TSUE: A Two-Stage Data Update Method for an Erasure-Coded Cluster File System

Zheng Wei[1], Jing Xing[1], Yida Gu[12], **Wenjing Huang**[12], Dong Dai[3], Guangming Tan[1], Dingwen Tao[1]

[1]Sate Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.

[2]University of Chinese Academy of Sciences, Beijing, China.

[3]University of Delaware, Newark, DE, USA

Email: weizheng@ncic.ac.cn

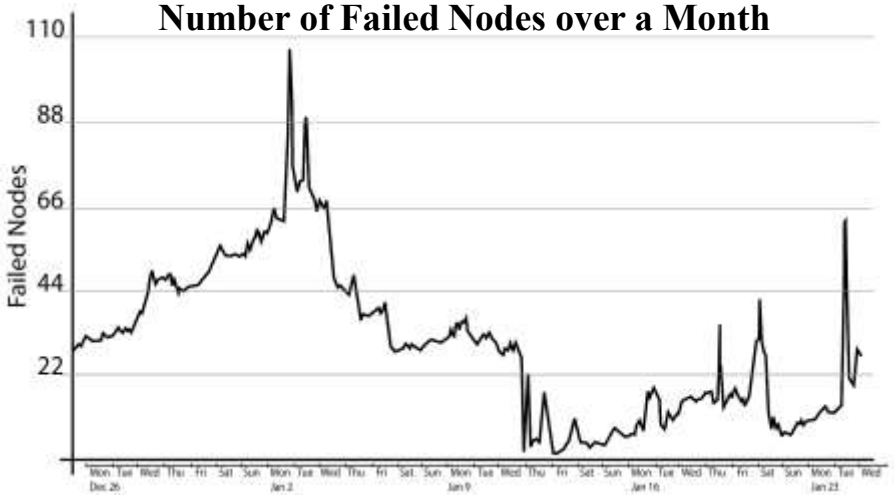
taodingwen@ict.ac.cn

IDC. Worldwide IDC Global DataSphere Forecast, 2024–2028: AI Everywhere, But Upsurge in Data Will Take Time, 2014.5.

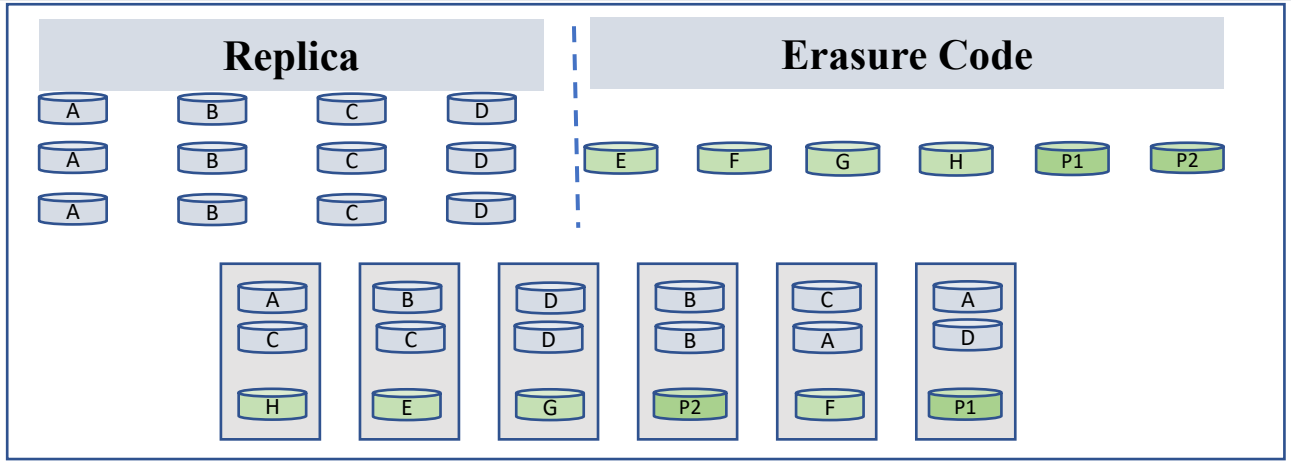
Sathiamoorthy M, Asteris M, Papailiopoulos D, et al. Xoring elephants: Novel erasure codes for big data[J]. arXiv preprint arXiv:1301.3791, 2013.

Background: Erasure Code is widely adopted by…

Storage Scale: the volume of global data soared & the scale of data in the AI4S era



Reliability guarantee:
the regular failure of commercial components



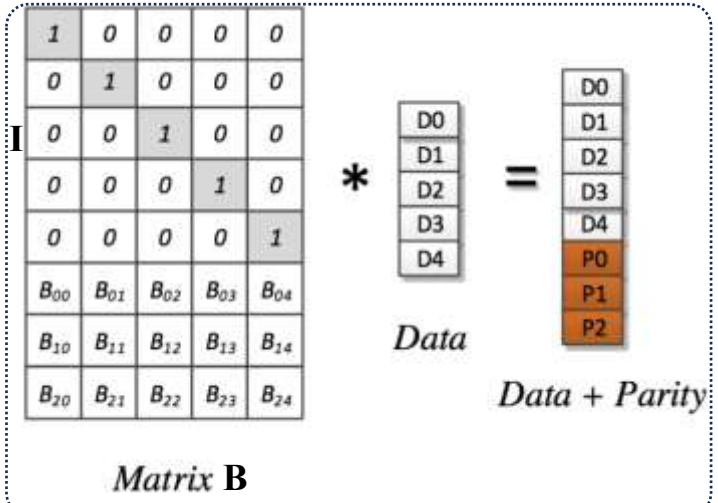
- Replica: storage overhead **200%**
- Erasure Coding: storage overhead **12.5-50%** **Erasure**

Method	Capacity	HDD Node		All-Flash Node	
		HDD 400$/16TB	Storage Node 3000$/Node	U.2 SSD 1200$/7.6TB	Storage Node 9000$/node
Replica	**3EB**	**78643200$**	**49152000$**	**539267657$**	**337042285$**
Erasure Coding	**1.5EB 4+2**	**39321600$**	**24576000$**	269633828$	168521152$
	1.15EB 28+4	**30146560$**	**18841600$**	206719268$	129199542$

The erasure code mechanism is increasingly adopted by both open-source and commercial storage systems!

