Whamcloud

Xfast: Extreme File Attribute Stat Acceleration for Lustre

Qian Yingjin, Liu Ying
qian@ddn.com, emoly@whamcloud.com
Nov. 3rd, 2023

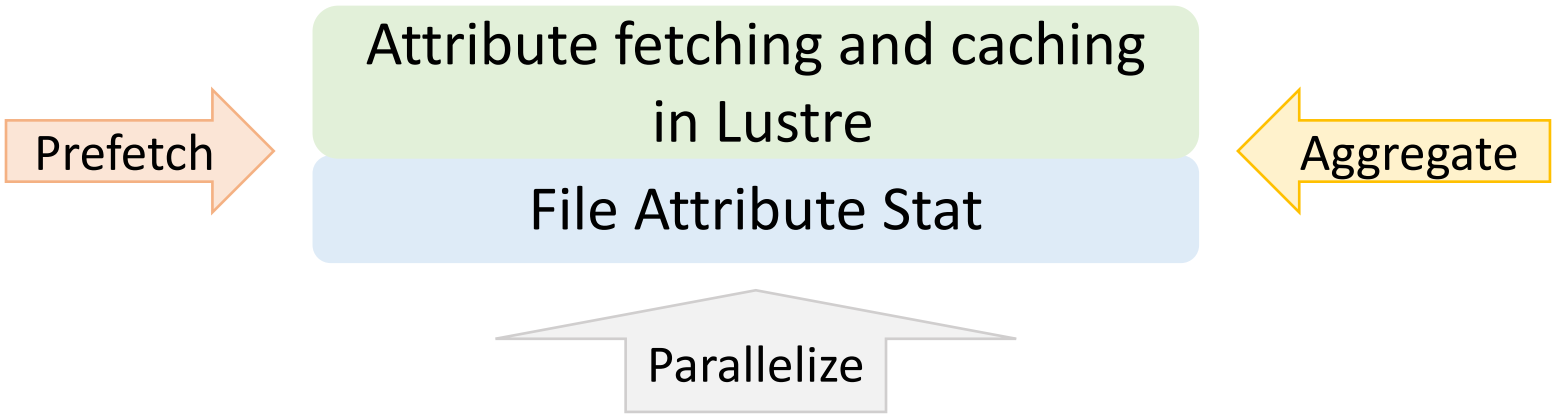
ddn

Outline

- Background and motivation
- Xfast design and implementation
  - Scalable statahead
  - Batch RPC engine
  - Subtree aggregate statahead(SAS)
  - Size on MDT(SoM)
  - Scale-out stathead
  - Thrashing avoidance
- Performance evaluation
- Conclusion and future work

Background and Motivation

- Data is growing at an extreme pace
  - 10,000,000+ files in a singe directory
- Many HPC applications suffer most from slow directory scans
  - Directory tree walks cost much time (minutes to hours)
- How to improve directory tree walks performance



File Attribute Stat

Whamcloud

- Serialized POSIX interface
  - Retrieval only operate on a single directory entry at a time;
  - The traversal of a directory with millions of entries can take tens of minutes to complete due to repetitive stat() calls.
  - Use predictable access patterns to **prefetch** metadata.
- POSIX semantics
  - Need to return the most recent file information when listing directories;
  - New statx() system call allows applications to request specific attributes to minimize unnecessary overhead.
  - **Reduce** the number of RPC calls per statx() operation and allowed us to implement lazy and strict *Size on MDT-feature* (SoM) for Lustre.
- Parallel prefetching of attributes
  - mpiFileUtils + {dfind, drm, dcp, ...}
  - Convert the serial stat() access from user process into **parallel** asynchronous operations.

Attribute Fetching and Caching in Lustre

Distributed lock manager (DLM)

- Protect data and metadata consistency;
- If a client holds a read lock, it can access the data or metadata locally, without concern that another client modifies it.

stat() path in Lustre

1. An RPC is sent to the MDT to acquire a lock;
2. MDT returns a protected read (PR) lock, along with metadata attributes and layout extended attribute(EA);
3. Send a ***glimpse*** PR lock request with the extent range [*0*, *EOF*] to OSTs to obtain the current file size and blocks attributes.

