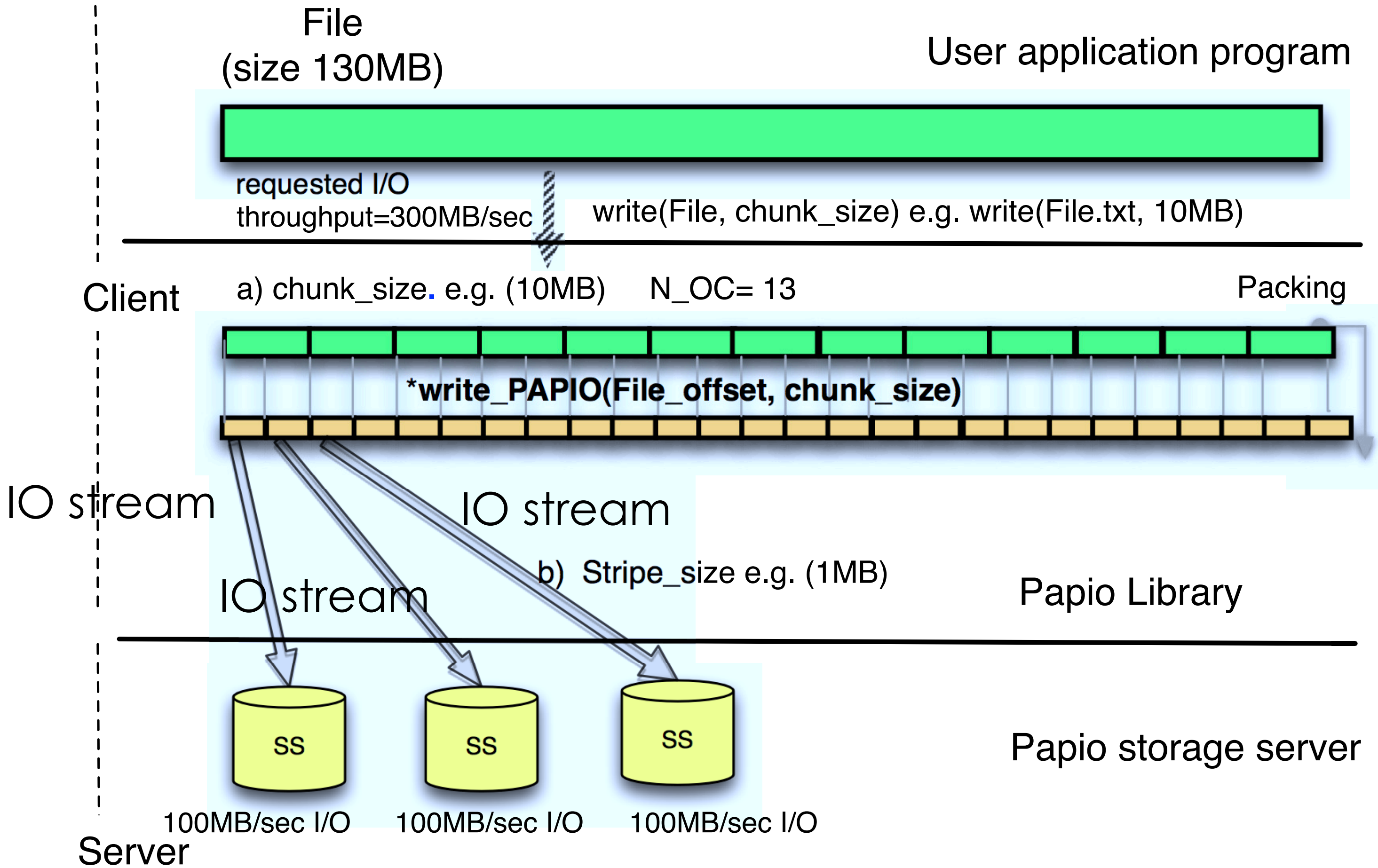
Papio – basic concepts

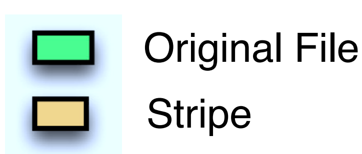
- It allows users to reserve I/O performance with desired:
  - throughput (e.g. MB/sec)
  - access type (read or write)
  - access time (from start to end)
- When requested throughput > than that provided by a single Storage Server (SS):
  - Multiple SSs are used → increasing the I/O parallelization

Papio – terminology

- ***I/O stream*** → the channel which the data flow between the client's application and one SS.
- ***Stripe count*** → Total number of I/O streams = *Number of SSs.*
- ***Chunk*** → Users can cut up a file into parts (*chunks*).
- In each I/O operation, the ***chunk*** is striped over ***all of the I/O. streams.***
- ***Stripe*** → The minimum unit of transfer via an ***I/O stream.***
- ***Stream width*** → The total amount of data written or read each time by the ***I/O streams*** → ***stripe size* stripe count***