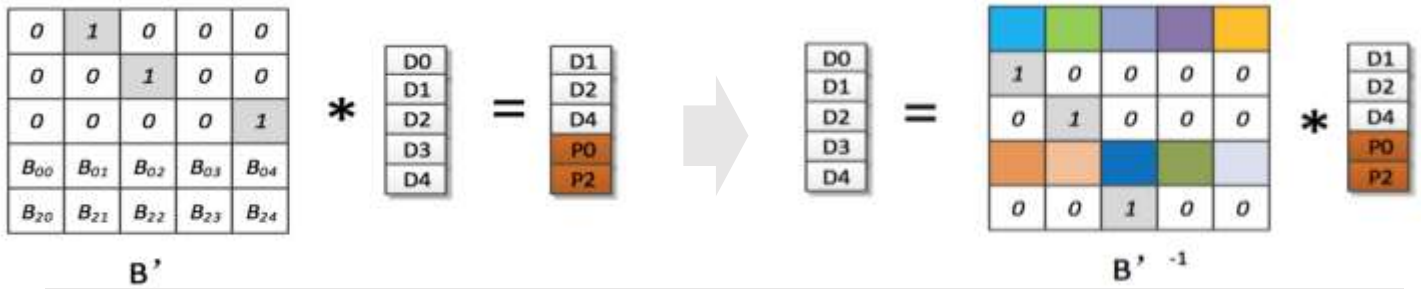
Background: The Issues of Erasure Coding

ENCODING



DECODING

The Invertibility of Full-Rank Matrices $B^{'} * B^{'-1} = I$

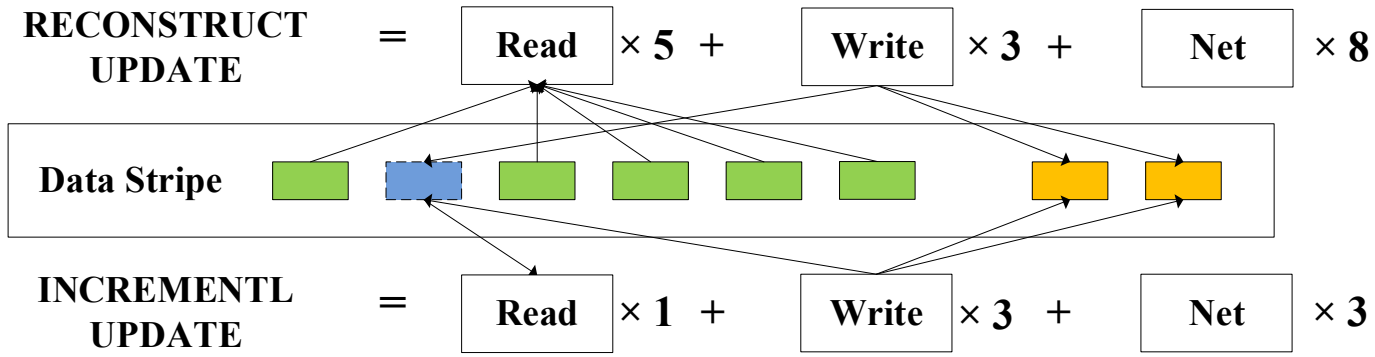


A k-order invertible matrix enables the construction of any K blocks within a stripe using K original data blocks, and vice versa.

UPDATING

Mainly Update Approaches:

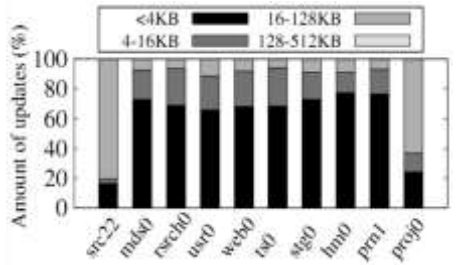
- **Reconstruct Update**
  - full stripe coverage
- **Incremental Update**
  - small grained coverage



Unable to support updates of original data, instead of log or copy-on-write(COW), but introduce overhead!

The Analysis of Access Characteristic

The Scene of Updating



Volume	Workload Type	No. of Writes	WSS (GB)	Updated WSS(%)	Update Writes(%)
src22	Source control	805955	20.17	99.57	99.68
mds0	Media server	1067061	3.09	29.27	95.77
rsrch0	Research	1300030	0.36	69.53	97.41
usr0	Home directory	1333406	2.44	42.54	96.08
web0	Web/SQL server	1423458	7.26	37.25	96.23
ts0	Terminal server	1485042	0.91	49.84	95.65
stg0	Web staging	1722478	6.31	21.04	97.82
hm0	HW monitor	2575568	2.31	73.16	93.21
prn1	Print server	2769610	80.9	18.55	73.43
proj0	Project directory	3697143	3.16	56.67	98.89

In the MSR-Cambridge trace, over 90% of operations are update operations, and more than **60%** of these updates have a granularity no larger than **4 KB**.

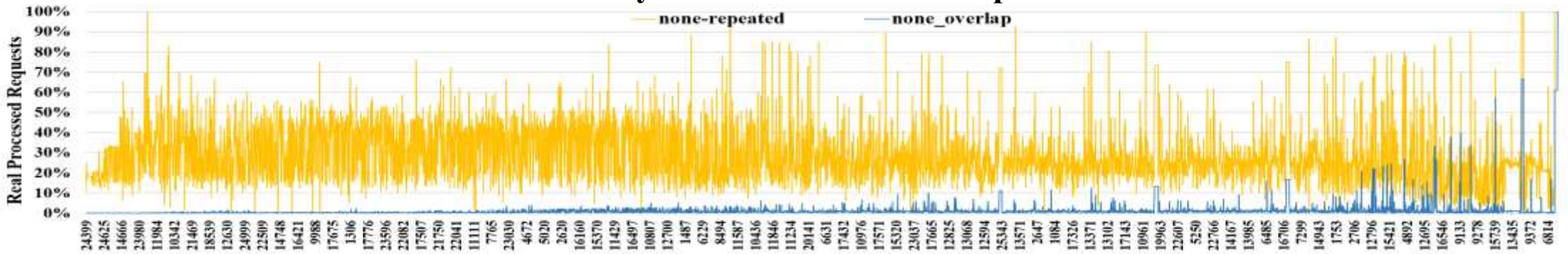
	Update Writes(%)	<4KB(%)	<16KB(%)
Ali-Cloud	75	**46**	60
Tencent Cloud	69	**69**	88

In the block traces from Alibaba Cloud and Tencent Cloud, over 69% of operations are update operations, among these, 46%-69% have a size no exceeding 4KB.

In real-world applications, a large number of fine-grained update operations are present. Incremental update method is suited for fine-grained update seene.

The Analysis of Update Operations

The Locality characteristics of the update

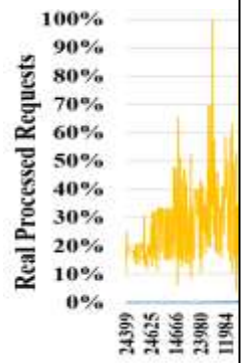


In Tencent cloud block trace, It has significant characteristics of temporal and spatial locality.

The Utilization of Spatial Locality

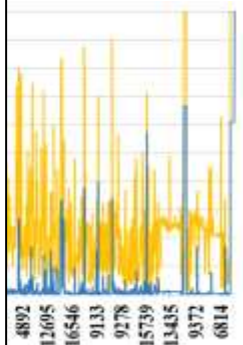
$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ \vdots \\ P_M \end{bmatrix} = \begin{bmatrix} \partial_{11} & \partial_{12} & \dots & \partial_{1K} \\ \partial_{21} & \partial_{22} & \dots & \partial_{2K} \\ \partial_{31} & \partial_{32} & \dots & \partial_{3K} \\ \vdots & \vdots & \ddots & \vdots \\ \partial_{M1} & \partial_{M2} & \dots & \partial_{MK} \end{bmatrix} \cdot \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ \vdots \\ D_K \end{bmatrix} \quad (1)$$

The Analysis of Update Operations



$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ \vdots \\ P_M \end{bmatrix} = \begin{bmatrix} \partial_{11} & \partial_{12} & \dots & \partial_{1K} \\ \partial_{21} & \partial_{22} & \dots & \partial_{2K} \\ \partial_{31} & \partial_{32} & \dots & \partial_{3K} \\ \vdots & \vdots & \ddots & \vdots \\ \partial_{M1} & \partial_{M2} & \dots & \partial_{MK} \end{bmatrix} \cdot \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ \vdots \\ D_K \end{bmatrix} \quad (1)$$

In T … cality.



