Piggybacked Erasure Codes for Distributed Storage &

Findings from the Facebook Warehouse Cluster

Berkeley
University of California

facebook

K. V. Rashmi, Nihar Shah, D. Gu,
H. Kuang, D. Borthakur, K. Ramchandran

Piggybacked Erasure Codes for Distributed Storage &

Findings from the Facebook Warehouse Cluster

Berkeley
University of California

facebook

K. V. Rashmi, Nihar Shah, D. Gu,
H. Kuang, D. Borthakur, K. Ramchandran

Presented by Kangwook Lee

=> Unhandled questions will be happily forwarded

Outline

- Introduction & Motivation
  - Measurements from Facebook’s Warehouse cluster

- The Piggybacking framework

- Via the Piggybacking framework
  - Best known codes for several settings
  - Comparison with other codes
  - Preliminary practical experiments

- Summary & future work

Outline

- Introduction & Motivation
  - Measurements from Facebook's Warehouse cluster

- The Piggybacking framework

- Via the Piggybacking framework
  - Best known codes for several settings
  - Comparison with other codes
  - Preliminary practical experiments

- Summary & future work

Motivation: Facebook's Warehouse Cluster Measurements

- Multiple tens of PBs and growing
- Multiple thousands of nodes

Reducing storage requirements is of high importance

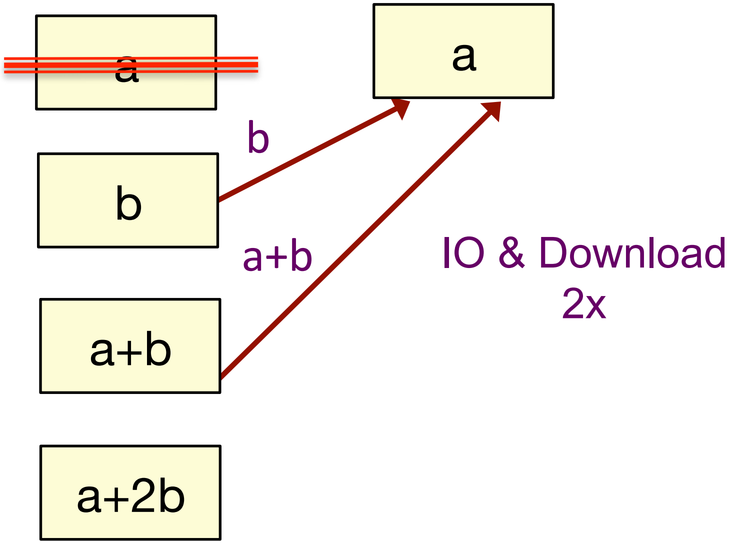
- Uses (14, 10) RS code for storage efficiency
  - on less-frequently accessed data
- Multiple PBs of RS coded data

[Rashmi et al., USENIX HotStorage 2013]
"A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the Facebook warehouse cluster"

