HotStorage 2024

Revisiting Erasure Codes: A Configuration Perspective

Runzhou Han[1], **Chao Shi**[1], Tabassum Mahmud[1], Zeren Yang[2], Vladislav Esaulov[3], Lipeng Wan[3], Yong Chen[4], Jim Wayda[4], Matthew Wolf[4], Mai Zheng[1]

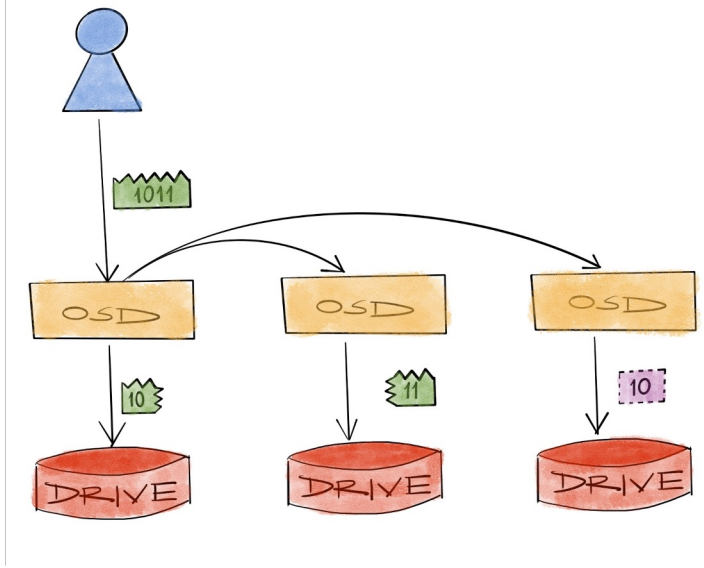
[1]Iowa State University; [2]University of Wisconsin-Madison; [3]Georgia State University; [4]Samsung

IOWA STATE UNIVERSITY

SAMSUNG

Erasure Coding (EC) Is Important

- EC plays a crucial role in modern distributed storage systems (DSS)
  - Used in Ceph, HDFS, DAOS, Azure, Colossus, etc.
  - Ensure fault tolerance with less space overhead compared to traditional replication
    - Tradeoff encoding/decoding computations for space efficiency
  - Increasingly valuable as data volume keeps growing



Great Efforts Have Been Made, But …

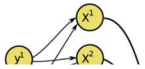
- Reed Solomon (RS) code, Regenerating codes (RGCs), Locally recoverable codes (LRCs), Clay codes, … in both theory and systems communities
  - E.g., Plank@FAST'09, Dimakis@TOIT'10, Gopalan@TOIT'12, Pamies-Juarez@FAST'16, Kolosov@ATC'18, …

IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 56, NO. 9, SEPTEMBER 2010 4539

Network Coding for Distributed Storage Systems

Alexandros G. Dimakis, *Member, IEEE*, P. Brighten Godfrey, *Student Member, IEEE*, Yunnan Wu, *Member, IEEE*, Martin J. Wainwright, *Senior Member, IEEE*, and Kannan Ramchandran, *Fellow, IEEE*

***Abstract*—Distributed storage systems provide reliable access to data through redundancy spread over individually unreliable nodes. Application scenarios include data centers, peer-to-peer storage systems, and storage in wireless networks. Storing data using an erasure code, in fragments spread across nodes, requires less redundancy than simple replication for the same level of reliability. However, since fragments must be periodically replaced as nodes fail, a key question is how to generate encoded fragments in a distributed way while transferring as little data as possible**



IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 58, NO. 11, NOVEMBER 2012 6925

On the Locality of Codeword Symbols

Parikshit Gopalan, Cheng Huang, Huseyin Simitci, and Sergey Yekhanin

***Abstract*—Consider a linear $[n, k, d]_q$ code $\mathcal{C}$. We say that the ith coordinate of $\mathcal{C}$ has locality r, if the value at this coordinate can be recovered from accessing some other r coordinates of $\mathcal{C}$. Data storage applications require codes with small redundancy, low locality for information coordinates, large distance, and low locality for parity coordinates. In this paper, we carry out an in-depth study of the relations between these parameters. We establish a tight**

small value of locality is particularly important for information packets.

These considerations lead us to introduce the concept of an (r, d)-*code*, i.e., a linear code of distance d, where all information symbols have locality at most r. Storage systems based on (r, d)-codes provide fast recovery of information packets from a

A Performance Evaluation and Examination of Open-Source Erasure Coding Libraries For Storage

James S. Plank
University of Tennessee
plank@cs.utk.edu

Jianqiang Luo
Wayne State University

Catherine D. Schuman
University of Tennessee

Lihao Xu
University

Zooko Wilcox-O'Hearn
AllMyData, Inc.

ge installations with high quality tools. As such, there is a need to understand how these codes and implementations perform. In this paper, we compare the encoding and decoding

Great Efforts Have Been Made, But …

- Mostly only measure EC in (largely) simplified setups
  - E.g., simulated DSS, limited EC/DSS parameters and faults, …
- Different from how EC may be configured & used in practice
  - E.g., a Ceph EC pool may be affected by various configurations

Config. Parameters	Options
Ceph storage backend	BlueStore, FileStore
BlueStore cache	meta_ratio, kv_ratio, autotune, etc.
Ceph interface	RADOS, RGW, RBD, CephFS
Num. of PGs in pool	customized, autoscale
EC plugin	Jerasure, ISA, Clay, LRC, SHEC
EC technique	reed_sol_van, cauchy_orig, etc.
EC failure domain	device (OSD), host, rack, etc.
EC device class	HDD, SSD
EC parameter	k, m (equal to n-k), d, stripe_unit, etc.

Table. Highly configurable EC pool in Ceph

- Potential gap b/w theory and practice

Great Efforts Have Been Made, But …

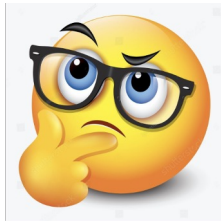
- Mostly only measure EC in (largely) simplified setups
  - E.g., simulated DSS, limited EC/DSS parameters and faults, …
- Different from how EC may be configured & used in practice
  - E.g., a Ceph EC pool may be affected by various configurations

Config. Parameters	Options
Ceph storage backend	BlueStore, FileStore
BlueStore cache	meta_ratio, kv_ratio, autotune, etc.
Ceph interface	RADOS, RGW, RBD, CephFS
Num. of PGs in pool	customized, autoscale
EC plugin	Jerasure, ISA, Clay, LRC, SHEC
EC technique	reed_sol_van, cauchy_orig, etc.
EC failure domain	device (OSD), host, rack, etc.
EC device class	HDD, SSD
EC parameter	k, m (equal to n-k), d, stripe_unit, etc.