The Utilization of Temporal Locality

$$P_1^n = P_1^{n-1} + \partial_{11} * (D_1^n - D_1^{n-1}) \quad (2)$$

$$P_1^n = P_1 + \partial_{11} * \left((D_1^n - D_1^{n-1}) + \ldots + (D_1^1 - D_1) \right) \quad (3)$$

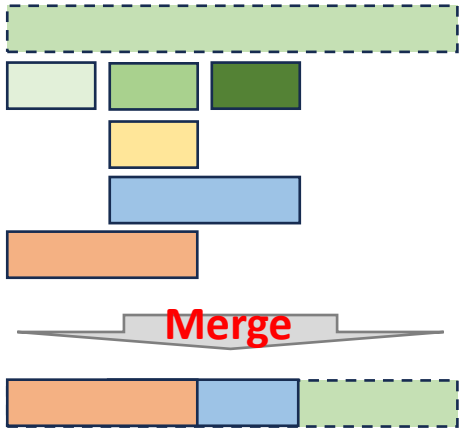
Merge

$$P_1^n = P_1 + \partial_{11} * ((D_1^n - D_1)) \quad (4)$$

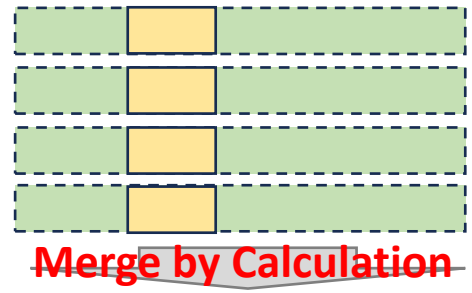
Data Blks: Merge Direct, Parity Blks: Merge with XOR

The Utilization of Spatial Locality

Intra-Block



Inter-Block

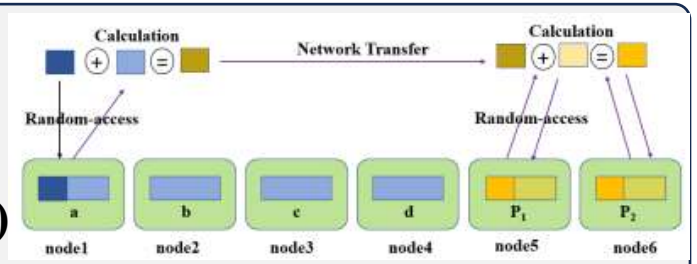


Merge by Calculation

$$\begin{bmatrix} P_1^n \\ P_2^n \\ P_3^n \\ \vdots \\ P_M^n \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ \vdots \\ P_M \end{bmatrix} + \begin{bmatrix} \partial_{11} & \partial_{12} & \partial_{14} \\ \partial_{21} & \partial_{22} & \partial_{24} \\ \partial_{31} & \partial_{32} & \partial_{34} \\ \vdots & \vdots & \vdots \\ \partial_{M1} & \partial_{M2} & \partial_{M4} \end{bmatrix} \cdot \begin{bmatrix} (D_1^n - D_1) \\ (D_2^n - D_2) \\ (D_4^n - D_4) \end{bmatrix} \quad (5)$$

The Challenge of Incremental Update Method

- **High Update Latency**
  - The lengthy update path
  - The inherent nature of random access
    - ✓ HDD: millisecond latency (seeking + rotation + data transfer)
    - ✓ SSD: there is big gap between sequential access and random access



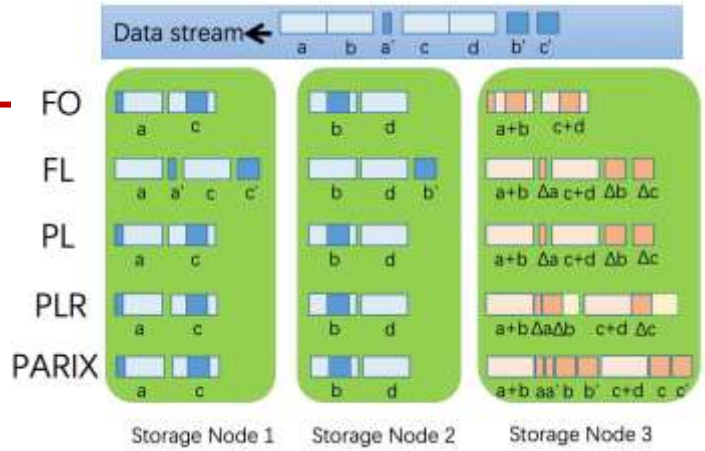
- **Low Update Throughput**
  - Fine-grained random-access constrains the enhancement of update throughput
  - ✓ HDD: there is a performance gap of two orders of magnitude
  - ✓ SSD: there is a performance gap of several times

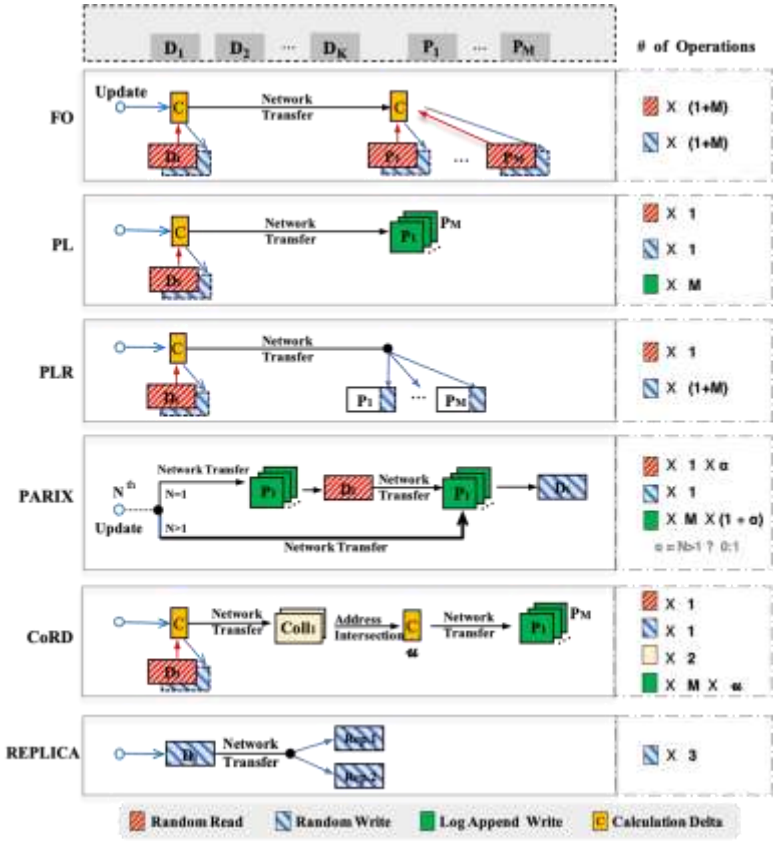
- **Consistency Issue due to Parity Log**
  - ✓ Log loss and prolonged log recycling lead to secondary data loss
  - ✓ The exist of log will prolong the recovery efficiency