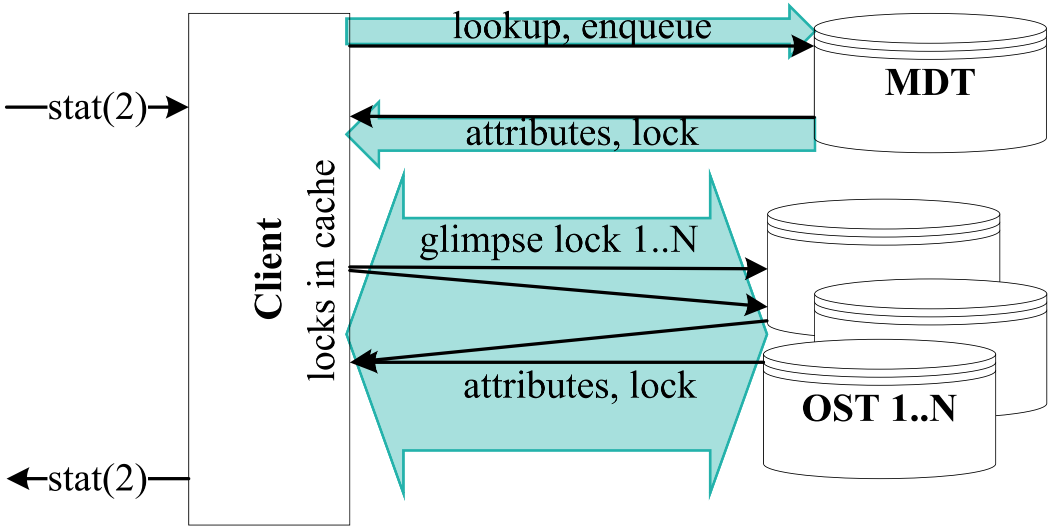
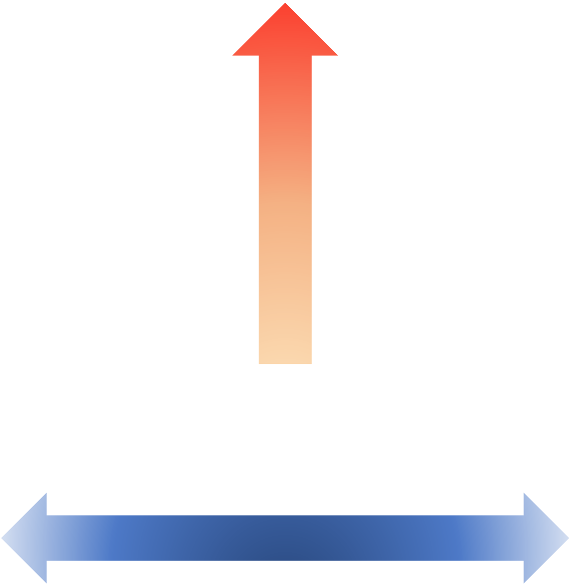


Figure: stat() workflow

Cached locks on the client protects the strong consistency for file attribute caching.

Xfast Design and Implementation

- Scalable statahead
- Batch RPC engine
- Subtree aggregate statahead(SAS)
- Size on MDT(SoM)
- Scale-out stathead
- Thrashing avoidance



Overview of released Lustre feature about xFast

Feature	Lustre version	Year
FLAT statahead	v1.8	2009
Asynchronous glimpse lock	v.2.2	2012
Lazy size on MDT(LSoM)	V2.12	2018
Strict size on MDT (SSoM)	new	-
Batch RPC engine	V2.14	2021
Batched statahead	V2.16	2023
Subtree aggregate statahead (SAS)	new	-
Scale-out statahead(pENT, pSTL and pSTH)	new	-
File naming pattern statahead	In merging	2023

Scalable Statahead

- **Flat statahead algorithm** (Lustre 1.8 in 2009)
  - Traverse a flat directory: **opendir()** followed by **readdir()** and **stat()**;
  - Launch a kernel statahead thread when kernel detects user stat() in readdir() order;
  - The statahead thread is notified to release its resources when the user process stops the directory traversal by calling **closedr()**.
- **Asynchronous glimpse lock (AGL) for size** (Lustre 2.2 in 2012)
  - Once obtain attributes form MDT, push it into AGL pipeline;
  - AGL thread scans its pipeline, send asynchronous glimpse RPC to OSTs to fetch file size.