Repair in Conventional Erasure Codes: High disk IO & download

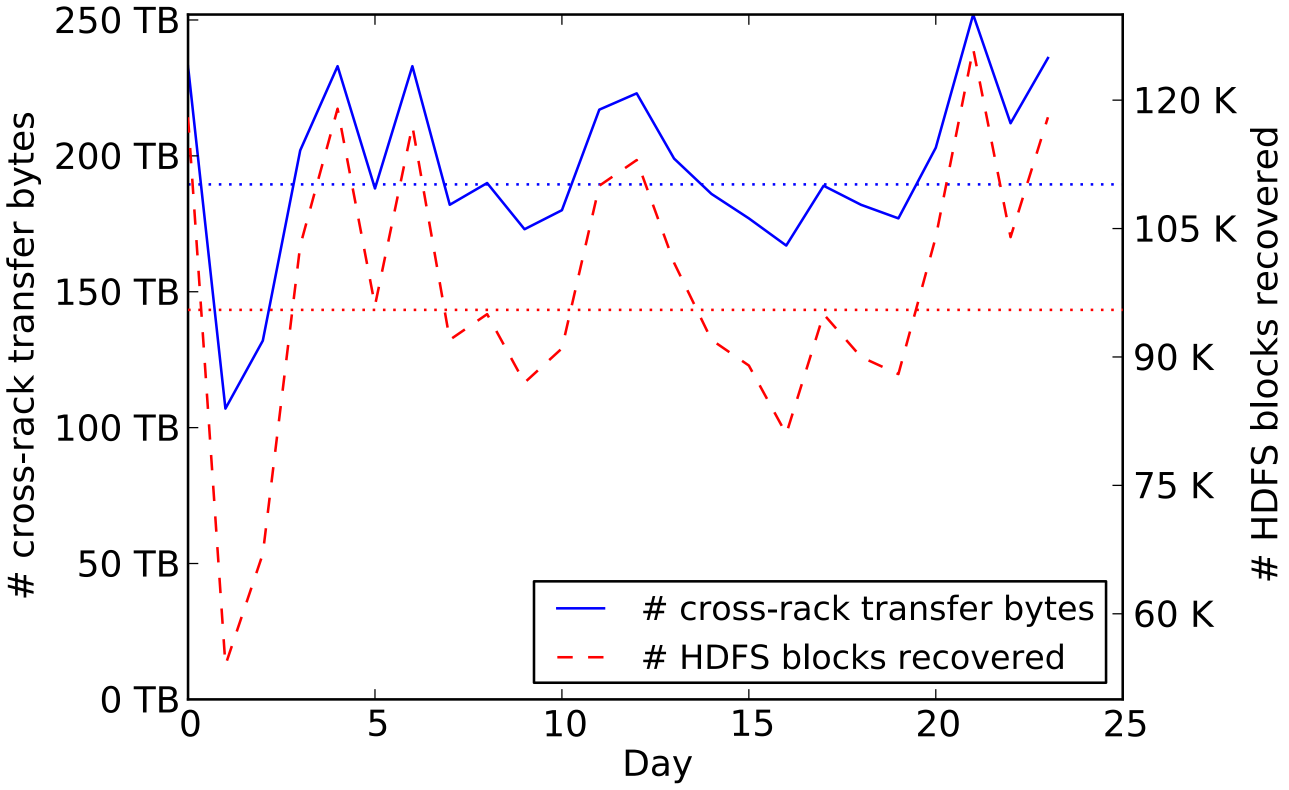
- Conventional repair in erasure codes:

  Download & IO = Size of entire message



For (14, 10) RS code, it's 10x!

Amount of transfer



- Median of 180 TB transferred across racks per day for repair

Rashmi et al., USENIX HotStorage 2013

Breakdown of repairs

# repairs	% of repairs
1	98.08
2	1.87
3	0.036
4	9×10^{-6}
≥ 5	9×10^{-9}

⇒ **Code should perform efficient single repair.**

Rashmi et al., USENIX HotStorage 2013

Outline

- Introduction & Motivation
  - Measurements from Facebook’s Warehouse cluster

- The Piggybacking framework

- Via the Piggybacking framework
  - Best known codes for several settings
  - Comparison with other codes
  - Preliminary practical experiments

- Summary & future work

Piggybacking RS codes: Toy Example

Step 1: Take 2 stripes of (4, 2) Reed-Solomon code

systematic 1	a_1	b_1
systematic 2	a_2	b_2
parity 1	a_1+a_2	b_1+b_2
parity 2	a_1+2a_2	b_1+2b_2

Piggybacking RS codes: Toy Example

Step 2: Add ‘piggybacks’

a_1	b_1
a_2	b_2
a_1+a_2	b_1+b_2
a_1+2a_2	$b_1+2b_2+a_1$

No additional storage!