Writing a file to Papio





Number of *stripes* = 130
Stripe count = Number of *SSs* = 3 .
Stream width = *stripe size* (1MB) * *stripe count*= 3MB

* Number of write_PAPIO operations = 13

Adding compression to Papio

- Selecting the compression algorithm
- Compression Strategies:
  - *Sequential Compression Strategy*
  - *Parallel Compression Strategy*
  - *Selectively Parallel IO Compression Strategy*

Selecting the compression algorithm

- **Hypothesis**: depending on the data-type and redundancy, some algorithms:
  - can compress better than others
  - may need more time to compress/decompress
- Compression algorithms selected: RLE , HUFFMAN, LZO, **Snappy, LZ4**.
- Studies with synthetic and real files: different data-types, data-sizes, and redundancy levels.
  - **LZ4** is the fastest algorithm in all cases.
  - The highest compression ratios are not always achieved by the faster algorithms.
- Compression criteria for our strategies:
  - High-speed storage and access to Papio.
  - High-speed algorithms are preferred over high-compression algorithms
  - **LZ4** is selected as the compression algorithm to use.

Selecting the compression algorithm- LZ4

File	Size(MB)	Compression ratio					Time compr. + decompr(sec)				
		RLE	HUFF	LZO	Snappy	LZ4	RLE	HUFF	LZO	Snappy	LZ4
3D_spatial	20	1.00	2.14	1.67	1.61	1.65	10.4	54.00	8.45	6.33	5.36
pop_failure	0.25	1.17	2.3	1.63	1.65	1.52	12.89	74.69	15.07	11.74	10.00
regression_tom	15	1.24	3.51	4.95	4.79	4.70	10.61	35.33	3.70	2.80	2.46
regression_twitter	217	1.20	2.99	4.08	4.27	3.81	7.06	36.04	4.18	3.24	3.26
ad	9.8	1	5.2	33.05	16.6	38.65	9.6	24.04	0.98	0.83	0.60
E.coli	4.5	1.01	4	2.03	2.14	1.60	14.72	46.01	9.91	5.97	4.69
Bible.txt	3.9	1.00	1.82	2.02	2.03	1.93	20.01	92.49	7.92	5.33	4.82
World192.txt	2.4	1.02	1.58	1.98	1.99	2.00	10.81	123.02	7.76	5.34	5.05
plrabn12.txt	0.47	1.00	1.74	1.55	1.51	1.49	10.09	125.13	16.43	12.0[illegible]	10.79
pi.txt	0.97	1.00	2.35	1.22	1.20	1.26	11.53	85.07	14.00	14.0[illegible]	11.89
kennedy.xls	1	1.00	2.22	2.84	2.42	2.74	11.02	62.21	6.29	5.51	4.88
enwki8	35	1.00	1.56	1.79	1.76	1.97	6.44	75.07	8.25	6.33	6.10
enwiki9	954	1.00	1.54	1.99	1.97	1.75	6.08	76.25	7.55	5.61	5.16
Mesh3	14	1.00	2.40	7.81	7.61	6.50	8.18	49.01	4.98	3.74	3.29
Mesh4	26	1.00	2.26	2.48	2.55	2.05	8.18	42.81	5.42	5.64	3.59