Table. Highly configurable EC pool in Ceph

- Potential gap b/w theory and practice

Can we bridge the gap?



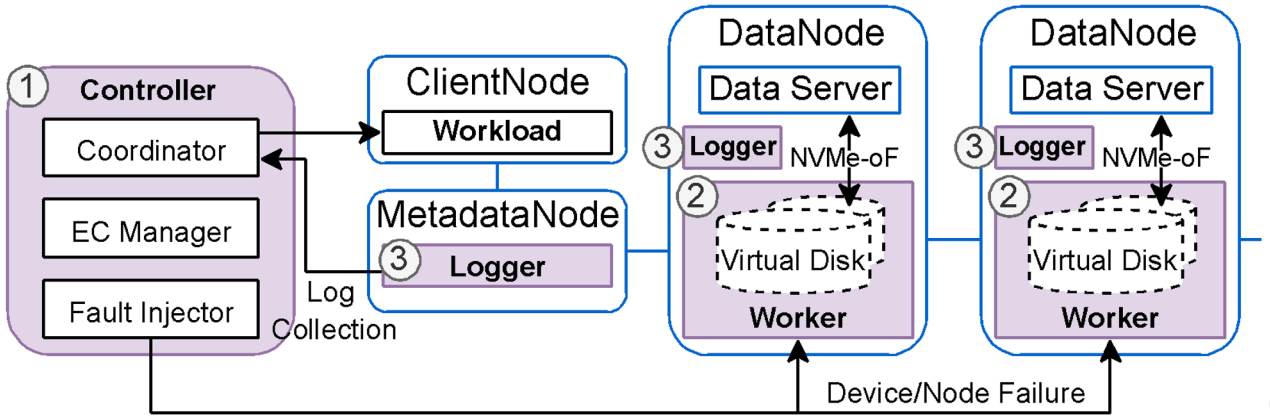
Outline

- ~~Motivation~~
- Methodology
  - How to measure EC in practical systems
- Case Study: EC in Ceph
  - What configurations can affect EC recovery time? To what extent?
  - Is EC recovery time always the bottleneck?
  - What is the impact on write amplification?
- Conclusion and Future Work

" Tech. details coming, feel free to fall asleep"

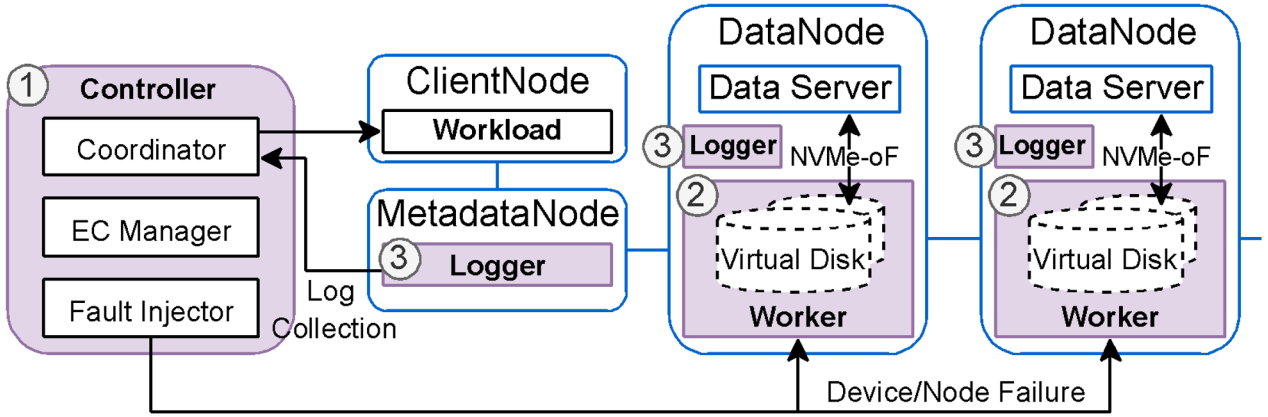
Methodology

- Key Observation: EC is different from regular DSS operations
  - Need controlled faults to trigger EC recovery in DSS systematically
  - Need controlled configurations in multiple dimensions, e.g.:
    - Locality and concurrency of faults
    - Encoding: DSS regular code path
    - Decoding: DSS failure handling code path
    - Workload execution



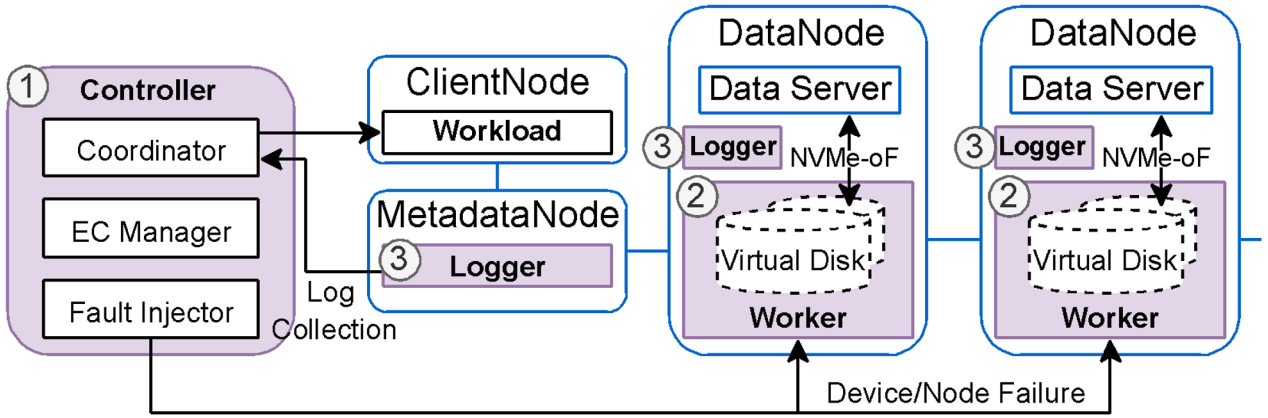
Methodology

- Key Observation: EC is different from regular DSS operations
  - Need controlled faults to trigger EC recovery in DSS systematically
  - Need controlled configurations in multiple dimensions, e.g.:
    - Locality and concurrency of faults
    - Encoding: DSS regular code path
    - Decoding: DSS failure handling code path
    - Workload execution
- ECFault: A framework for measuring EC in practical DSS systematically
  - Three main components
    - Controller
    - Worker
    - Logger