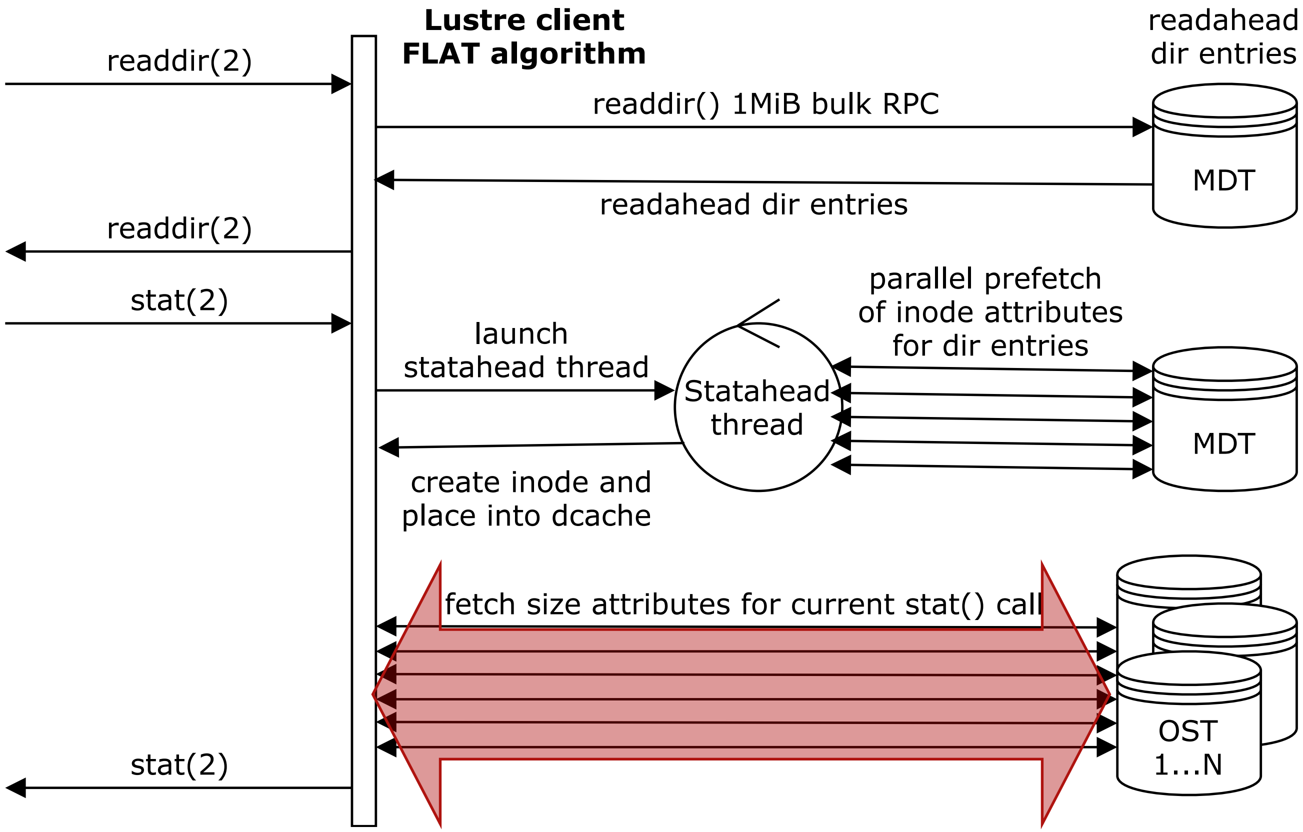


Figure: Simplifile statahead workflow for `ls -l`

Batch RPC Engine (Lustre 2.16 in 2023)

- Statahead batching packs several dentry names resulting from a readdir() call into one large batched RPC, which is transferred via bulk I/O.
  - Increase communication efficiency
  - Reduce the message size by compacting requests with a similar format.
  - *batch_max* controls the maximum number of items to batch in one aggregate RPC.
  - *statahead_max* controls the statahead window size, default 1024 (*batch_max* <= *statahead_max*)

Subtree Aggregate Statahead (SAS)

- Tools *find, du are* Depth First Search(DFS) access pattern.
- SAS: FLAT + DFS
  - It always starts with FLAT algorithm and if traversal process drills down into the first subdirectory, it changes into DFS mode.
  - It is controlled via ***statahead_max*** for a directory and via ***dmax*** for a new maximum subdirectory lookahead.