- **Low Lifespan**
  - HDD: random-access cause frequently head movement
  - SSD: fine-grained random-access leads to frequently erase of flash

The SOTA Update Method

- FO
  - Data block & Parity block: In-place update
  - Lengthy update path
  - Random-access
  - no additional read/write ops & no log files
- PL
  - Data block: in-place update
  - Parity block: introduce parity log,
    - Transfer random-access into sequential access
    - Log was protected by write_lock, concurrent→serial write
    - Log has impact on recovery→consistency issue
- PLR
  - Data block: in-place update
  - Parity block: reserved place to keep parity log in blocks
    - avoid random-access during recycle process of log
    - Log appending like concurrent random-access (introduced)
    - More disk fragment (space management issue)
- PARIX
- CoRD

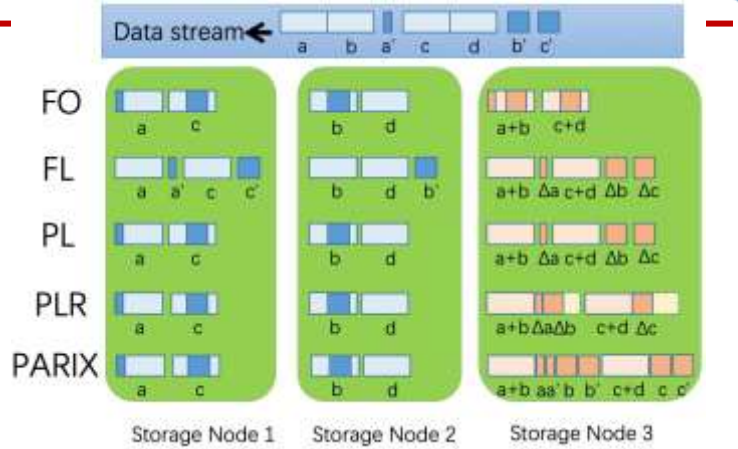


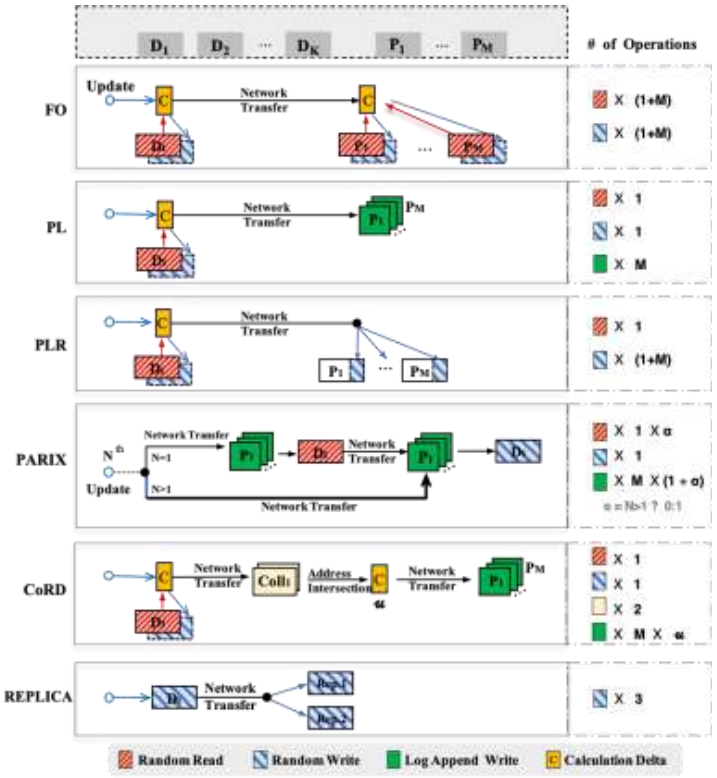


The SOTA Update Method

- PARIX (designed for data warehouse)
  - Data Block: in-place update (need to transfer update request)
  - Parity Block: like PL, but Parity log keep original data
    - Suit for data warehouse, which present temporal locality
      - Forward update request to parity log directly
      - Bypass the calculation of parity delta into recycle of parity log
      - Avoid partial read overhead of data blocks
      - Avoid calculation overhead during in-place update
    - Introduce 2x network latency
    - Need bigger space to store log data

- CoRD (designed for minimizing update traffic)
  - Data Block: in-place update, introduce buffer
  - Parity Block: Parity Log
    - Utilize collector to collects and aggregates the deltas of updated parts to reduce update traffic
    - Calculate parity delta of multiple intersection updates for same location across various data blocks within same stripe
    - a single log buffer → bottleneck
      - overlook parallelism and throughput considerations





The SOTA Update Method: summary

- **Data Block: in-place update → time-consuming process to calculate parity delta**
- **Parity Block:**

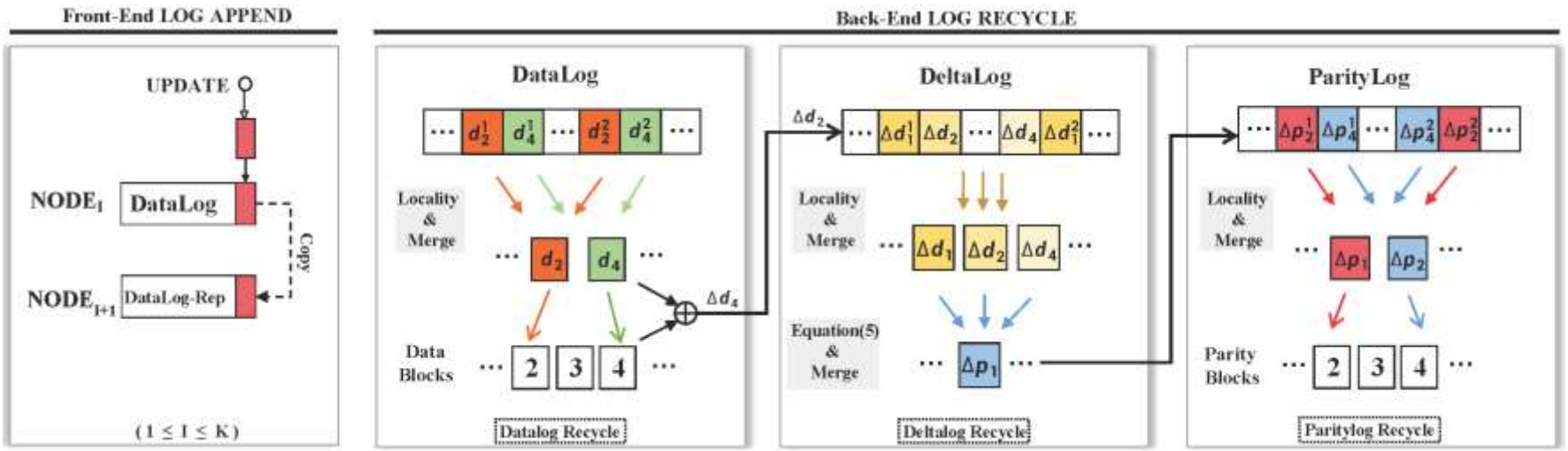
- **FO: in-place update**
  - **HDD unfriendly, SSD friendly**
- **PL: parity log**
  - **Consistency issue**
  - **Appending is protected by write-lock**
- **PLR: parity log with reserved space**
  - **Appending ops like concurrent random-access**
  - **Recycling and appending are mutually exclusive**
  - **More fragment**