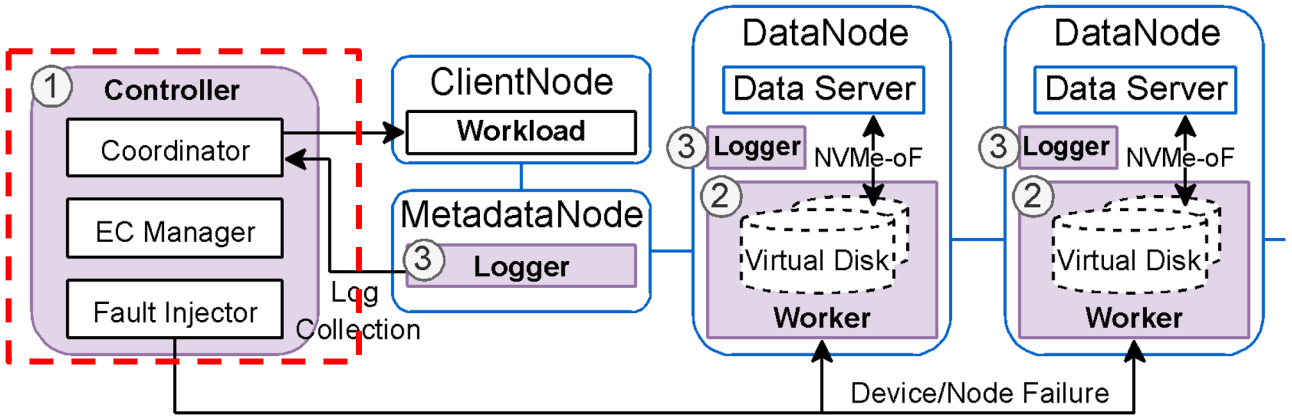
Methodology

- Overview of ECFault components



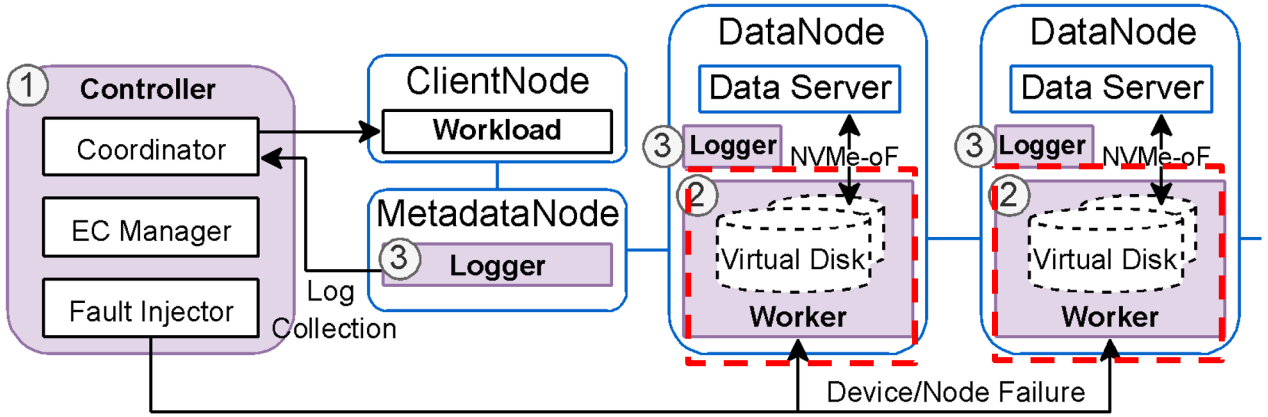
Methodology

- Overview of ECFault components
  - Controller: Control overall configuration & execution of EC exps on target DSS
    - EC Manager: manage all EC-related configurations in a profile
    - Fault Injector: send fault injection requests to DSS nodes based on fault models
    - Coordinator: orchestrate all activities



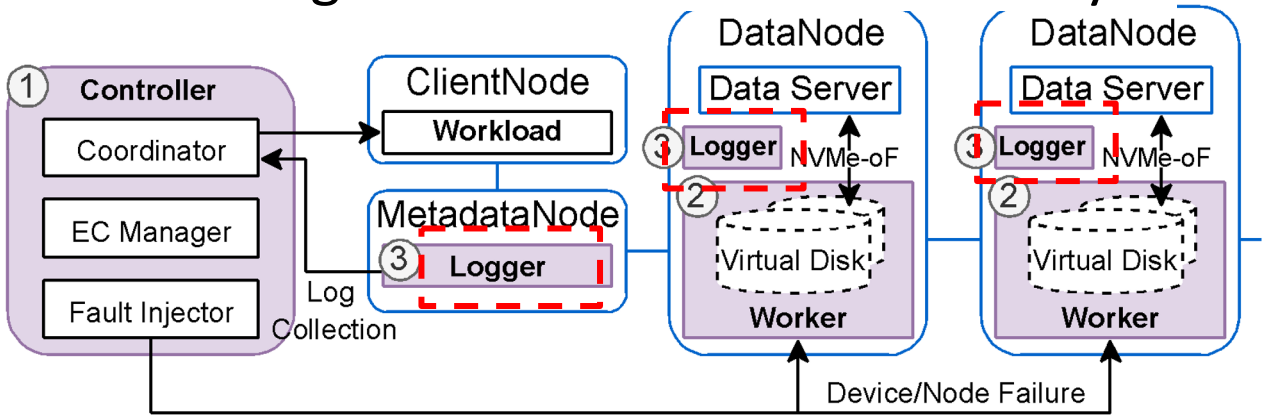
Methodology

- Overview of ECFault components
  - Controller: Control overall configuration & execution of EC exps on target DSS
    - EC Manager: manage all EC-related configurations in a profile
    - Fault Injector: send fault injection requests to DSS nodes based on fault models
    - Coordinator: orchestrate all activities
  - Worker: Manipulate the states of individual nodes
    - Virtual disk provisioning via NVMe-oF to enable easy control of storage states
    - Change DSS states based on Fault Injector's requests