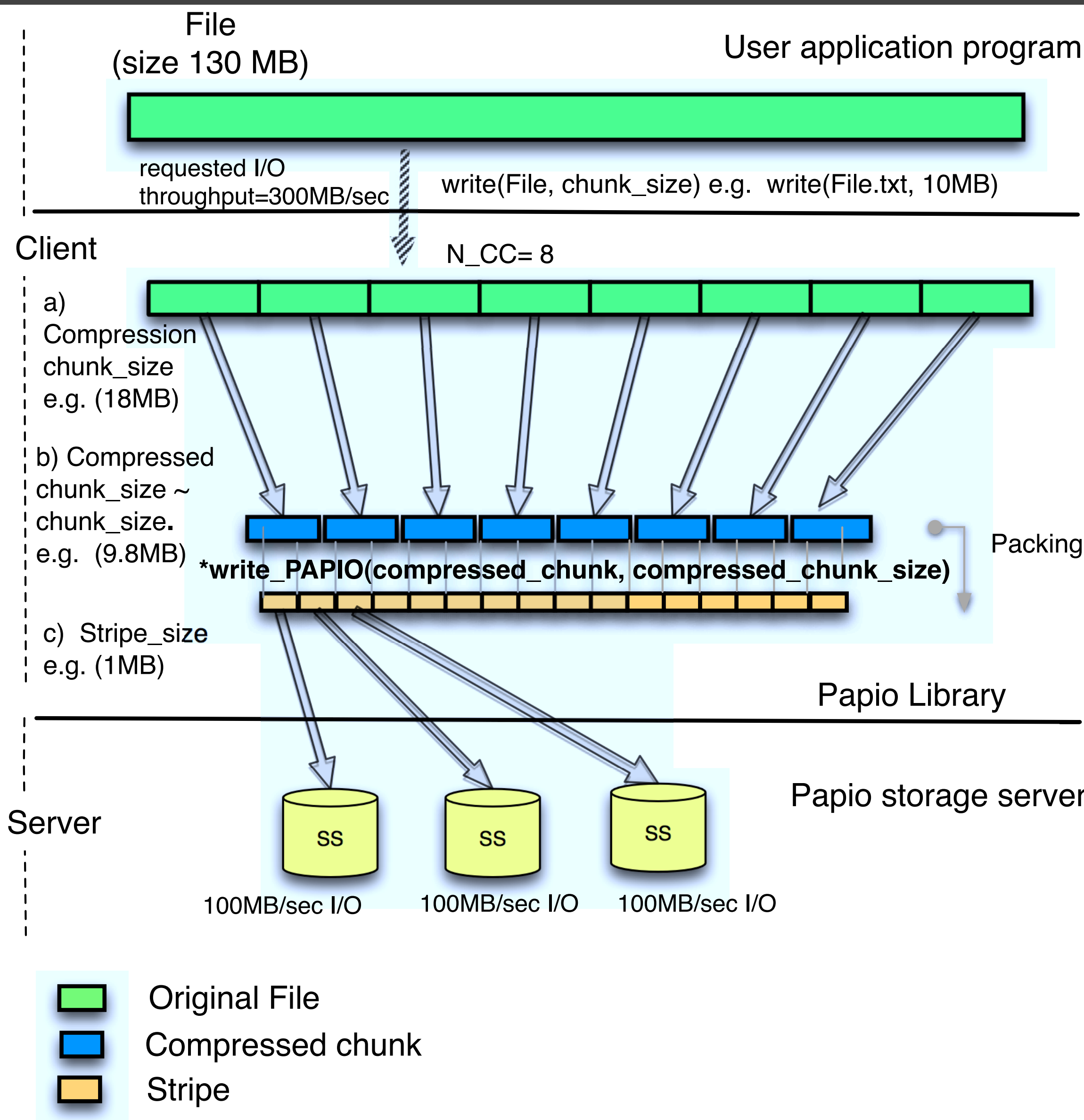
Compression ratio, and compression and decompression times for real files

LZ4 focused on compression and decompression speed.
It belongs to the LZ77 family of byte-oriented compression schemes.

Sequential Compression Strategy

- It automatically divides the file into several ***chunks*** and **compresses** them.
- ***Compressed chunk size*** is ~ as the ***chunk size*** specified by the user:
  - To prevent that the ***compressed chunks*** *are not* < than ***stream width***
- **Algorithm-1 :** returns the compression parameters
  - The ***number of chunks to be compressed*** (*N_CC*)
  - The ***size of the chunks to compress*** (*compression_chunk size*)
  - The ***compression ratio*** (*compression_ratio*):
    - Three slices located randomly from the middle until the end
    - The size of each slice is *5%* of the *chunk size*
    - ***Compression ratio*** = average of the compressed size of those slices.

Sequential Compression Strategy



* Number of write_PAPIO operations = 8

knowing the ***compression ratio*** (**1.8**), this algorithm computes:

- ***compression chunk size*** = ⌈*chunk size*× compression ratio⌉) → **18MB**
- ***the number of chunks to compress*** (N_CC)= ⌈*file size*/ *compression chunk size*⌉) → **8**

- The number of *chunks* reduced from (N_OC) 13 to 8 (N_CC)
- The number of *stripes* reduced from 130 to 80.