- **PARIX:**
  - **Introduce 2x latency for scene without temporal locality**
  - **More log space to store forwarded original data**
- **CoRD:**
  - **Overlooks parallel and throughput consideration**
  - ✓ **Reduce network traffic by calculation**

CONCLUSION

1. Long **Latency:** **in-place update** of data blocks is time-consuming process (**long update path**).
2. Low **Throughput-issue:**
   - 2.1 Random-issue: **The local characteristics of data access have not been fully utilized.**
   - 2.2 Lack of **concurrency** design: single log is performance bottleneck and is not suitable for high-concurrency scenarios involving operations such as append and recycle
3. **Necessary of Log:**
   - 3.1. Logs is meaningful: transform random I/O operations into sequential ones, improving appending performance!
   - 3.2. Logs is also harmful: lead to consistency problems.
   - 3.3. Replica is best choice: the update mechanism of the replication is the simplest and most efficient!
4. network traffic issue(CoRD): Merge calculations during forwarding process reduce the volume of data transferred

Overview of TSUE



- Reduce random-access involving in update process
  - Utilize Spatial-temporal locality to reduce random-access overhead in three-layer log structure

- TSUE: Two-Stage Data Update Strategies for Erasure Code
  - Two-stage update process with swift recycle mechanism
    - ✓ introduce data log: instead of in-place update of data blocks → Low Latency
    - ✓ recycle log asynchronous & log content is temporal → no consistency introduced by the log
  - 3-layer Log: organized by data characteristic & spatial-temporal locality & recycle pipeline → high throughput & lifespan
    - Data log & delta log & parity log
  - Log Pool Structure: high concurrent between appending and recycling & adaptive workload-aware

TSUE: Two-Stage Update method for Erasure Code

- Two-Stage Data Update Method with Swift Recycle Mechanism
  - Front-End(Synchronous appending)
  - Back-End(Asynchronous recycling)

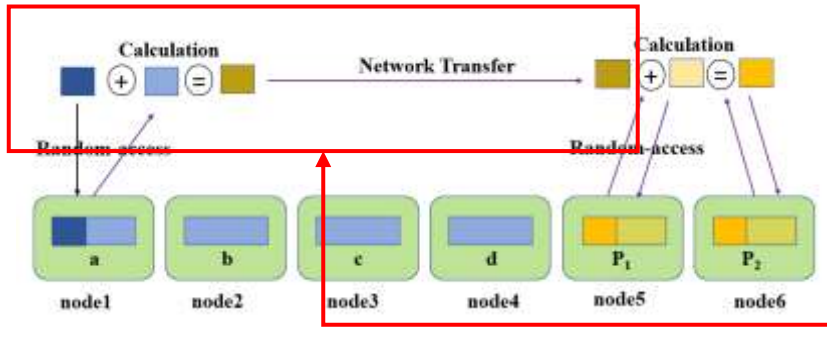
Front-End update process

- introduce **data log for the data block**
  - Convert random-access into sequential access
  - instead of in-place update of data blocks
    - Avoid time consuming process of perform write-after-read process to calculate data delta (original & new)

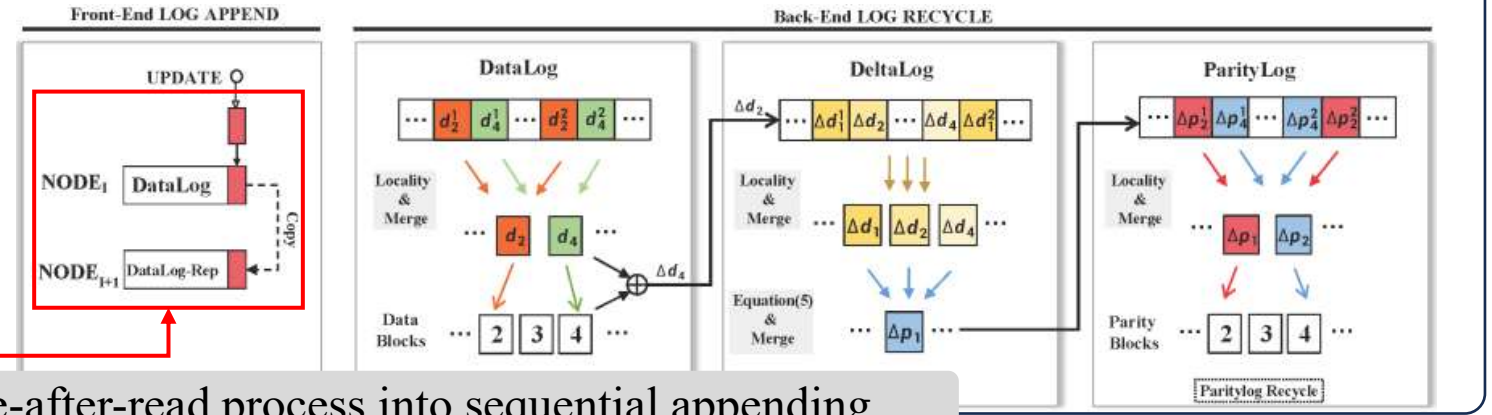
Back-End update process

- **Swift recycle** mechanism (vs PL PLR PARIX CoRD)
  - recycle log asynchronous
  - log content is temporal
    - **vs** PL PLR PARIX CoRD
      - ✓ Log is recycled rapidly and timely
      - ✓ minimal influence on data recovery

Traditional update process



TSUE' update process



Low Latency: Transfer time-consuming write-after-read process into sequential appending

TSUE: utilize 3-layer Log to reduce random I/O

- **3-Layer Log: Data log & delta log & parity log**
  - data characteristic & spatial-temporal locality & recycle pipeline → high throughput & lifespan

Data Characteristic

- **Raw data**
  - Overwrite directly according on order
  - Used to calculate data delta
- **Data delta**
  - Perform XOR with each other
  - Same for m parity blocks in same stripe
- **Parity delta** ($\partial_{ij} * Data\_delta$)
  - Perform XOR with each other
  - Specially for each parity block