Methodology

- Overview of ECFault components
  - Controller: Control overall configuration & execution of EC exps on target DSS
    - EC Manager: manage all EC-related configurations in a profile
    - Fault Injector: send fault injection requests to DSS nodes based on fault models
    - Coordinator: orchestrate all activities
  - Worker: Manipulate the states of individual nodes
    - Virtual disk provisioning via NVMe-oF to enable easy control of storage states
    - Change DSS states based on Fault Injector's requests
  - Logger: Collect various logs to facilitate fine-grained measurement & analysis
    - General I/O events
    - DSS failure logs
    - EC recovery logs



Outline

- ~~Motivation~~
- ~~Methodology~~
  - ~~How to measure EC in practical systems~~
- Case Study: EC in Ceph
  - What configurations can affect EC recovery time? To what extent?
  - Is EC recovery time always the bottleneck?
  - What is the impact on write amplification?
- Conclusion and Future Work

Case Study: EC in Ceph

- Platform
  - 31-node Ceph cluster, 1 MON/MGR + 30 OSD nodes (AWS EC2 m5.xlarge)
  - 6TB virtual storage capacity to Ceph
- Studied two classic codes w/ various configurations
  - EC: Reed-Solomon (RS) & Clay codes
  - Configuration parameters
    - EC plugins: e.g., RS(12,9), Clay(12,9,11)
    - Ceph: e.g., caching scheme, placement group, stripe unit
    - Fault locality & concurrency: e.g., two/three concurrent OSD failures on the same or different hosts

Case Study: EC in Ceph

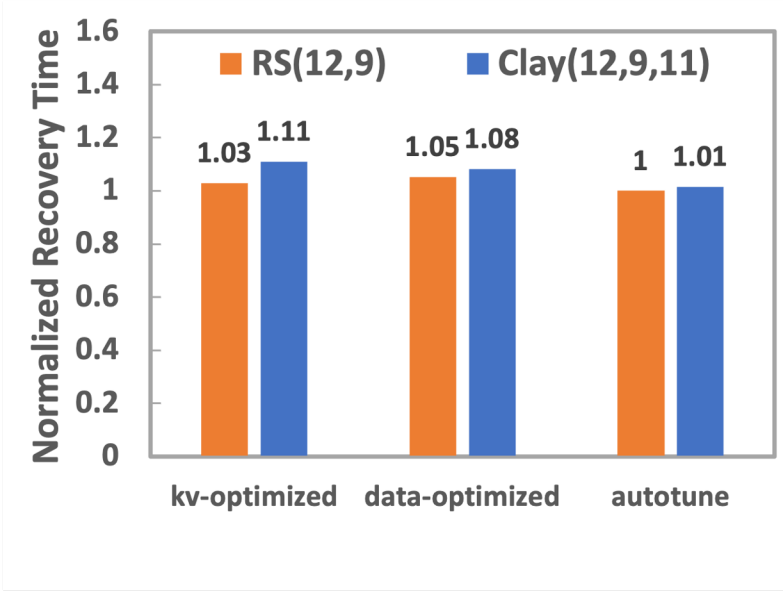
- Result Summary
  - Configurations may affect the EC recovery time significantly (e.g., up to 426%)
  - Theoretically superior codes may actually perform worse under certain configurations
  - There is a system checking period before EC recovery that may account for 41% to 58% of the overall system recovery time
    - EC itself may not necessarily be the bottleneck
  - EC may introduce 32.3% to 72.0% more write amplification (WA) than the theoretical expectation

Case Study: EC in Ceph

- Impact on EC recovery time: Caching Configuration
  - Caching caused up to 11% difference
    - Clay w/ kv-optimized led to worst recovery performance
      - Even worse than traditional RS

ID	Caching Scheme	KV-ratio	Metadata-ratio	Data-ratio
C1	kv-optimized	70%	20%	10%
C2	data-optimized	20%	20%	60%
C3	autotune (init value)	45%	45%	10%

Table. Three Caching Configurations

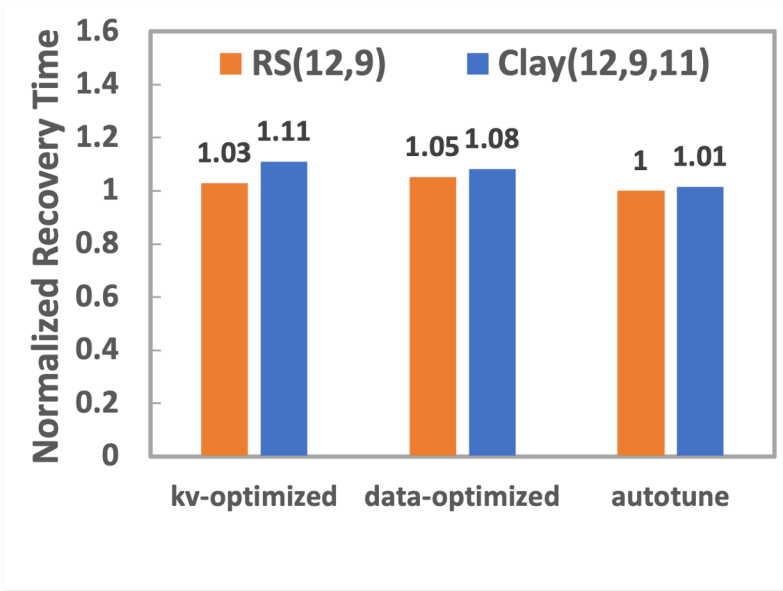


Case Study: EC in Ceph

- Impact on EC recovery time: Caching Configuration
  - Caching caused up to 11% difference
    - Clay w/ kv-optimized led to worst recovery performance
      - Even worse than traditional RS

ID	Caching Scheme	KV-ratio	Metadata-ratio	Data-ratio
C1	kv-optimized	70%	20%	10%
C2	data-optimized	20%	20%	60%
C3	autotune (init value)	45%	45%	10%