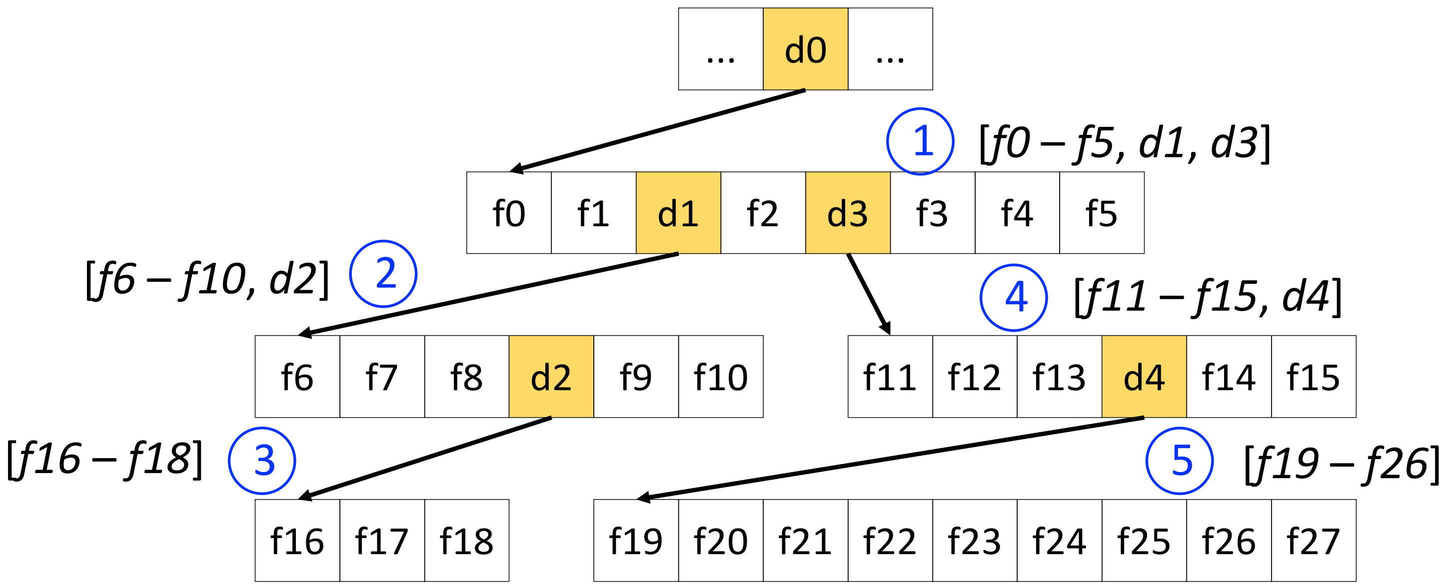


Figure: SAS algorith for DFS mode (*statahead_max* = 8, *dmax* = 3)

Size on MDT (SoM)

▶ Lazy SoM (LSoM, Lustre 2.12 in 2018)

- Reduces the number of RPCs required to fetch the size of a file, but cannot guarantee its accuracy.
  - Store the latest file size update and its block count as extended attributes on MDT, which can be accessible via a single RPC without accessing several OSTs
  - Update on the file *close*() and *truncate*() on MDT.

▶ LSoM → Strict SoM (SSoM)

- An entry is added into the Lustre changelog every time when a file is opened for write or being truncated.
- A dedicated Lustre client uses a ***lease lock*** to access these changelog records.
- A flag can be specified in stat() to return strict or lazy size information.

Scale-out Statahead

Whamcloud

- Combine Xfast with mpiFileUtils to provide scale-out performance for tree walks
  - Parallel stat on entires (pENT)
    - A single file is the minimal work set for the parallel tree walk.
    - Files within a directory can therefore be randomly distributed among different MPI ranks.
    - Break the sequential stat() order from readdir().
  - Statahead with limit (pSTL)
    - Trade-off strategy that balances parallelization and stata-head speedup .
    - Perform a local directory walk for the first ***stmax*** (default 256) files in a directory by FLAT algorithm
    - Enqueues the remaining entries into the global libCircle queue.
  - Statahead by hash division (pSTH)
    - Hashing the filename ensures that file names and file name sizes evenly partition the hash key space, especially for a larger directory.
    - Split stat() workload under a directory according to the hash space evenly (by ***segment_size***).

Thrashing Avoidance

- If statahead guesses the wrong access pattern, scarce memory and I/O bandwidth would be wasted.
- In this case
  - Statahead decreases the next statahead window size by a factor of 2
  - When it decreases to 1, it waits for the traversing process until it catches up to the current statahead position or exits and disables statahead processing
  - When the traversing process catches up, it enlarges the window size again.