Sequential Compression Strategy

- The strategy needs extra memory for allocating the ***compressed chunks***.
- The write API of Papio has been modified to implement the **Algorithm-1** and to write the **compressed *chunks***.
- The strategy stores a **mapping file** with information for decompressing the file.
- To achieve an improvement in the I/O write. **Equation-1**:

Time to write the original file

Time to write the compressed file

$$\frac{Time\_write * N\_OC}{(Time\_write * N\_CC) + (Time\_comp * N\_CC)} > 1$$

- ***Time_write*** is the time needed for writing a *chunk*. ***Time_comp*** is the time for compressing a *chunk*

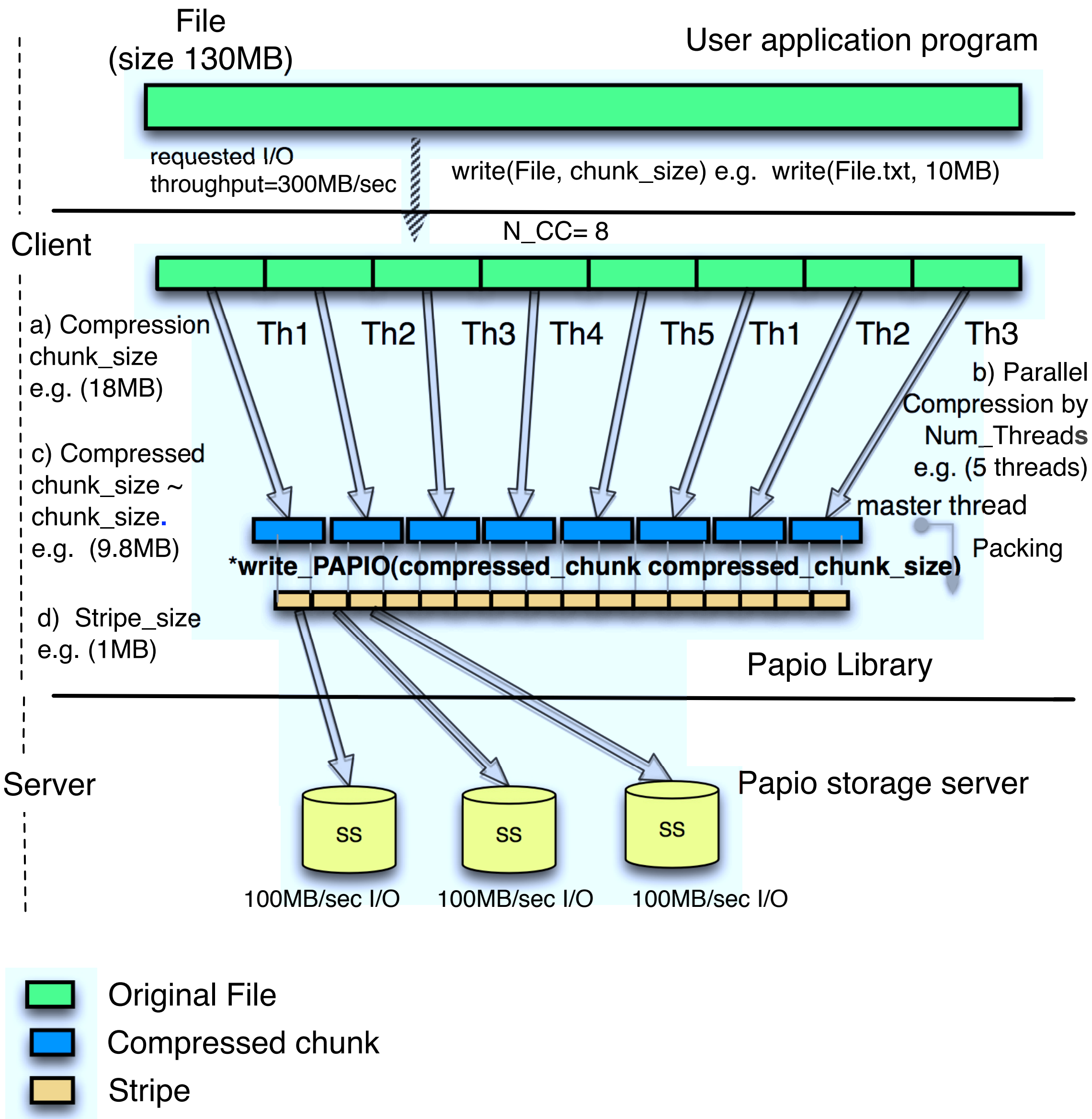
Parallel Compression Strategy

- This strategy compresses several *chunks* in parallel by using a **multithread** technique.
- Number of threads = number of cores available.
- **Algorithm-1**: To get the compression parameters:
  - ***Compression ratio, compression chunk size, number of chunks to compress***
- **Algorithm-2**: To perform the **multithread compression** and write operations.
- Papio's write requires that the file's *chunks* must be written in order:
  - Compression can be performed in parallel by several threads
  - The master thread writes *compressed chunks* following a sequential order

Parallel Compression Strategy

- **Algorithm-2**:
  1. Allocates the memory needed for compressing the *chunks*.
  2. Creates as many threads as number of cores (nc).
     1. Except: *number of chunks to compress* (N_CC) < nc.
  3. To each thread a *chunk* is assigned to compress.
  4. Each thread writes the compressed data in its buffer.
  5. The master thread waits until the first thread has finished the compression of its *chunk*.
  6. Then the master writes the *compressed chunks* to Papio.
  7. **If more chunks to compress + write → go to step 3.**

Parallel Compression Strategy



* Number of write_PAPIO operations = 8

- *Chunks* are compressed in groups of the number of threads.
- As soon as the first *compressed chunk* of each group is written to Papio, a new group of *chunks* are assigned to threads to be compressed.
- The master thread only waits for the first *compressed chunk*.