Table. Three Caching Configurations

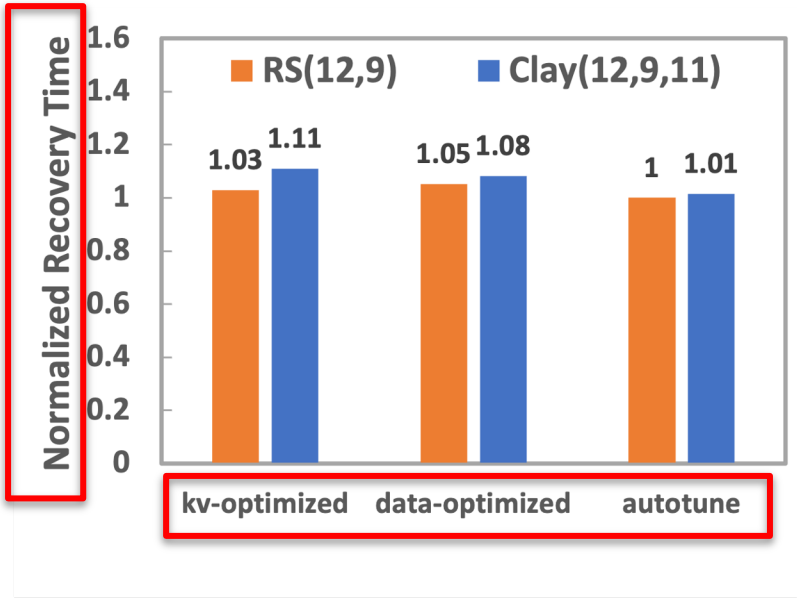


Case Study: EC in Ceph

- Impact on EC recovery time: Caching Configuration
  - Caching caused up to 11% difference
    - Clay w/ kv-optimized led to worst recovery performance
      - Even worse than traditional RS

ID	Caching Scheme	KV-ratio	Metadata-ratio	Data-ratio
C1	kv-optimized	70%	20%	10%
C2	data-optimized	20%	20%	60%
C3	autotune (init value)	45%	45%	10%

Table. Three Caching Configurations

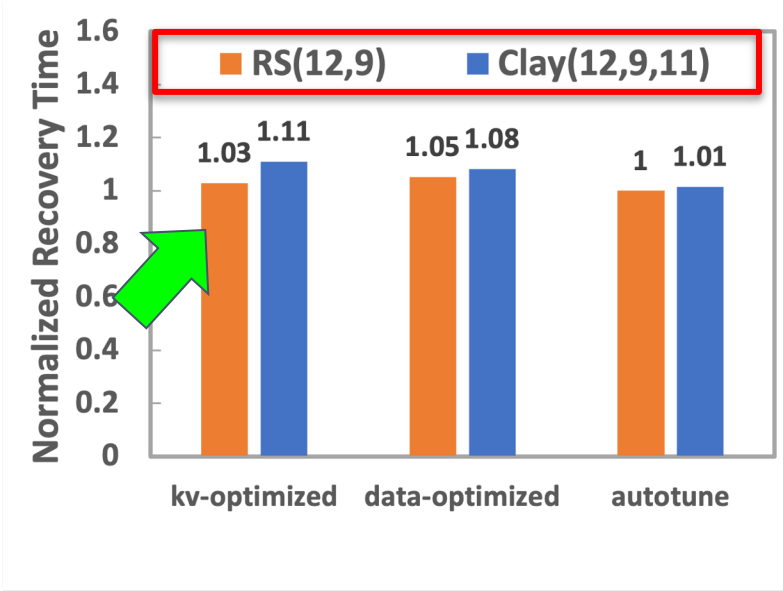


Case Study: EC in Ceph

- Impact on EC recovery time: Caching Configuration
  - Caching caused up to 11% difference
    - Clay w/ kv-optimized led to worst recovery performance
      - Even worse than traditional RS

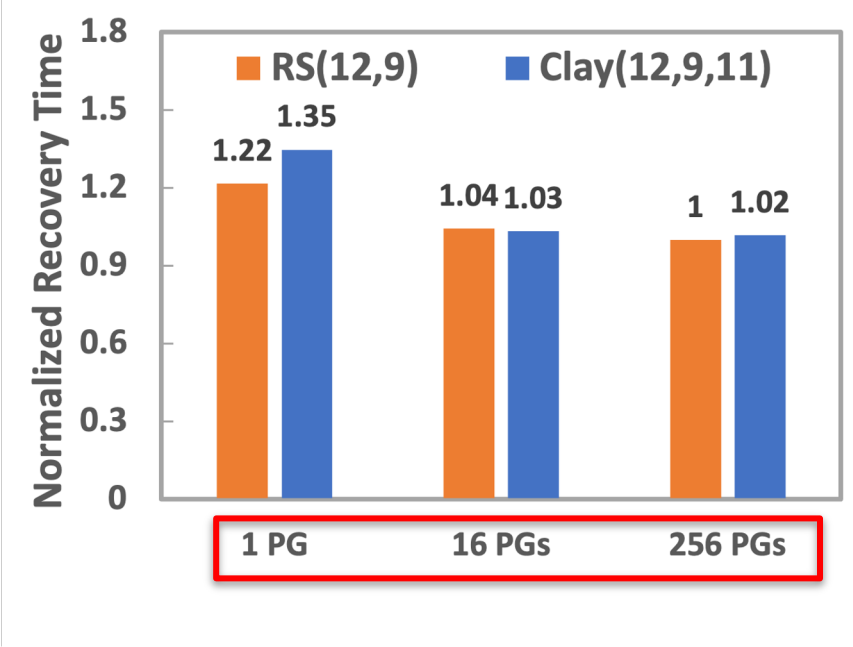
ID	Caching Scheme	KV-ratio	Metadata-ratio	Data-ratio
C1	kv-optimized	70%	20%	10%
C2	data-optimized	20%	20%	60%
C3	autotune (init value)	45%	45%	10%