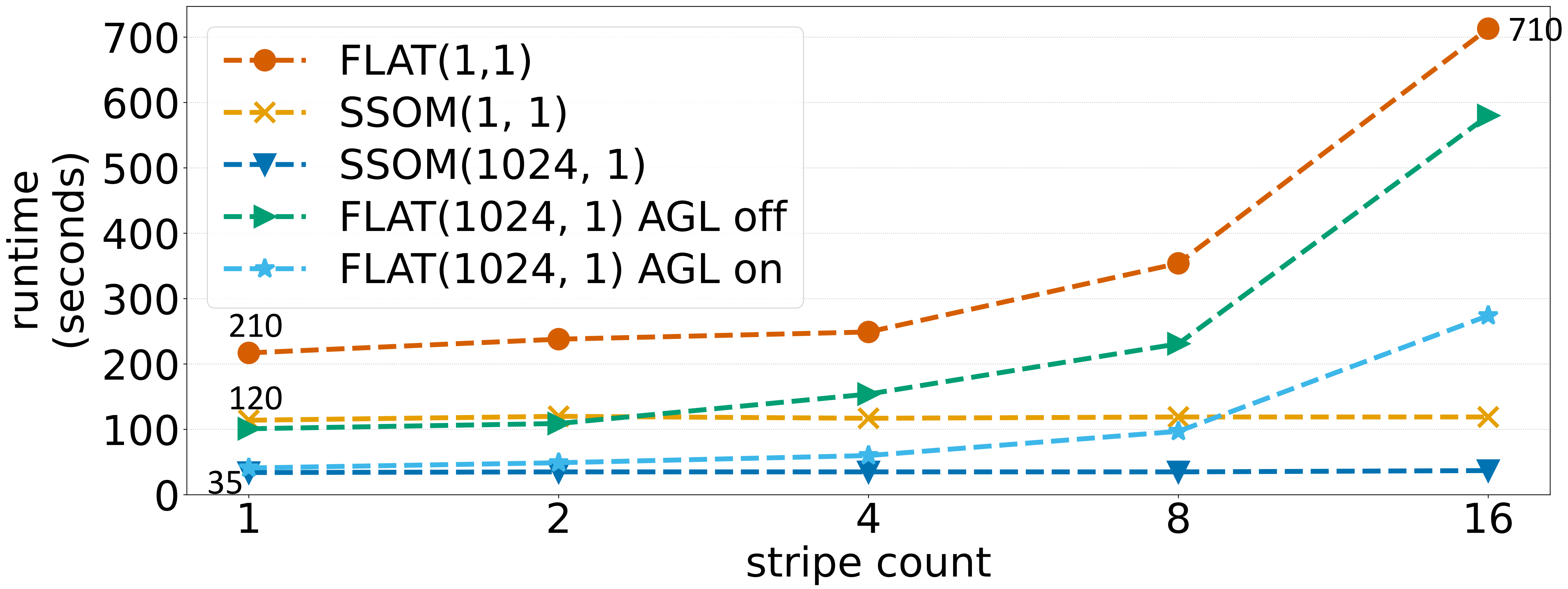
Performance Evaluation

- **Flat directory traversal**
  - Comparison of FLAT and SSoM
  - Client-side caching of file attributes
  - Network bandwidth impact including batching
- **SAS algorithm**
  - FLAT vs. SAS
- **Scale-out statahead**
  - pENT vs. pSTL vs. pSTH
- **IO500 mdtest**

- **Testing Environment**:
  - Lustre version: 2.14
  - Server: 1MDT, 8 OSTs (DDN AI400X Appliance (20x SAMSUNG 3.84 TB NVMe, 4X IB-HDR100))
  - Client:16 nodes (1x Intel Gold 5218 processor, 96 GB DDR4 RAM, CentOS 8.1 Linux)
  - Network: Infiniband IB-HDR100(by default) + 1 Gbps Ethernet interface
- The **Lustre Network Request Scheduler Token Bucket Filter (NRS-TBF)** is used to enforce RPC rate limitations to emulate **different server capabilities**.
- Tool **netem** is used to emulate different network conditions with delays of 1-10ms into the 1 Gbps Ethernet network.
- The tuple **XXX**$(\boldsymbol{i}, j)$ with $j \le i$ defines the combination of ***statahead_max=***$\boldsymbol{i}$ and ***batch_max***$=j$.