Fault-Tolerance

Same fault tolerance as RS code:

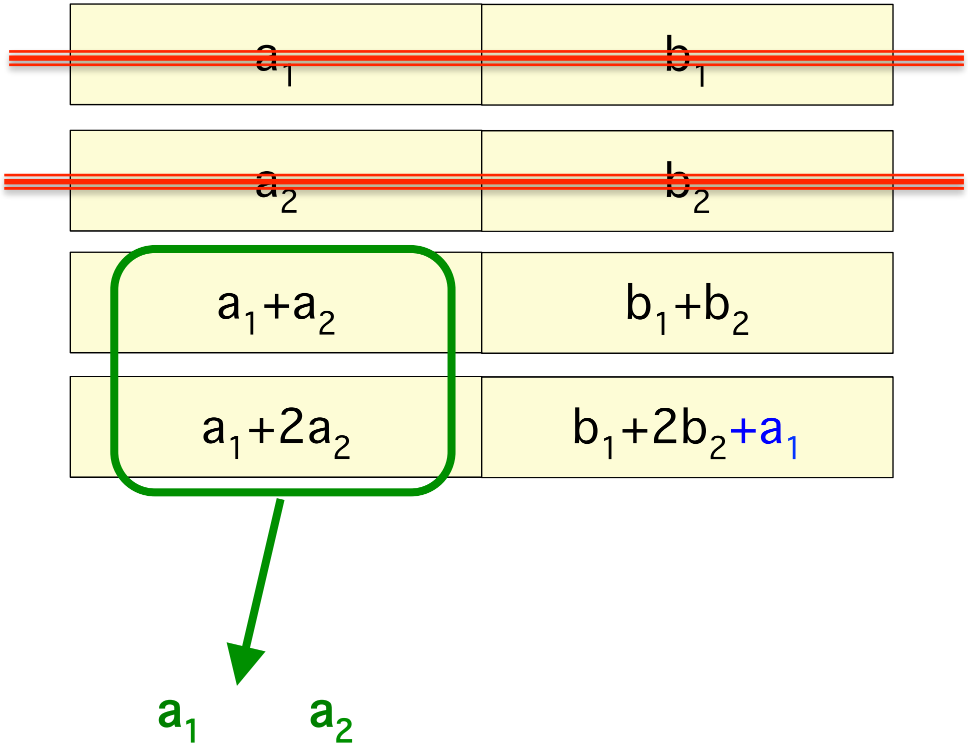
can tolerate any 2 failures

~~a_1~~	~~b_1~~
~~a_2~~	~~b_2~~
a_1+a_2	b_1+b_2
a_1+2a_2	$b_1+2b_2+a_1$

Fault-Tolerance

Same fault tolerance as RS code:

can tolerate any 2 failures



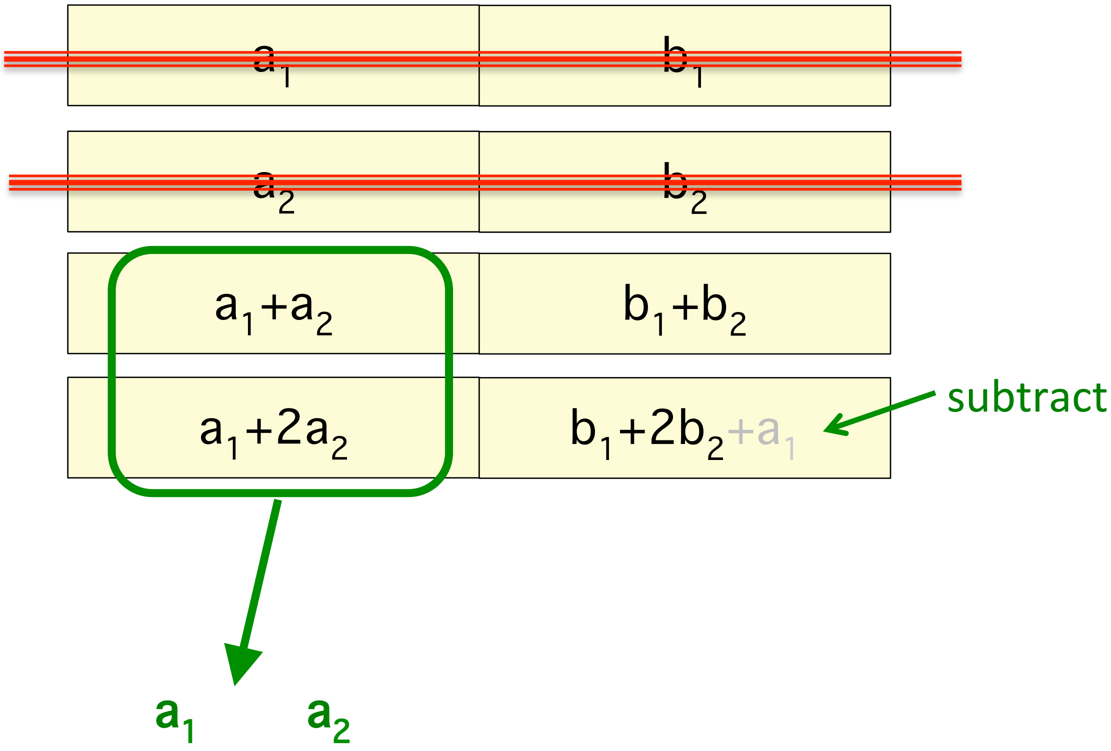
~~a_1~~	~~b_1~~
~~a_2~~	~~b_2~~
a_1+a_2	b_1+b_2
a_1+2a_2	$b_1+2b_2+a_1$

a_1 a_2

Fault-Tolerance

Same fault tolerance as RS code:

can tolerate any 2 failures



~~a_1~~	~~b_1~~
~~a_2~~	~~b_2~~
a_1+a_2	b_1+b_2
a_1+2a_2	$b_1+2b_2+a_1$

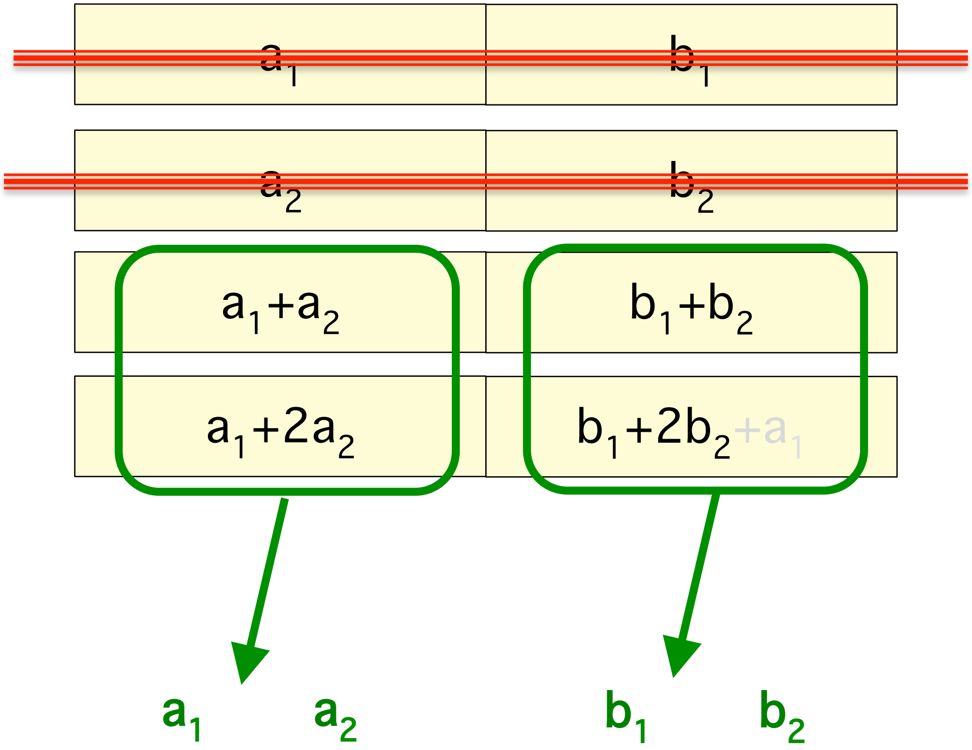
subtract

a_1 a_2

Fault-Tolerance

Same fault tolerance as RS code:

can tolerate any 2 failures



~~a_1~~ ~~b_1~~

~~a_2~~ ~~b_2~~

a_1+a_2 b_1+b_2

a_1+2a_2 $b_1+2b_2+a_1$

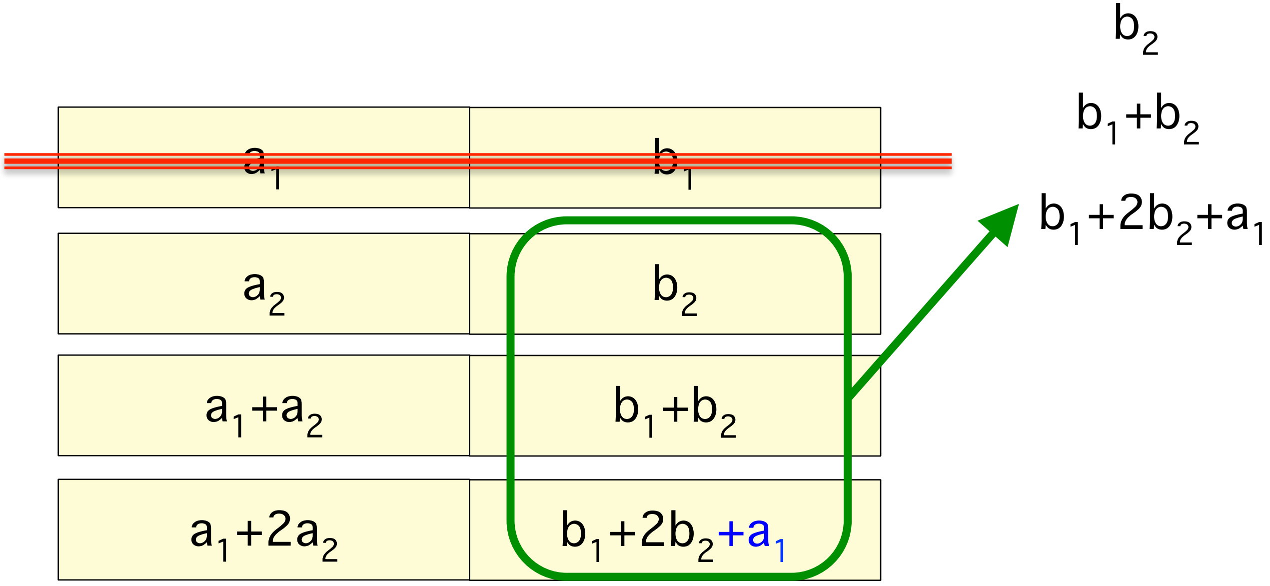
a_1 a_2 b_1 b_2

Repair

~~a_1~~	~~b_1~~
a_2	b_2
a_1+a_2	b_1+b_2
a_1+2a_2	$b_1+2b_2+a_1$

Repair

IO & Download = 3
(instead of 4 as in RS)