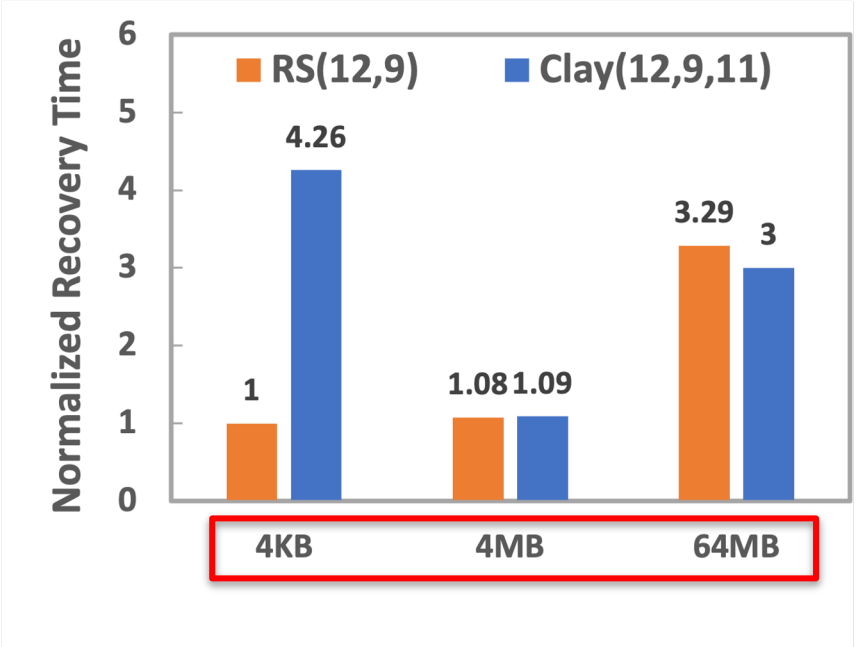
Table. Three Caching Configurations



Case Study: EC in Ceph

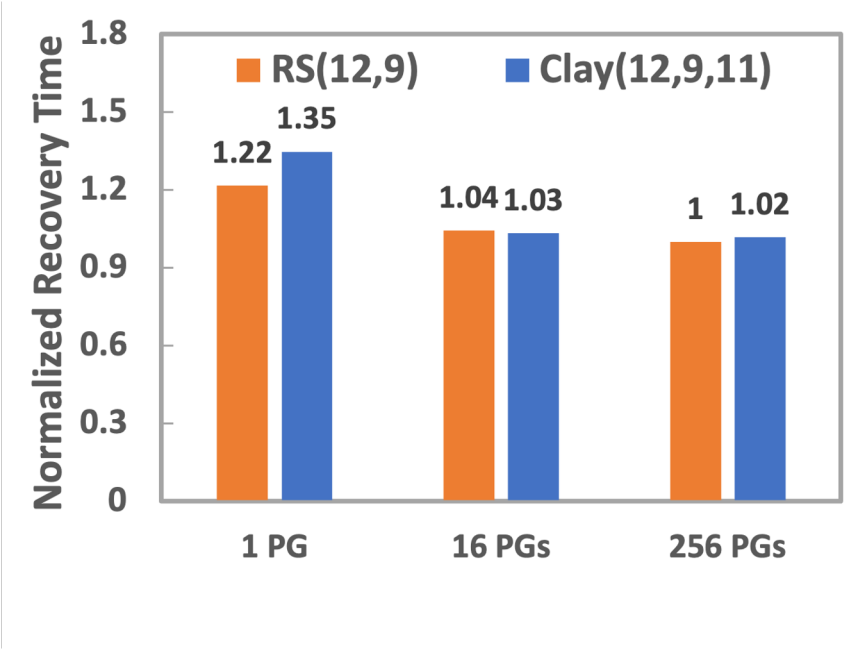
- Impact on EC recovery time: Placement Group (PG) & Stripe Unit (SU)
  - PG configuration caused up to 35% difference
  - SU configuration caused up to 426% difference

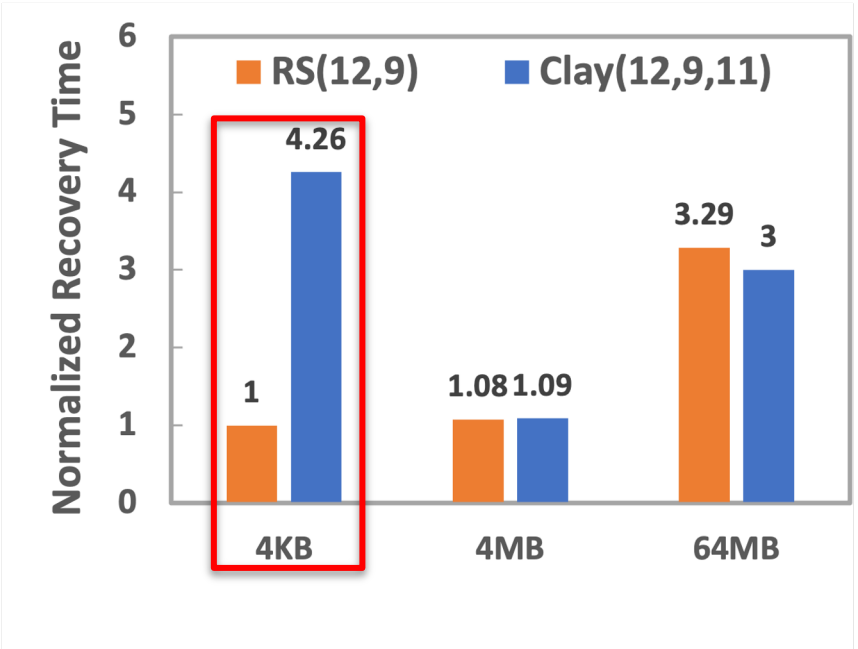




Case Study: EC in Ceph

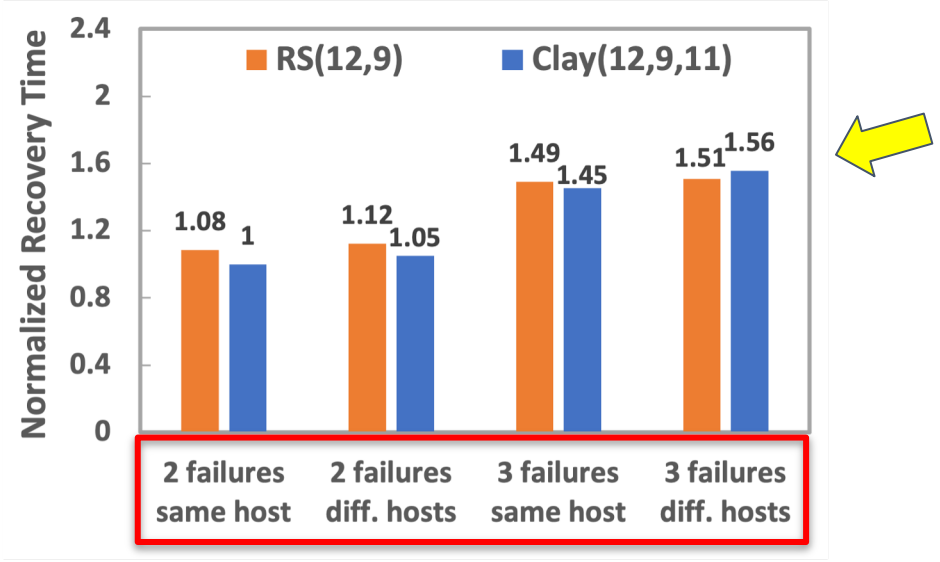
- Impact on EC recovery time: Placement Group (PG) & Stripe Unit (SU)
  - PG configuration caused up to 35% difference
  - SU configuration caused up to 426% difference