Spatial-Temporal Locality

- Spatial Locality
  - Completely overlapping
  - Partial overlap
  - Adjacent in position
  - Close in location
- Temporal Locality
  - Completely overlapping
  - Raw data * [parity|data] delta

Recycle Pipeline

- Pipeline in Layer-Log
  - Reduce the number of I/OS step by step
- Parallel in log units
- Parallel in blocks
  - Two-level index
  - Organize updates in blocks

Principle of update process in data log and pairty log

$$P_1^n = P_1^{n-1} + \partial_{11} * (D_1^n - D_1^{n-1}) \quad (2)$$

$$P_1^n = P_1 + \partial_{11} * \left((D_1^n - D_1^{n-1}) + \ldots + (D_1^1 - D_1)\right) \quad (3)$$

$$P_1^n = P_1 + \partial_{11} * ((D_1^n - D_1)) \quad (4)$$

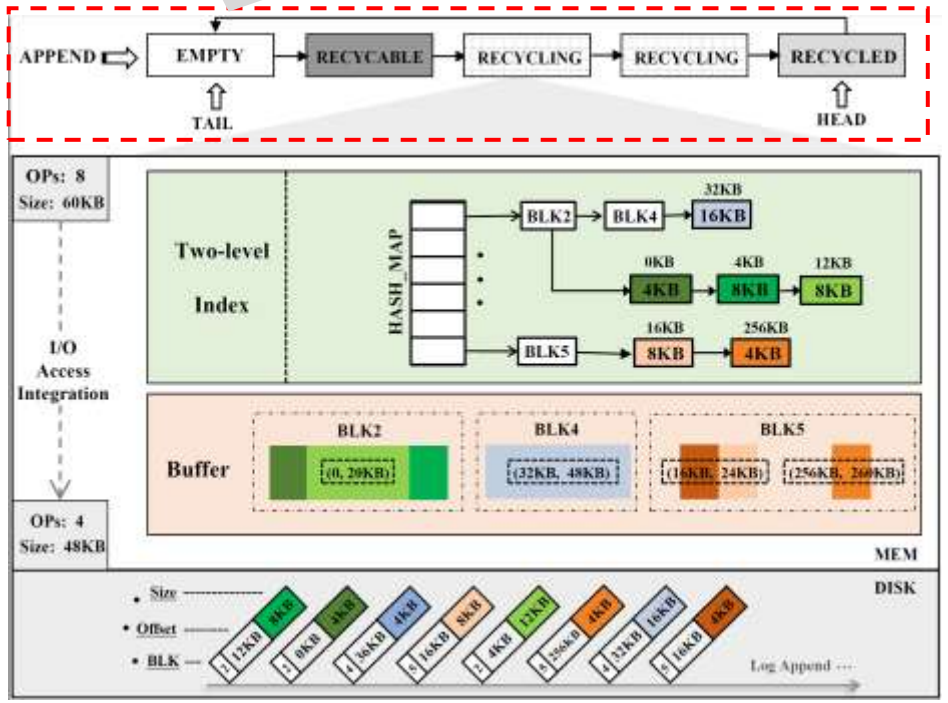
Princple of update process in delta log

$$\begin{bmatrix} P_1^n \\ P_2^n \\ P_3^n \\ \vdots \\ P_M^n \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ \vdots \\ P_M \end{bmatrix} + \begin{bmatrix} \partial_{11} & \partial_{12} & \partial_{14} \\ \partial_{21} & \partial_{22} & \partial_{24} \\ \partial_{31} & \partial_{32} & \partial_{34} \\ \vdots & \vdots & \vdots \\ \partial_{M1} & \partial_{M2} & \partial_{M4} \end{bmatrix} \cdot \begin{bmatrix} (D_1^n - D_1) \\ (D_2^n - D_2) \\ (D_4^n - D_4) \end{bmatrix} \quad (5)$$

Same location in multi blocks

TSUE: Log Pool structure

FIFO-based log pool structure



- **Adaptive FIFO-based log pool structure**
  - **FIFO-based & adaptive & quota-schedule**

FIFO-based Log pool structure

- Multiple log units in log pool
  - Each log unit equipped with two-level index
    - The insert update record is organized by index
  - Just one log unit is active
  - Multiple filled full log units are recycled in parallel
    - Blocks are mapped into single recycle engine based on hash

adaptive update process

- Minimize the occupation of system memory by logs
  - The number of log units is according on the workload

The schedule of quota for log pool

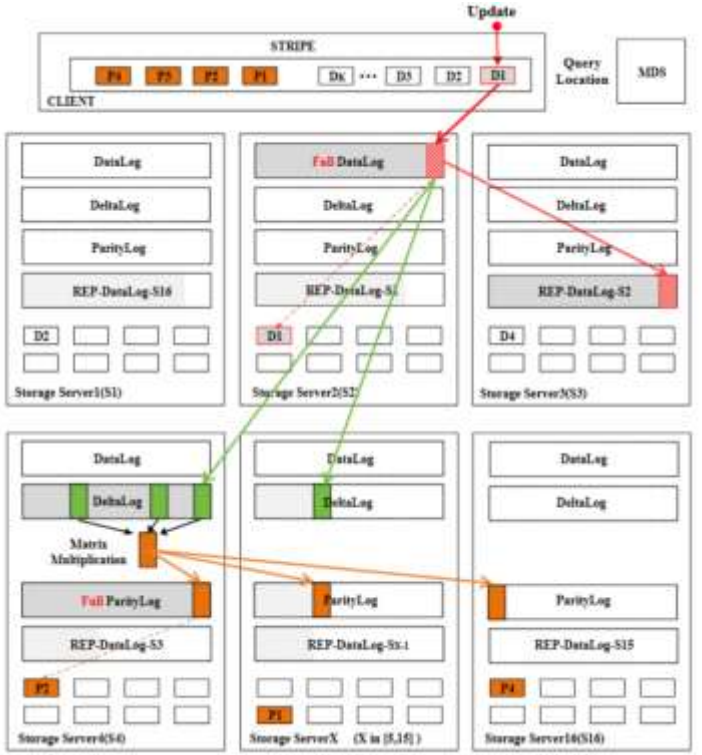
- Access skew
- Break down quotas to the light-load log pool

TSUE achieve maximum update performance with limited memory resources.

