Discovering Structure in Unstructured I/O

Jun He[1,2], John Bent[3], Aaron Torres[4], Gary Grider[4], Garth Gibson[5], Carlos Maltzahn[6], Xian-He Sun[1]

[1]Illinois Institute of Technology, [2]New Mexico Consortium, [3]EMC, [4]Los Alamos National Laboratory,

[5]Carnegie Mellon University, [6]University of California Santa Cruz

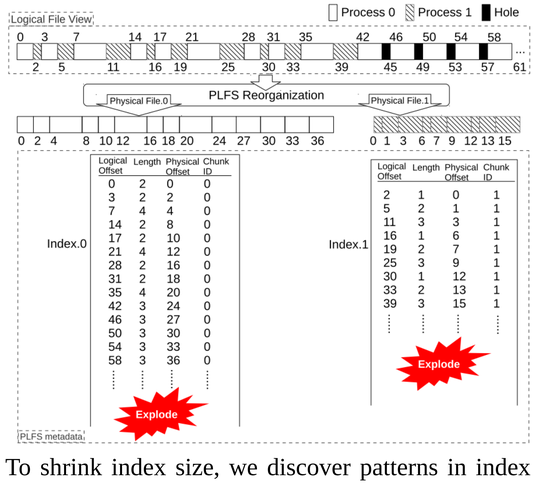
Motivation

- Extremely challenging storage requirements for exaflop supercomputers:
  - Checkpoints of 32 petabytes in size should complete in 300 seconds (DOE projection)
- PLFS (Parallel Log-structured File System) reduces checkpoint time by up to several orders of magnitude
  - By transparently transforming N-1 writes (N processes write to 1 file) to N-N.
- **However**, PLFS metadata (index) grows as the application size and number of data writes increase.
  - Considerable overhead of meatadata fetching
  - Considerable memory occupation by metadata
- Example: an anonymous application from LANL
  - Data: **4 GB**
  - PLFS metadata (on disk): 192 MB
  - PLFS metadata (in memory): **12 GB**

PLFS Index

PLFS transparently transforms shared-file writing into log-structured file-per-process writing. **Index** has a mapping between the bytes within the logical file and their location within the physical files.

The figure below shows an example of two processes writing to a traditional PLFS file. **Indices can become very large as the number of writes increases.**



To shrink index size, we discover patterns in index entries and represent them in a compact way.

For a simplified example, sequence (8,8,8,8,8,8,8,8,8,8) can be represented as 8^10 (ten eights). So the size is compressed.

Discovering Pattern Structure

Notation for Complex Patterns

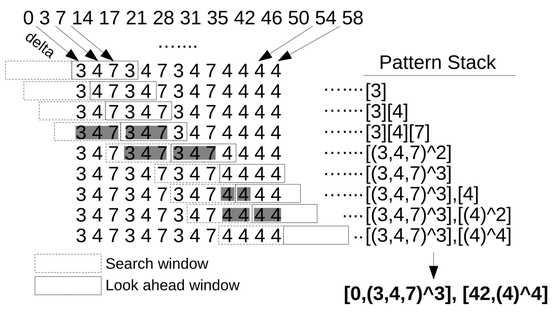
$$[i, (d[0], d[1], ...)^\wedge r]$$

- ***i*** is the first element of the original sequence.
- ***d[]*** (delta) is the repeating part of an array containing the distances of any two consecutive elements in the original sequence.
- ***r*** is the number of repetitions.

For example, (5, 7, 10, 12, 15) can be represented as [5, (2, 3)^2].