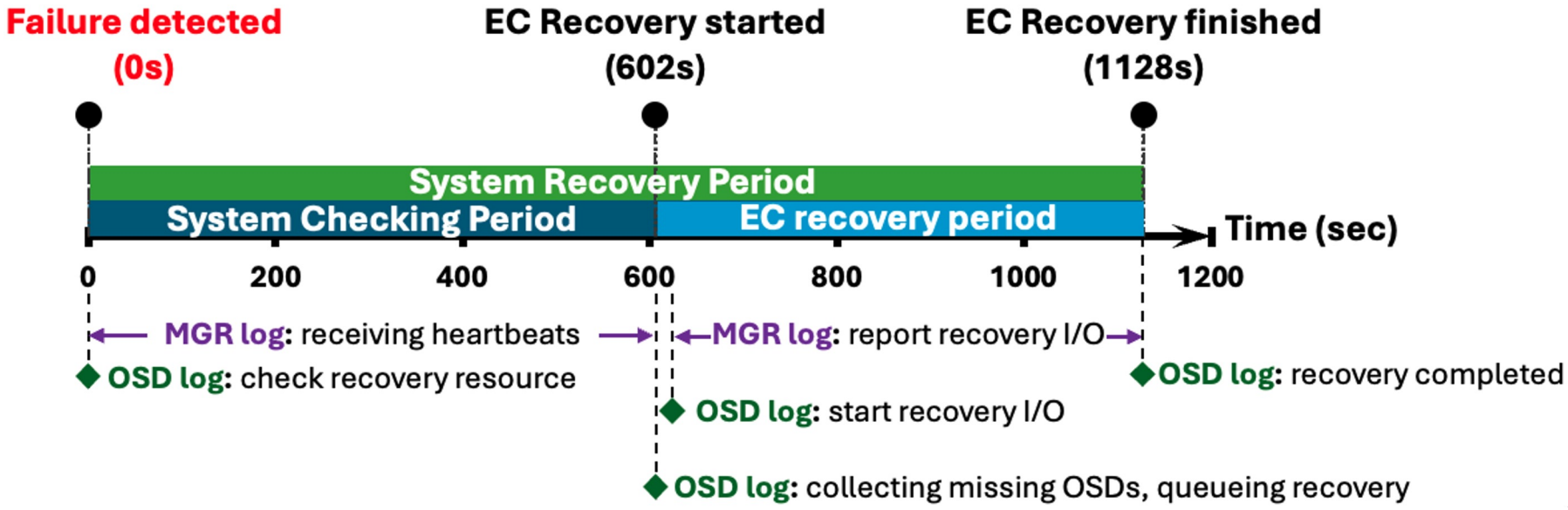
Case Study: EC in Ceph

- Impact on EC recovery time: Fault
  - The locality of three OSD failures may affect the relative performance of RS & Clay
  - The main advantage of Clay over RS (e.g., reduction of repair network traffic) may disappear with only three concurrent failures (on different hosts)



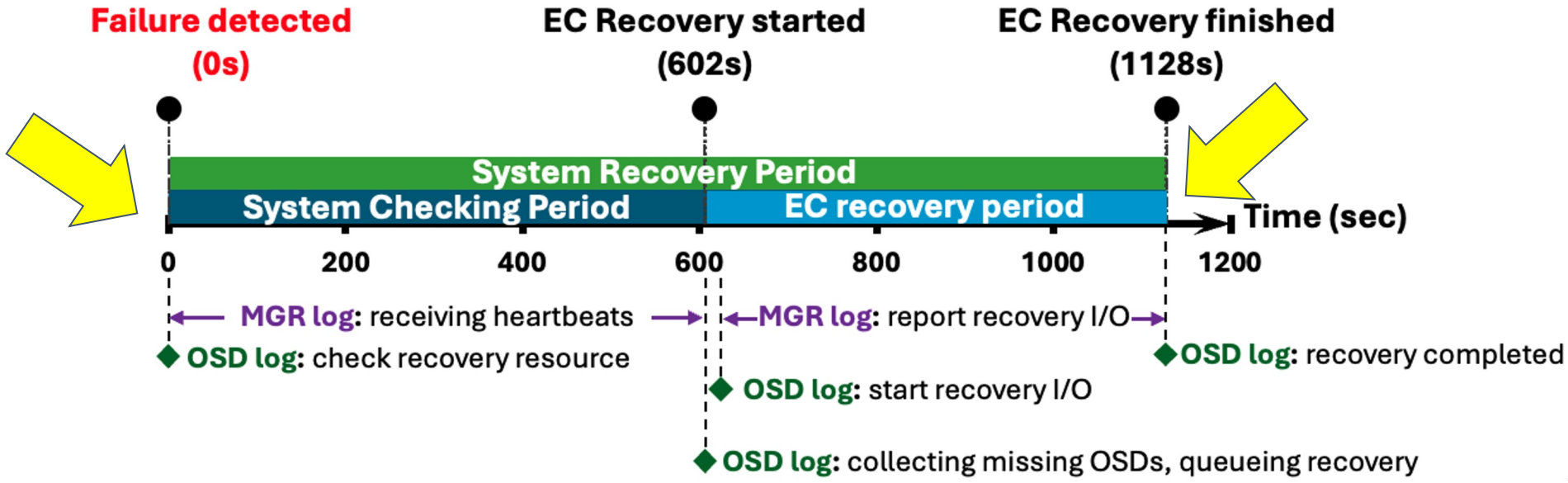
Case Study: EC in Ceph

- Breakdown Analysis of Recovery
  - System Checking Period + EC recovery period
    - System Checking Period accounts for 53.7% of the overall system recovery time