Implements of TSUE

Architecture and Configuration

- ECFS: Erasure-coded file system (self-develop)
  - Client: write(encoding), read(degrade read)
  - OSD: block storage, update
  - MDS: file/entry management, location management
- TSUE
  - Implemented in OSD
  - Data log, delta log, parity log
    - Data log: 2 replica(SSD), 3 replica(HDD)
    - Delta log: keep data delta, 1 for each disk
    - Parity log: keep parity delta, 1 for each disk
  - Distinguish write or update in Client
    - HDD: 1 logpool/disk, 8 recycle_thread, 2-20 log units
    - SSD: 4 logpool/ssd, 8 recycle_thread, 2-20 log units



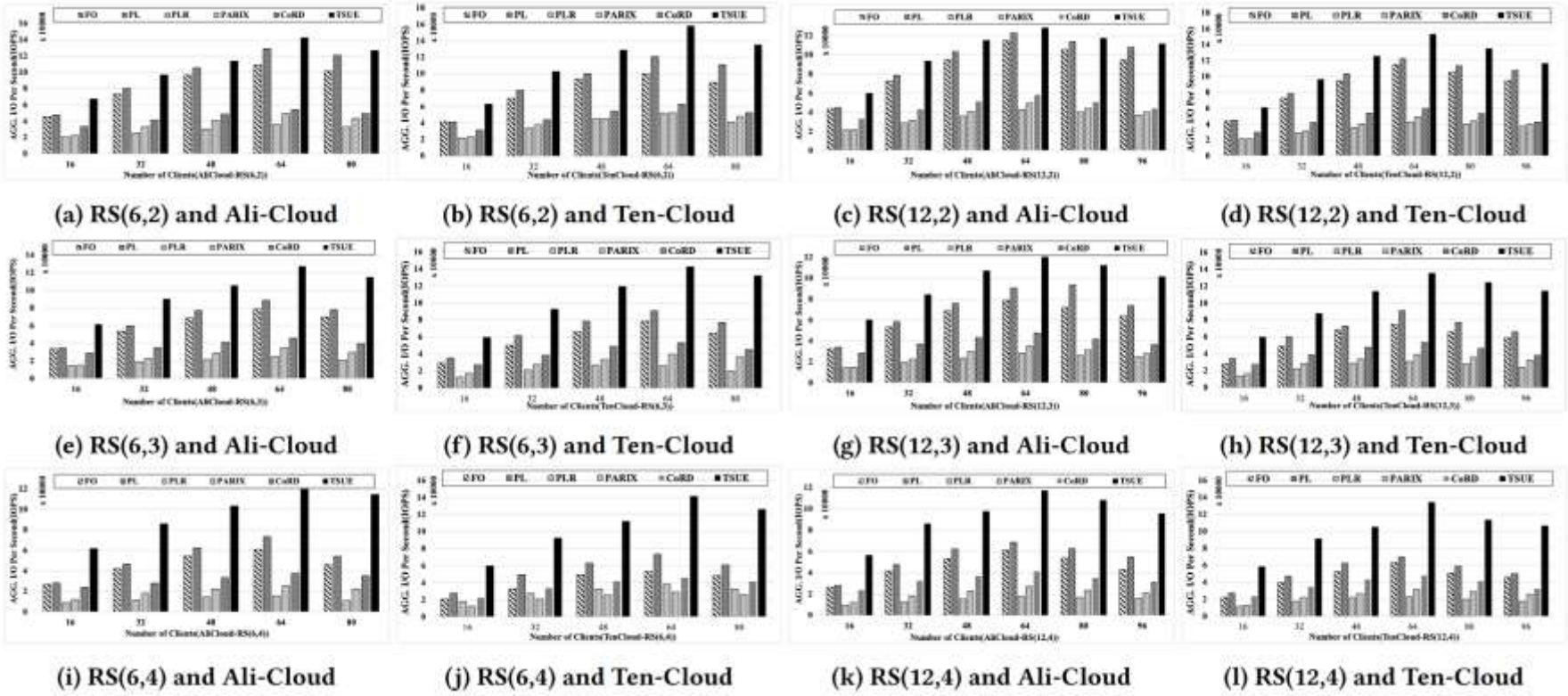
Update Procedure

- Step1(append of data log): append update request into data log and its' replica, insert the index, update is finished;
- Step2(recycle of data log): FULL data log is appended into recycle list; obtain merged and combined data from index, read old data, calculate data delta and forward the corresponding delta log(local: k, k+1);
- Step3(recycle of delta log): obtain multiple data deltas for same location across multiple blocks in same stripe, calculate M parity delta and forward corresponding parity log (without storage access)
- Step4(recycle of parity log): obtain overlapping parity deltas, and read the old parity data in big arrangement, and calculate the new parity data.

Evaluation

Chamemleon Cloud Environment(16 nodes)

- 2 intel Xeon 8380 CPU, 256GB memory(3200MT/s 16GB*16) ; 400GB Intel SATA SSD
- 200Gb/s Mellanox NIC (but, connected by 25Gb/s ethernet switch)



(a) RS(6,2) and Ali-Cloud (b) RS(6,2) and Ten-Cloud (c) RS(12,2) and Ali-Cloud (d) RS(12,2) and Ten-Cloud

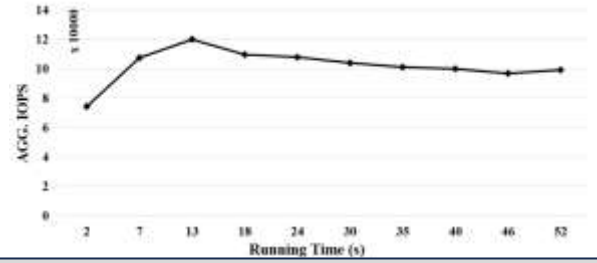
(e) RS(6,3) and Ali-Cloud (f) RS(6,3) and Ten-Cloud (g) RS(12,3) and Ali-Cloud (h) RS(12,3) and Ten-Cloud

(i) RS(6,4) and Ali-Cloud (j) RS(6,4) and Ten-Cloud (k) RS(12,4) and Ali-Cloud (l) RS(12,4) and Ten-Cloud

TSUE demonstrates the best overall performance.
When the M value increases, the performance advantage becomes more pronounced,
TSUE is well-suited for scenarios where the M value is greater than 2.
TSUE+'s performance improves steadily with an increasing number of clients.

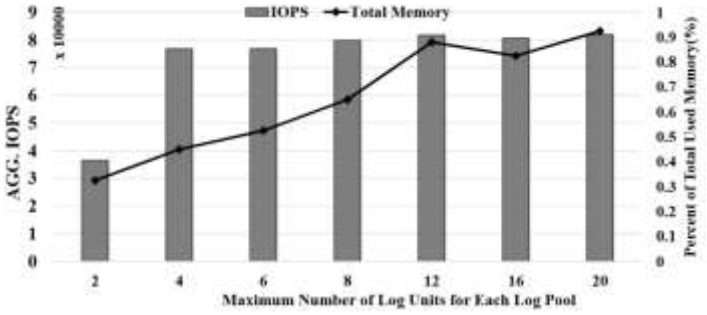
Evaluation

Log Recycle Overhead



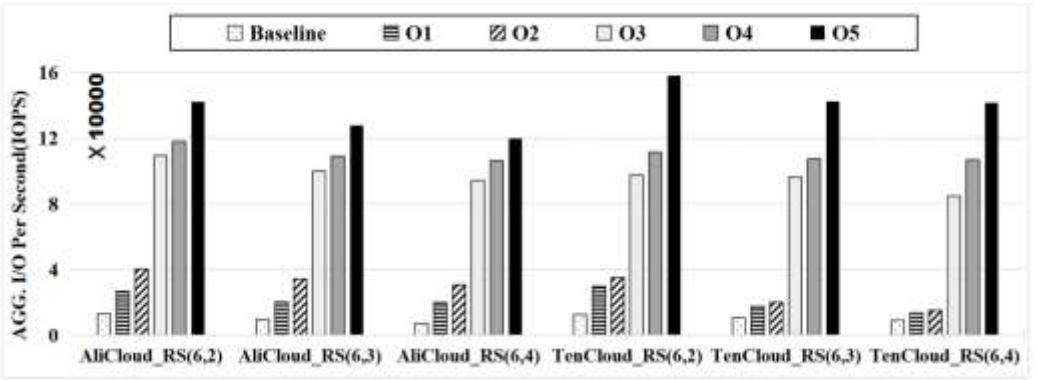
The update performance initially rises, subsequently declines, and ultimately stabilizes after 18 seconds.
The data suggests that the influence of the back-end log recycle process on update performance is negligible.