~~a_1~~	~~b_1~~
a_2	b_2
a_1+a_2	b_1+b_2
a_1+2a_2	$b_1+2b_2+a_1$

b_2

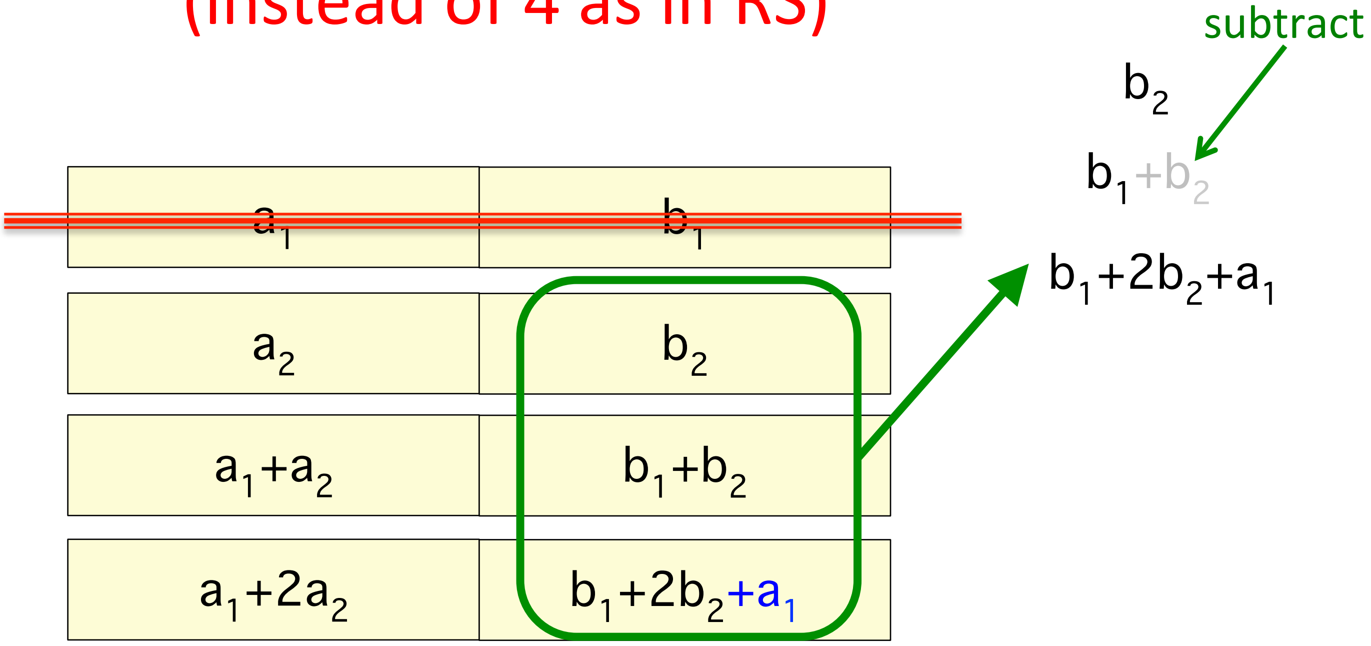
b_1+b_2

$b_1+2b_2+a_1$

Optimal !

Repair

IO & Download = 3
(instead of 4 as in RS)