Case Study: EC in Ceph

- Breakdown Analysis of Recovery
  - System Checking Period + EC recovery period
    - System Checking Period accounts for 53.7% of the overall system recovery time



Case Study: EC in Ceph

- Impact on Write Amplification (WA)
  - WA is an undesirable phenomenon that can affect storage capacity, device lifetime, system performance, etc. negatively
  - Theoretical WA: n/k for EC(n, k)
  - Actual WA Factor: actual storage usage divided by the write size of the workload
  - The theoretical WA and the actual WA may change significantly depending on (n,k)
    - E.g., from 32.3% to 72.0%

ID	Code(n,k)	$\frac{n}{k}$	Actual WA Factor	Diff. %
J1	RS(12,9)	1.33	1.76	+32.3%
J2	RS(15,12)	1.25	2.15	+72.0%

Case Study: EC in Ceph

- Impact on Write Amplification (WA)
  - WA is an undesirable phenomenon that can affect storage capacity, device lifetime, system performance, etc. negatively
  - Theoretical WA: n/k for EC(n, k)
  - Actual WA Factor: actual storage usage divided by the write size of the workload
  - The theoretical WA and the actual WA may change significantly depending on (n,k)
    - E.g., from 32.3% to 72.0%

ID	Code(n,k)	$\frac{n}{k}$	Actual WA Factor	Diff. %
J1	RS(12,9)	1.33	1.76	+32.3%
J2	RS(15,12)	1.25	2.15	+72.0%

Case Study: EC in Ceph

- Impact on Write Amplification (WA)
  - WA is an undesirable phenomenon that can affect storage capacity, device lifetime, system performance, etc. negatively
  - Theoretical WA: n/k for EC(n, k)
  - Actual WA Factor: actual storage usage divided by the write size of the workload
  - The theoretical WA and the actual WA may change significantly depending on (n,k)
    - E.g., from 32.3% to 72.0%

ID	Code(n,k)	$\frac{n}{k}$	Actual WA Factor	Diff. %
J1	RS(12,9)	1.33	1.76	+32.3%
J2	RS(15,12)	1.25	2.15	+72.0%

Case Study: EC in Ceph

- Impact on Write Amplification (WA)
  - WA is an undesirable phenomenon that can affect storage capacity, device lifetime, system performance, etc. negatively
  - Theoretical WA: n/k for EC(n, k)
  - Actual WA Factor: actual storage usage divided by the write size of the workload
  - The theoretical WA and the actual WA may change significantly depending on (n,k)
    - E.g., from 32.3% to 72.0%

ID	Code(n,k)	$\frac{n}{k}$	Actual WA Factor	Diff. %
J1	RS(12,9)	1.33	1.76	+32.3%
J2	RS(15,12)	1.25	2.15	+72.0%

Outline

- ~~Motivation~~
- ~~Methodology~~
  - ~~How to measure EC in practical systems~~
- ~~Case Study: EC in Ceph~~
  - ~~What configurations can affect EC recovery time? To what extent?~~
  - ~~Is EC recovery time always the bottleneck?~~
  - ~~What is the impact on write amplification?~~
- Conclusion and Future Work

Conclusion & Future Work

- Configurations can affect EC in practical DSS significantly
  - E.g., up to 426% in recovery time
  - Theoretical advantage of codes may change depending on configurations
- Next Steps
  - More rigorous measurement with practical configurations
    - E.g., EC plugins, DSS, faults, workloads, etc
  - Configuration-aware optimizations for EC and EC-based DSS
  - Open Challenge: how to handle the (almost) infinite configuration space? AI/ML?

Revisiting Erasure Codes: A Configuration Perspective

Q & A

"Wake up! Coffee time!"

IOWA STATE UNIVERSITY

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

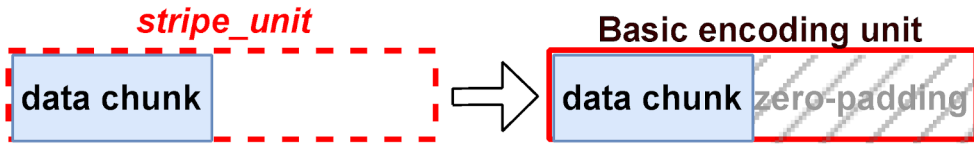
Georgia State University

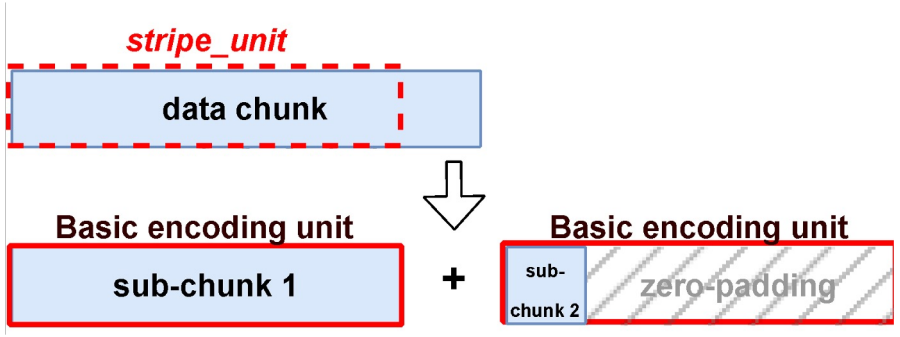
SAMSUNG

Backup

Write Amplification related issue

- Write amplification issue of EC in Ceph
  - Given a fixed encoding unit size (***stripe_unit***) and fault tolerance capacity (***n-k***), we find codes with larger ***n*** always have higher actual storage overhead
  - Partially caused by division-and-padding
    - Chunks smaller than ***stripe_unit*** (undersized) will be padded to a standard encoding unit
    - Chunks larger than ***stripe_unit*** (oversized) will be divided into smaller pieces first, and these pieces are then padded to standard encoding units





What is the impact on write amplification?

- Write amplification issue of EC in Ceph
  - Given a fixed encoding unit size (***stripe_unit***) and fault tolerance capacity (***n-k***), we find codes with larger ***n*** always have higher actual storage overhead
  - Partially caused by division-and-padding feature in Ceph
    - To justify this issue, we design a farsightedness formula to show the lower bound of actual WA.