- 8 N_CC
- First 5 *chunks compressed* in parallel.
- Last 3 *chunks compressed* in parallel.
- Reduced= number of *stripes* + compression time

Parallel Compression Strategy

- The strategy stores also a **mapping file** with information for decompressing the file.
- To achieve an improvement in the I/O write operations. **Equation-2**:

Time to write the original file

Time to write the compressed file

$$\frac{Time\_write * N\_OC}{(Time\_write * N\_CC) + (Time\_comp) + (Time\_total\_wait)} > 1$$

$$Time\_total\_wait = Time\_wait * (N\_CC/Num\_Threads)$$

- ***Time_wait*** could be zero or near zero:
  - When the time to compress in parallel a group of *chunks* (***Time comp***) is < than the time to write those *compressed chunks.*
  - Otherwise, ***Time_wait*** would be the difference between these two times.

Selectively Parallel IO Compression Strategy (SPIOC)

- Storage system compression can save disk space
- But, compressing data can adversely affect:
  - The data is not good for compressing it.
  - Compression algorithm is not fast enough.

- **Algorithm-3 :** Predicts whether to compress or not at runtime, and allows parallel or sequential compression techniques:
  - Single core machine → SPIOC applies **sequential compression**
  - Otherwise, **multithread parallel compression**.

Selectively Parallel IO Compression Strategy (SPIOC)

- **Algorithm_3:**
  - It checks the compression ratio > than a predefined *threshold (1)*.
  - Estimation of the time for writing the file with and without compression, applying the **equation-1** in case of single core, an the **equation-2** in case of multi core:
    - Modification of the **algorithm**-1 to measure the time for compressing the slices used for checking the compression ratio (***Time_comp***).
  - Estimated reduction > than a predefined *threshold (2)*:
    - *Sequential Compresion* or *Parallel Compression* strategy is applied

Equation-1

$$\frac{Time\_write * N\_OC}{(Time\_write * N\_CC) + (Time\_comp * N\_CC)} > 1$$

Equation-2

$$\frac{Time\_write * N\_OC}{(Time\_write * N\_CC) + (Time\_comp) + (Time\_total\_wait)} > 1$$

$$Time\_total\_wait = Time\_wait * (N\_CC / Num\_Threads)$$

Selectively Parallel IO Compression Strategy (SPIOC)

- **Algorithm-3** has been implemented by modifying the write API of Papio.
- **SPIOC** has been chosen as the strategy to use in Papio.
- **Mapping file** stores if the file is stored compressed or not.

Adding decompression to Papio

- The decompression algorithm performs full and partial reads.
- It decompresses the minimum part of the file.
- Papio allows read operations to be performed without following a specific order:
  - Multiple threads can read and decompress the *chunks* in parallel.
- **Decompression algorithm**:
  - It reads the mapping file.
  - File stored with compression: it obtains the *compressed chunks* that need to be read and decompresses them in parallel.
  - File stored without compression: original version of Papio.