Comparison of FLAT and SSoM

ls -l command on a directory with 1M file entries on different stripe count between 1 and 16 OSTs



Client-Side Caching of File Attributes

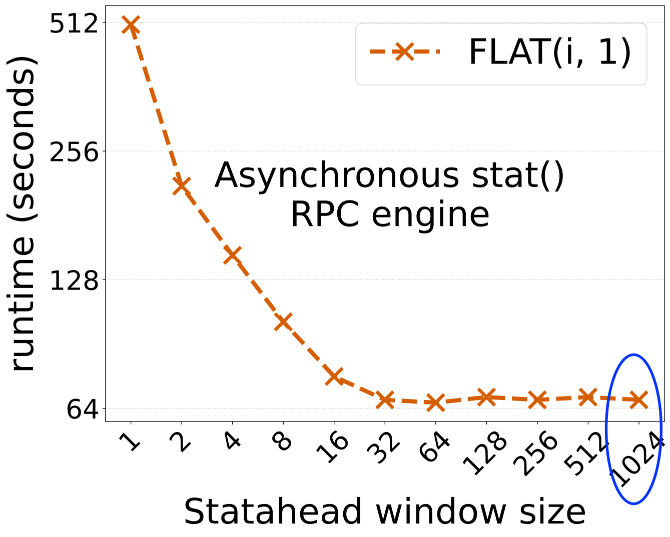
ls -l for 1K to 1M files for FLAT(1,1) on a single OST

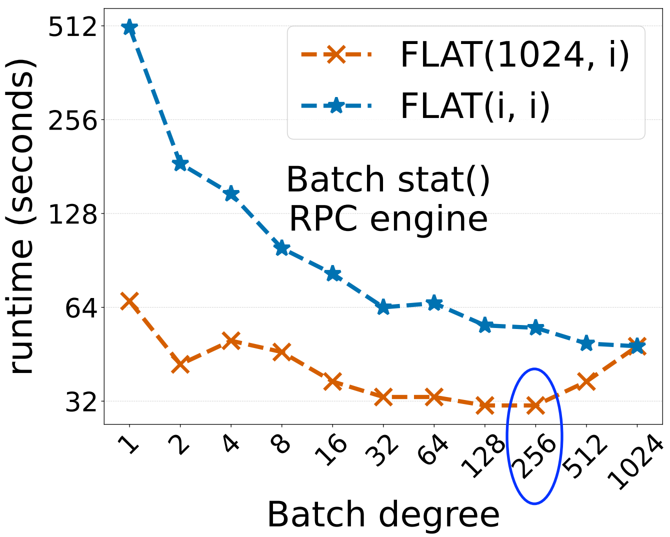
# files	Cold cache (s)	Warm cache (s)
1,000,000	221	15.7
100,000	21.5	0.993
10,000	2.26	0.102
1,000	0.253	0.015

Statahead performance with write conflicts
ls -l with FLAT(1024, 1)