Memory usage



In memory-constrained environments, the number of log units is set to 4, to reach higher performance with a maximum memory allocation of 1 GB for a single SSD.

Breakdown Analysis



Baseline: Utilizes DataLog and ParityLog to recycle data.
- O1: Utilizes spatio-temporal locality in data logs.
- O2: Utilizes spatio-temporal locality in parity logs.
- O3: Introduces log pool structure to manage log.
- O4: Configure 4 log pools for each SSD.
- O5: Introduces DeltaLog to reduce network load

1.Just only in the log structure which support append and recycle in parallel, the spatial-temporal locality can be utilized effectively to improve recycling performance.
2. Delta log obtain 30% improvement by reduce network traffic.

Evaluation

1.TSUE exhibits the lowest number of read/write and overwrite operations among all methods.
2. The network traffic generated by TSUE is only about 60% of that by other mechanisms, and slight more than that of CoRD.
3. The read/write volume of TSUE is bigger than that of PARIX and CoRD due to the existence of 3-layer log.
4. The overwrite volume generated by TSUE is only high than that of PARIX which without recycle log.

TABLE I: Storage Workload and Network Traffic

METHOD	READ/WRITE		OVERWRITE (Write Penalty)		NETWORK TRAFFIC (GB)
	Num.	Volume (GB)	Num.	Volume (GB)	
FO	9,739,320	894	4,869,660	447	447
PL	13,726,649	1,252	4,869,660	447	447
PLR	8,860,540	1,240	7,493,637	440	447
PARIX	6,439,527	471	1,456,968	96	454
CoRD	2,913,936	365	1,094,691	271	276
TSUE	**2,750,796**	**645**	**401,466**	**223**	**290**

• **NOTE:** Replaying Ten-Cloud Trace in RS(6,4). The data log and delta log in TSUE+ use the dual-copy mechanism, and all the logs are persisted to SSDs.

In average, the log resided in memory is 10 seconds, the short time indicate that, the two replica of data log is enough to ensure the data reliability.

TABLE II: Time (in us) of Data Resided in Memory

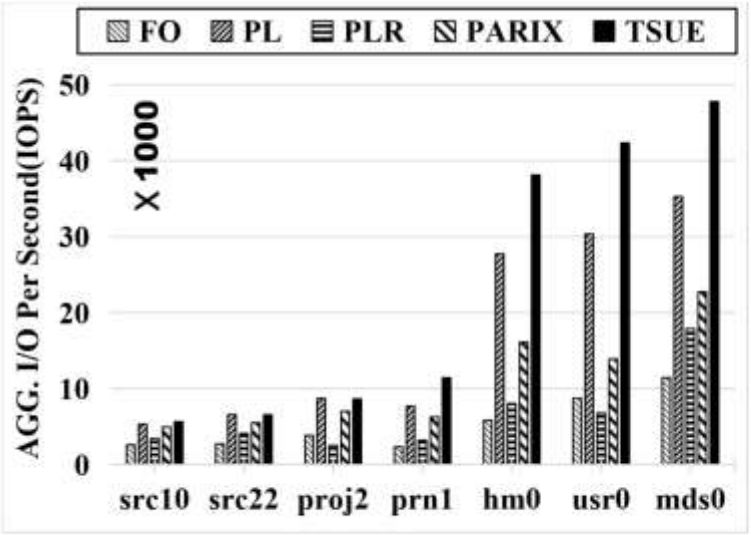
TRACE		DATA_LOG	DELTA_LOG	PARITY_LOG	TOTAL TIME
Ali-Cloud	APPEND	107	75	136	**9151364**
	BUFFER	1561410	6846260	742066	
	RECYCLE	350	526	434	
Ten-Cloud	APPEND	96	185	218	**10979659**
	BUFFER	4388070	59434880	64447	
	RECYCLE	323	237	2363	

• **NOTE:** The Execution is performed under RS(12,4).

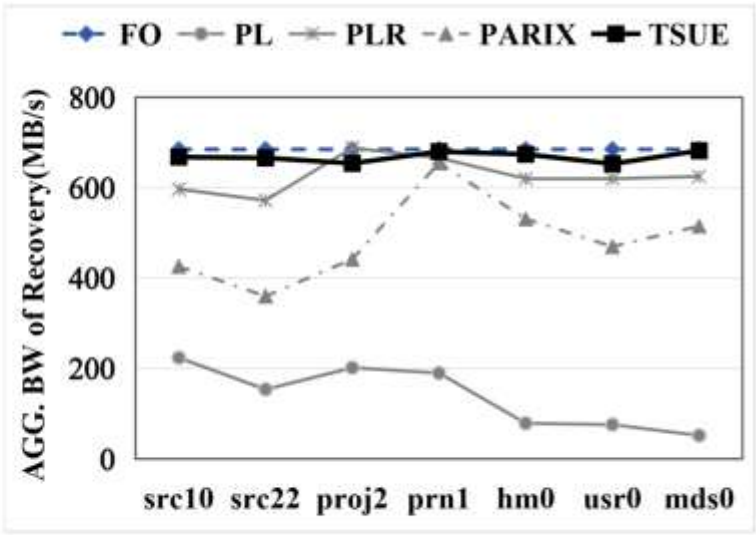
Evaluation

Physical Cluster with HDD(16 nodes)

- 2 intel E2630 CPU, 32GB memory; 2TB SATA HDD
- 56Gb/s Mellanox NIC (but, connected by 40Gb/s ethernet switch)



(a) Update Throughput



(b) Recovery Bandwidth

The performance of TSUE is best in all SOTA methods for HDDs cluster.
Compare with other update mechanism, TSUE has not impact on recovery performance.

Conclusion

TSUE introduces a data log to divide the update process into a two-stage procedure, replacing the time-consuming in-place update of data blocks, thereby achieving lower latency.

TSUE utilizes spatial-temporal locality to reduce the number of random I/O operations through a 3-layer log structure, thereby improving recycling performance.

TSUE designs an adaptive, FIFO-based log pool structure to support high concurrency between log appending and log recycling, which is the optimal structure for leveraging the spatial-temporal locality of requests.

Tests have shown that the TSUE mechanism is applicable not only in SSD environments but also in HDD environments, and is effective in scenarios with a large M value.

Thank you!

Any questions are welcome!

Contact Zheng Wei: weizheng@ncic.ac.cn
Dingwen Tao: taodingwen@ict.ac.cn