Evaluations

- High Performance cluster

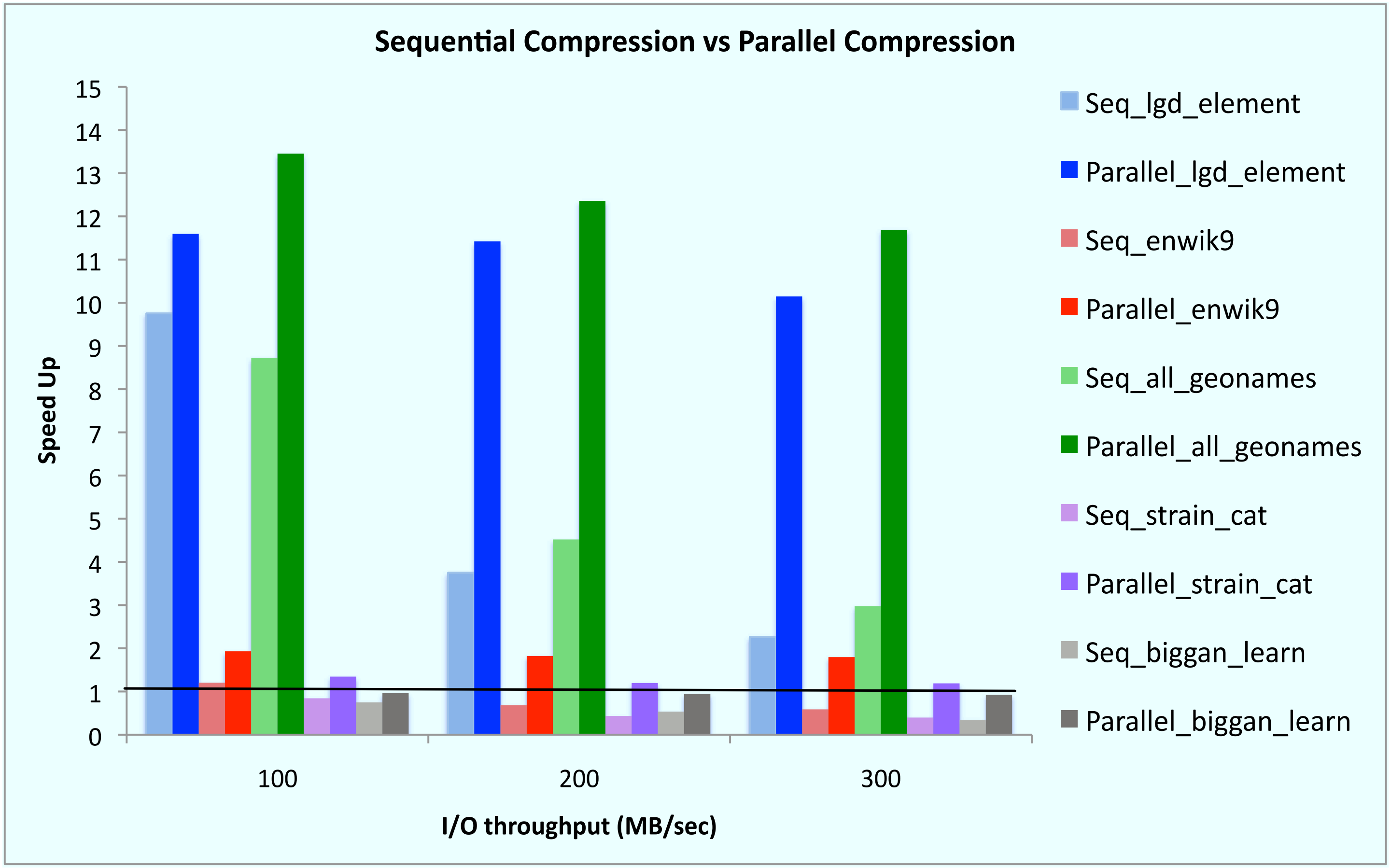
Nodes	Description
32 Compute nodes	Intel Xeon E5540 (2.53GHz, 4 cores) CPU *x* 2, 48GB memory, Broadcom NetXtreme-II (10 GbE)
8 Storage servers	Intel Xeon E3-1230 (3.2GHz, 4 cores) CPU, 8GB memory, Intel X520-DA2 (10 GbE)
1 Management server	AMD Opteron 6128 CPU (2GHz, 8 cores), 8GB memory, Intel X520-DA2 (10 GbE)

- Several files have been used to evaluate our proposal

File	Size	Category	Type	Comp. Ratio
lgd_element.rdf [19]	17GB	geographic	text	11.96
all_geonames.rdf [20]	6.3GB	geographic	text	12.76
enwik9.txt [21]	950MB	linguistics	text	1.75
strain_cat.txt [22]	433MB	earth science	float	1.3
biggan_learn.bvecs [23]	13GB	computer vision	multidata	1.05
tiny_metadata.bin [24]	38GB	computer vision	binary	8.13
dna_15.cel [25]	1.9GB	biology	numeric	1.52

- Speedup values by using 100MB/s, 200MB/s and 300MB/s throughputs
  - Higher IO parallelization with higher IO throughput

Evaluations



Speed Up= Original time/ Strategy time

Parallel Compression → The level of threading is 8 .