# nodes	0	2	4	6	8	10
Time (s)	42	62	115	170	228	229

Network bandwidth impact including batching

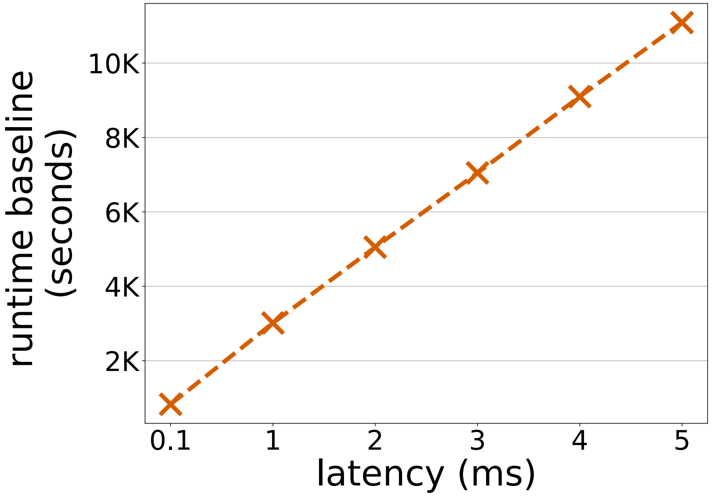


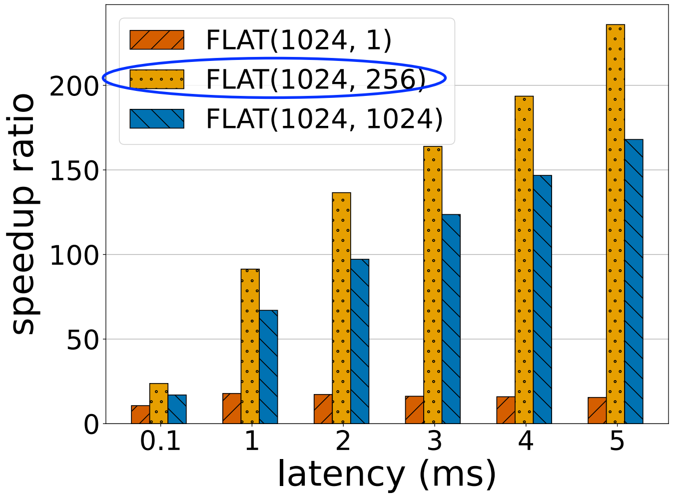


Impact of *statahead_max* and *batch_max*

FLAT(1024, 256)

1 Gbps Ethernet, stripe_count =1, AGL enabled





Speedup ratio for high network latencies

(compared to FLAT(1,1))

SAS Algorithm(FLAT+DFS) Evaluation

Traversed a directory containing 16 Linux source trees (linux-5.12-rc5) via the command *find src -uid 0* with *dmax*=16.

find using FLAT vs. DFS mode

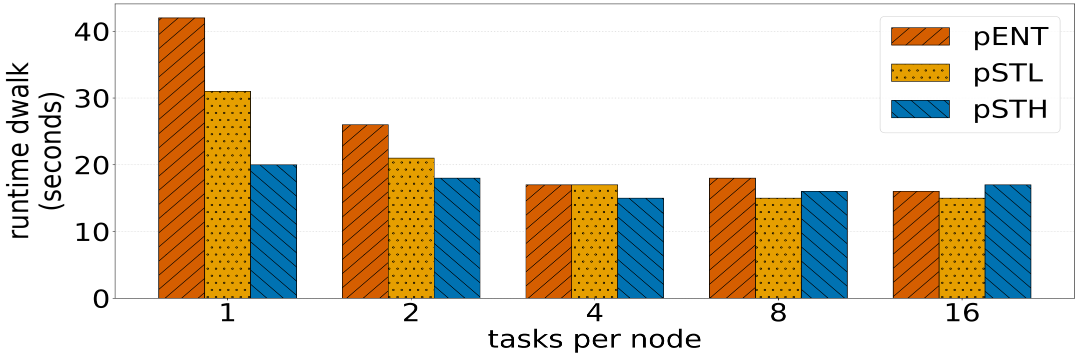
Mode	Thread count	stat() RPCs	Time (s)
FLAT(1024, 1)	75,697	1,219,008	114
FLAT(1024, 256)	75,665	108,439	112
SAS(1024,1)	1	1,219,544	73
SAS(1024, 256)	1	21,108	68

FLAT vs. DFS for different network latencies

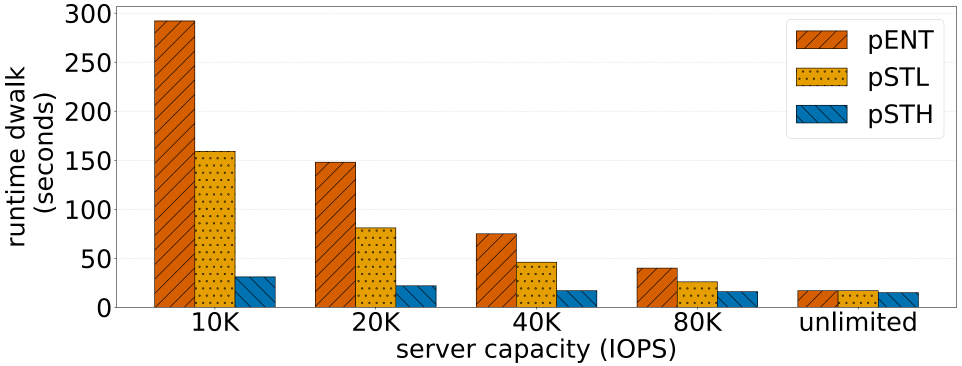
Latency(ms)	0.1	1	2	3	4	5
Baseline	1083	3931	6653	9354	12036	14730
FLAT(1024,1)	290	916	1564	2187	2823	3439
FLAT(1024,256)	286	876	1479	2056	2649	3245
SAS(1024, 1)	156	404	691	986	1261	1548
SAS(1024, 256)	163	314	491	674	866	1057