Discovering Pattern

An example to find the logical offset patterns in index.0 of the previous figure:



- New Index.0 using pattern structures:

Logical Offset	Length	Physical Offset	Chunk ID
[0,(3,4,7)^3]	[2,(0,-2,2)^2]	[0,(2,2,4)^3]	0
[42,(4)^4]	0	[24,3^4]	
	2		
	[3,0^4]		

- Time complexity: ***O(wn)***.
  - *w:* window size
  - *n:* length of input sequence

Look Up a Request in a Pattern

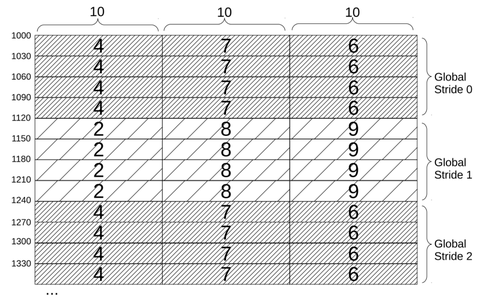
1. Assemble all indices at the time of opening PLFS file for read.
2. Request comes: read(offset=22, len=1).
3. Look up to what position offset=22 falls in the imaginary pattern matrix. For example, [0,(3,4,7)^3]

4. find the corresponding 6th length and physical offset of logical offset 22
5. Calculate and return the corresponding physical offset and length of the request.

Merging Isolated Patterns into a Global One

In order to further compress metadata, isolated per-process index patterns can be merged into a global one.

For example, in the figure below, all processes write with a fixed stride of 10 as shown.



4,7,6,2 and so on are PIDs.

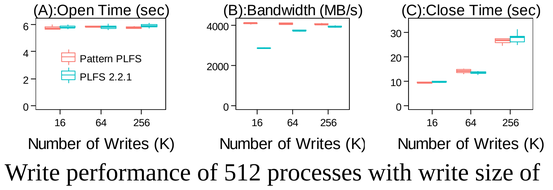
Blocks of same texture represent data area that is shared by a group of processes, e.g. global strides.

The writes from different processes in the figure above can be described by one global pattern:

id[]: [4,7,6,2,8,9,4,7,6,2,8,9]
logical: 1000,(30)^4
length: 10,(0)^4
physical: 0,(10)^4

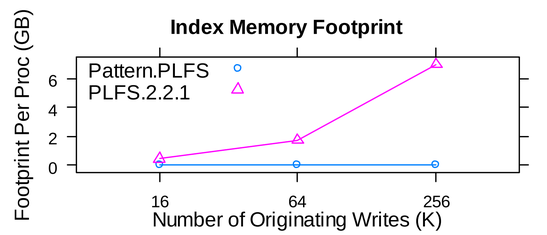
Evaluation

FS-TEST

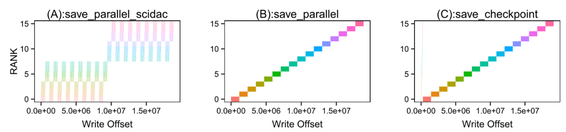


Write performance of 512 processes with write size of 4K.

Performance of uniform read (512 processes) and non-uniform read (256 processes) with originating write size of 4K.

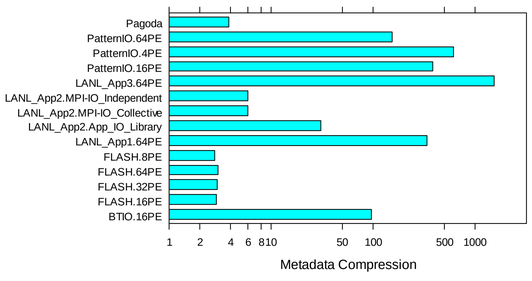


MILC



Write patterns of MILC. In-memory index compression rates by Pattern PLFS (higher is better): (A):37.0; (B):3.0;(C):3.6

Others



Conclusions

- This paper proposes efficient and practical techniques to discover structures from unstructured I/O operations, thereby enabling powerful I/O optimizations.
- Several orders of magnitude improvement in the size of the PLFS metadata.
- Up to 40% of write improvement
- Up to 480% of read improvement
- Other Possible Use Cases:
  - Pre-fetching, block pre-allocation, and index compression in SciHadoop.

Acknowledgement

The authors are thankful to Michael Lang (LANL) and Adam Manzanares (California State University) for their help toward this study. This work was performed at the Ultrascale Systems Research Center (USRC) at Los Alamos National Laboratory, supported by the U.S. Department of Energy DE-FC02-06ER25750. The publication has been assigned the LANL identifier LA-UR-12-25954.