Evaluations

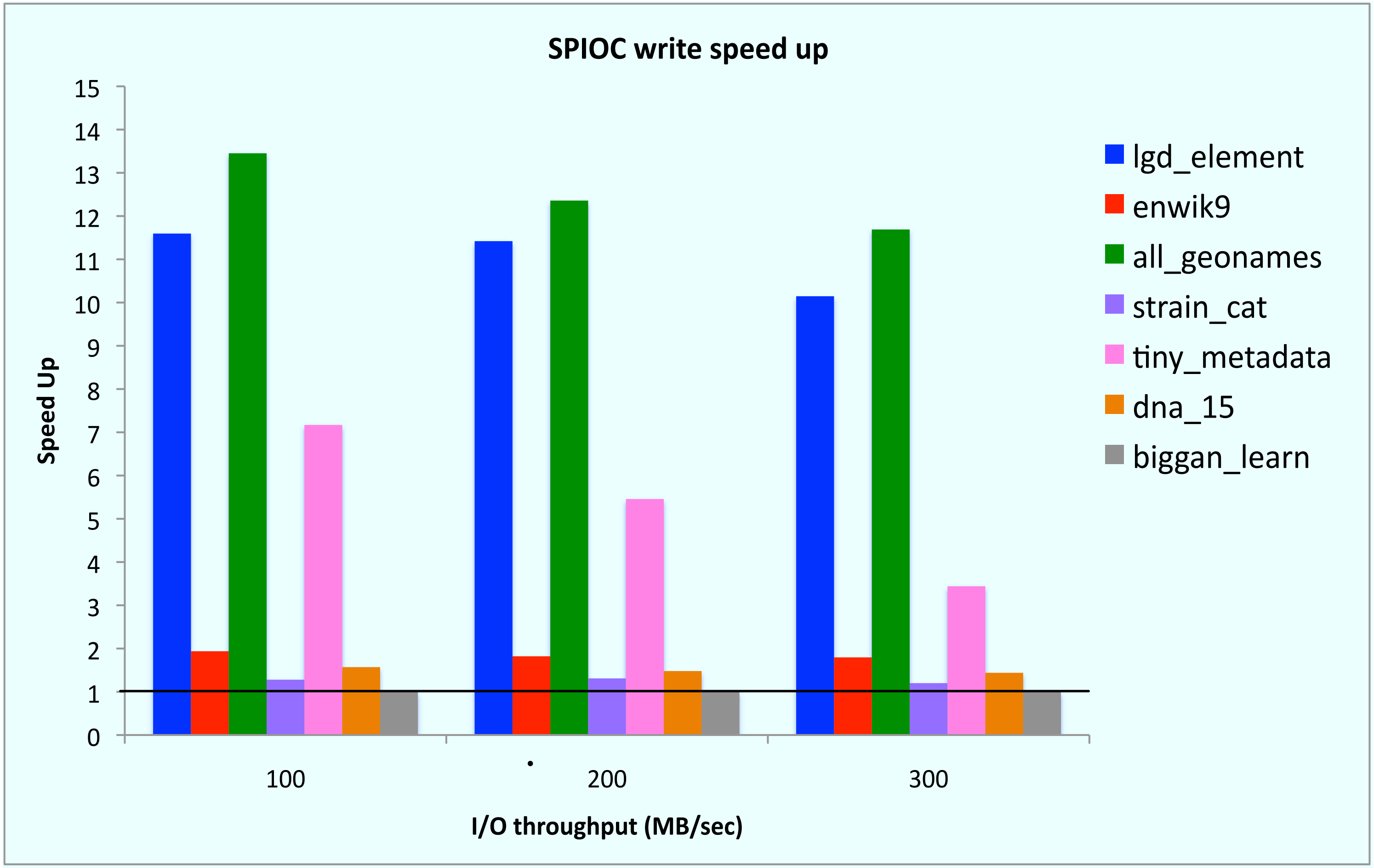
File/Strategy	100 MB I/O Speed Up		200 MB I/O Speed Up		300 MB I/O Speed Up	
strain_cat	estimated	real	estimated	real	estimated	real
Sequential	0.78	0.84	0.49	0.43	0.33	0.38
Parallel	1.28	1.34	1.21	1.24	1.20	1.21
all_geonames	estimated	real	estimated	real	estimated	real
Sequential	8.14	8.72	4.36	4.5	3.04	2.9
Parallel	12.58	13.45	12.25	12.35	11.97	11.68

Estimated and real speed up values for sequential and parallel compression.

The estimated values by SPIOC are very close to the real ones:
Error between 3%- 7%.