Scale-out Statahead Evaluation

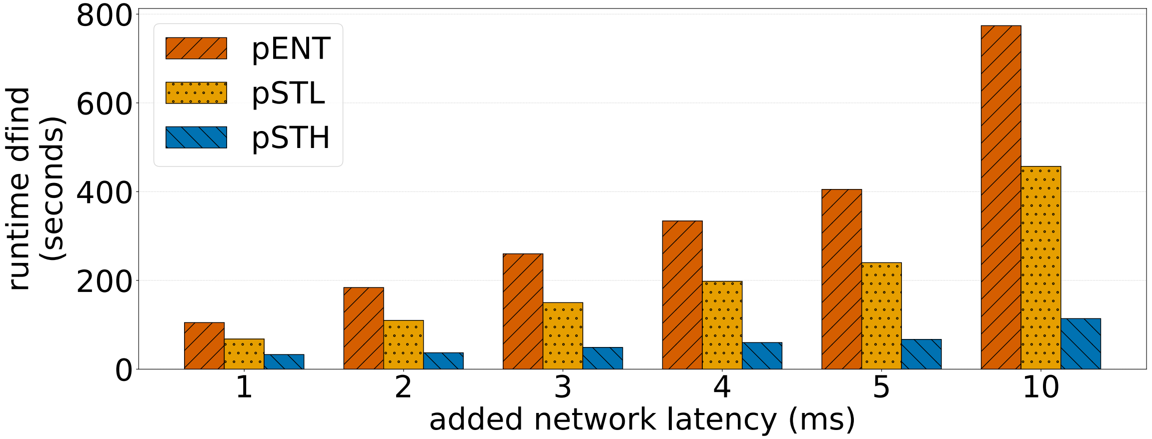
Ran ***dwalk*** and ***dfind*** commands on a flat directory with 1M files and a directory including 16 Linux source code trees. (*stmax*=256, *segment_size*=4096)



Statahead combined with mpiFileUtils on 16 nodes



dwalk on resource-limited metadata servers



dfind runtimes with various network latencies

IO500 mdtest - Sustained Performance Enhancements

Whamcloud

Storage Platform ES400NV ES400NVX ES400NVX2

8 x CPU/node → 12 x CPU/node(1.5x)

1 x EDR/node → 1 x HDR200/node(2x)

PCIGen3 NVMe → PCIGen4 NVMe (2x)

Performance improvement goes beyond what hardware upgrades can achieve

	Pre-SC19	SC19	ISC20	ISC22	SC22	ISC23	ISC23/PreSC19
IOR Easy Write	25.88	28.62	37.56	55.95	58.07	57.88	2.2x
IOR Easy Read	39.94	41.72	45.95	83.86	77.56	79.08	2.0x
IOR Hard Write	2.78	2.96	2.77	5.02	5.27	5.38	2.0x
IOR Hard Read	8.99	42.19	40.81	39.73	49.36	50.77	5.6x
Find	1,735.41	810	1,698.00	6,248.55	12628.78	13,229.11	7.6x
Mdtest Easy Write	143.88	152.84	157.22	270.04	312.9	344.70	2.3x
Mdtest Easy Stat	455.03	451.97	453.51	740.01	1,278.50	1,276.31	2.8x
Mdtest Easy Delete	88.52	132.76	135.09	223.61	272.64	311.16	3.5x
Mdtest Hard Write	32.33	79.65	90.47	119.41	157.4	199.36	6.1x
Mdtest hard Read	44.92	172.59	169	194.33	238.82	391.09	8.7x
Mdtest Hard Stat	20.41	449.93	446.75	514.36	1,214.03	1,105.33	54.1x
Mdtest Hard Delete	16.35	75.15	76.94	101.98	122.44	112.58	6.8x
Bandwdith	12.68	19.65	21.02	31.10	32.90	33.43	2.6x
IOPS	91.41	207.62	232.69	368.48	544.23	603.39	6.6x
Score	34.05	63.87	69.93	107.05	133.81	142.03	4.1x

https://io500.org/submissions/view/657

Conclusion and Future Work

- Xfast can significantly improve the performance of common directory operations.
- Future work
  - Other statahead patterns and optimizations
    - File naming statahead pattern
    - Given an input file name list, do batched statahead.
    - Combining with statahead and readahead.
  - Improve prefetching pipeline

Whamcloud

Thank You!

DDN STORAGE