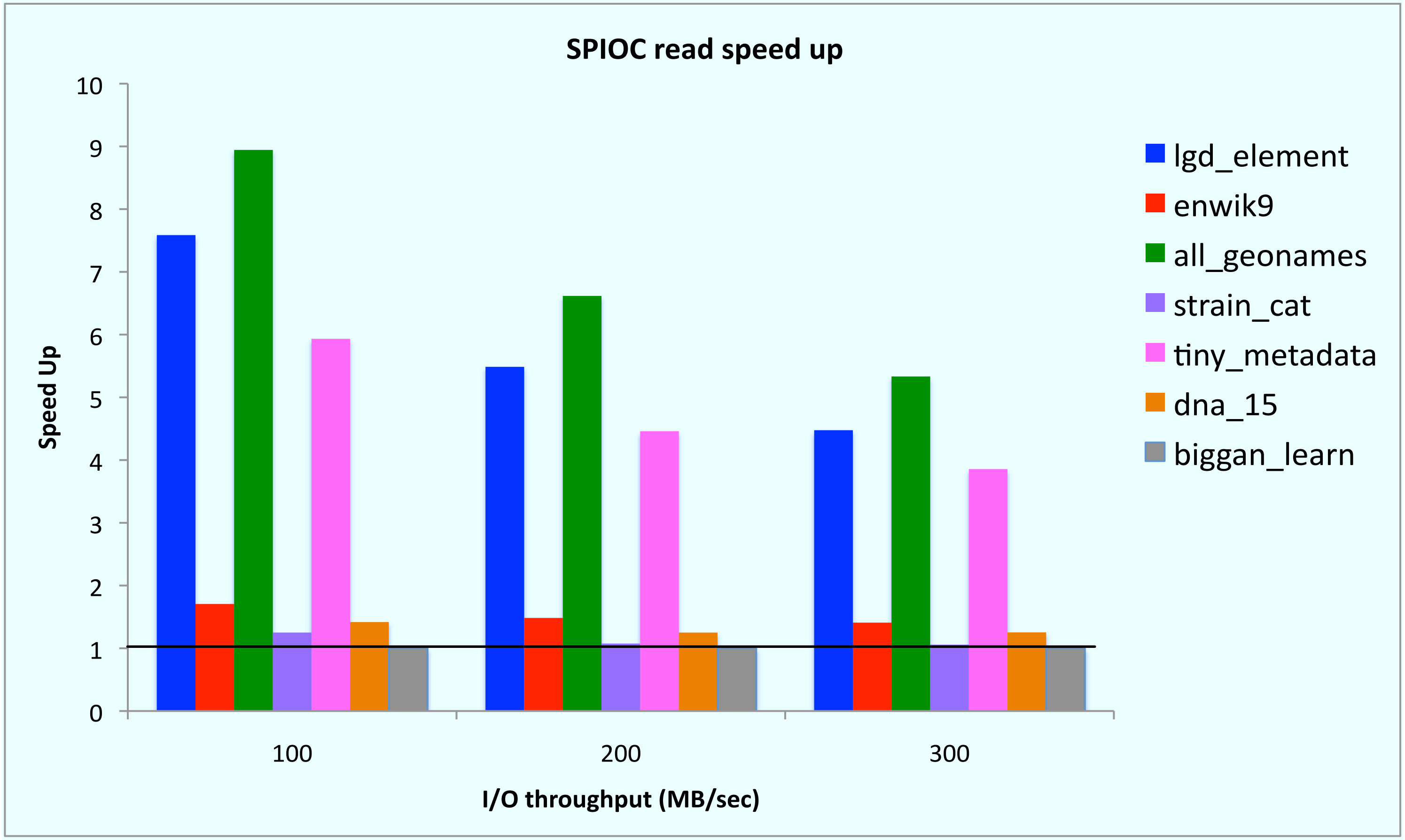
edinburgh data-intensive research

Evaluations



SPIOC → The level of threading is 8 .

Evaluations



SPIOC read speed up → The level of threading is 8 .

Conclusions

- 3 transparent compression strategies:
  - To improve the I/O performance in QoS enabled parallel storage system.
- *Sequential Compression* strategy: how the I/O operations could be improved by applying compression.
- *Parallel Compression* strategy: how to reduce the compression time by applying multithreading techniques.
- *Selectively Parallel I/O Compression* strategy:decides in run time whether to compress or not, and which technique apply (sequential or parallel).

Future work

- To detect the optimal value for the thresholds and threading level at run time :
  - depending on the characteristics of the computer nodes and files.
- To provide users the option to choose the compression criteria:
  - high-compression or high-speed.
- To apply different compression algorithms to the file's *chunks* depending on their data-types
- To apply our strategies to Papio collective I/O operations.
- To apply our strategies to other file systems.

Thanks

- Questions ?
- Email: rosa.filgueira@ed.ac.uk
- Compression study with synthetic files:
  - http://effort.is.ed.ac.uk/Compression/SyntheticResults.htm
- Pseudo codes:
  - Algorithm-1: http://effort.is.ed.ac.uk/Compression/SequentialCompression.pdf
  - Algorithm-2: http://effort.is.ed.ac.uk/Compression/ParallelCompression.pdf
  - Decompression-algorithm: http://effort.is.ed.ac.uk/Compression/ReadCompression.pdf