~~a_1~~	~~b_1~~
a_2	b_2
a_1+a_2	b_1+b_2
a_1+2a_2	$b_1+2b_2+a_1$

subtract

b_2

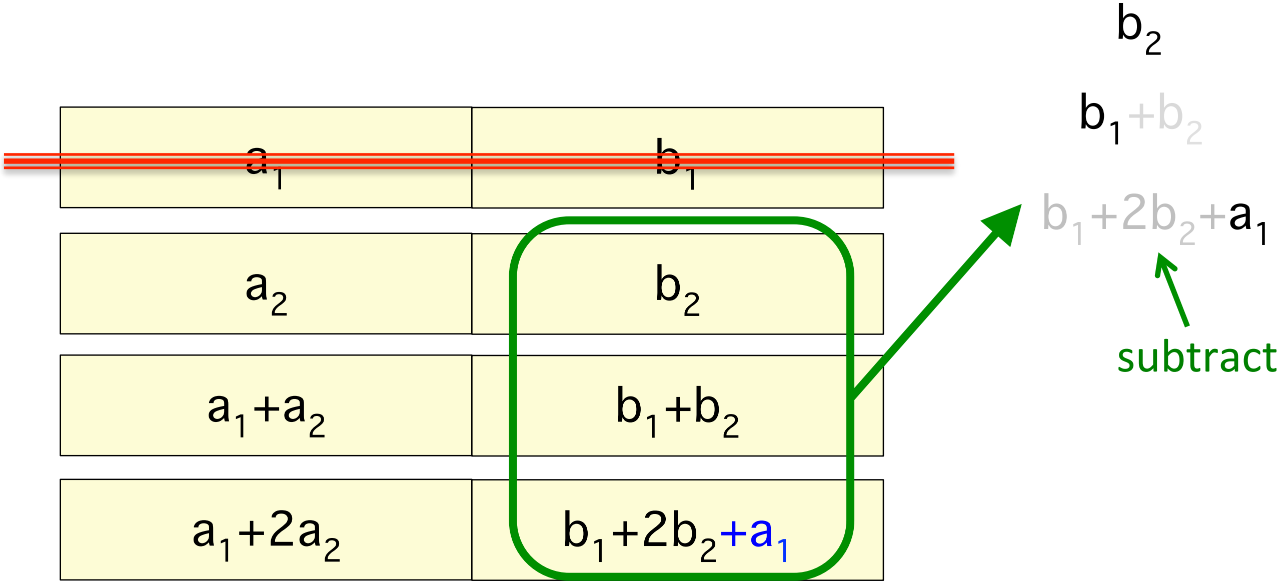
b_1+b_2

$b_1+2b_2+a_1$

Optimal !

Repair

IO & Download = 3
(instead of 4 as in RS)



~~a_1~~	~~b_1~~
a_2	b_2
a_1+a_2	b_1+b_2
a_1+2a_2	$b_1+2b_2+a_1$

b_2

b_1+b_2

$b_1+2b_2+a_1$

subtract

Optimal !

General Piggybacking Framework

Step 1: Take 2 (or more) stripes of (n, k) code C

a_1	b_1
⋮	⋮
a_k	b_k
$f_1(a_1,\ldots,a_k)$	$f_1(b_1,\ldots,b_k)$
⋮	⋮
$f_{n-k}(a_1,\ldots,a_k)$	$f_{n-k}(b_1,\ldots,b_k)$

General Piggybacking Framework

Step 2: Add `Piggybacks’

a_1	b_1
$\vdots$	$\vdots$
a_k	b_k
$f_1(a_1,\ldots,a_k)$	$f_1(b_1,\ldots,b_k) + p_1(a_1,\ldots,a_k)$
$\vdots$	$\vdots$
$f_{n-k}(a_1,\ldots,a_k)$	$f_{n-k}(b_1,\ldots,b_k) + p_{n-k}(a_1,\ldots,a_k)$

General Piggybacking Framework

Decoding: use decoder of C

a_1	b_1
$\vdots$	$\vdots$
a_k	b_k
$f_1(a_1,\dots,a_k)$	$f_1(b_1,\dots,b_k) + p_1(a_1,\dots,a_k)$
$\vdots$	$\vdots$
$f_{n-k}(a_1,\dots,a_k)$	$f_{n-k}(b_1,\dots,b_k) + p_{n-k}(a_1,\dots,a_k)$

recover $a_1,\dots,a_k$ as in C

subtract piggybacks;
recover $b_1,\dots,b_k$ as in C

General Piggybacking Framework

- Piggybacking does not reduce minimum distance

- ∴ Can choose arbitrary functions for piggybacking

> Piggybacking functions should be designed such that they can be used for repair

- 3 designs of Piggybacking functions in ISIT paper

Outline

- Introduction & Motivation
  - Measurements from Facebook's Warehouse cluster
- The Piggybacking framework
- Via the Piggybacking framework
  - Best known codes for several settings
  - Comparison with other codes
  - Preliminary practical experiments
- Summary & future work

Via the Piggybacking framework…

① “Practical” High-rate MDS codes:

Lowest known IO & download during repair

- Storage constrained systems: MDS & high-rate
- Then, why not high-rate Minimum Storage Regenerating (MSR) codes ?
  - Require exponential block length *(Tamo et al. 2011)*

n	k	Block length			IO & Download (% of message size)		
		RS	Piggy-RS	MSR	RS	Piggy-RS	MSR
16	14	1	4	128	100	77	54
25	22	1	4	3154	100	69	36
210	200	1	4	10^{20}	100	56	11

Comparison With Other Codes

Code	MDS	Parameters	Block length (in k)
High-rate MSR	Y	all	exponential
Product-matrix MSR etc.	Y	low rate	linear
Rotated-RS	Y	≤ 3 parities	constant
EVENODD/RDP	Y	≤ 2 parities	linear
MBR	N	all	linear
Local repair	N	all	constant
Piggyback	**Y**	**all**	**constant / linear**

- These are the only other codes that satisfy our requirements of being MDS, high rate and have small block lengths.
- Piggyback codes have smaller repair download and IO than them both.

Via the Piggybacking framework…

② Binary MDS (vector) codes

- lowest known IO & download during repair (when #parity>2)
- for all parameters where binary MDS (vector) codes exist

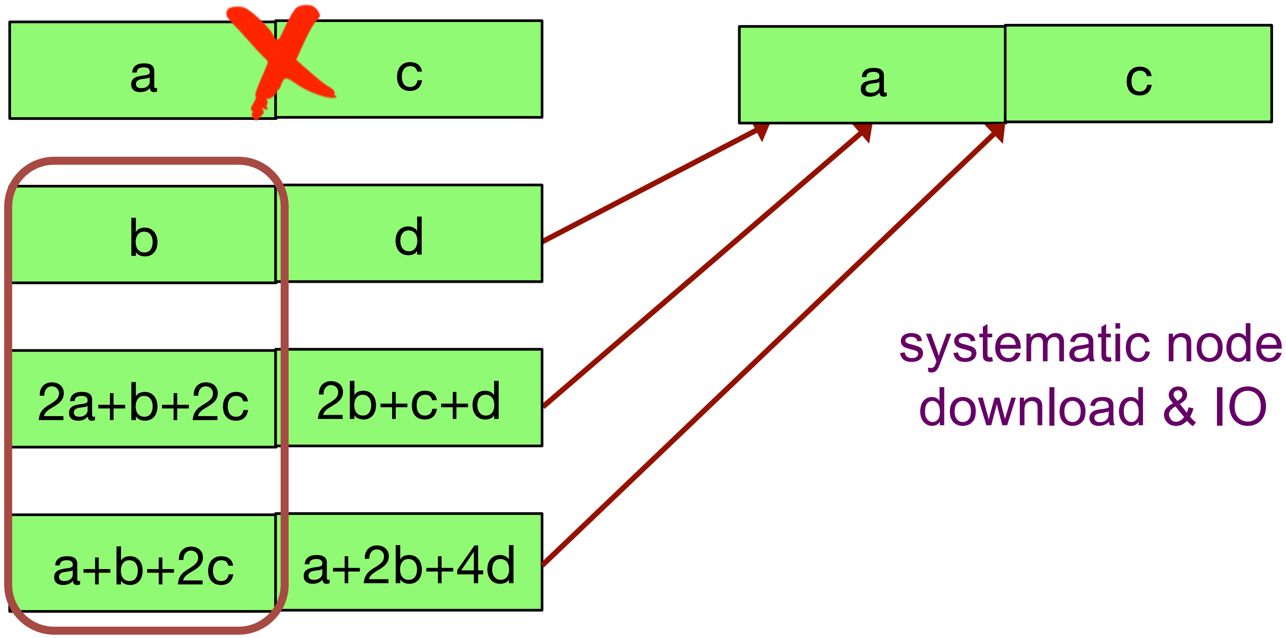
Via the Piggybacking framework…

③ Enabling parity repair in regenerating codes designed for only systematic repair

  - efficiency in systematic repair retained
  - parity repair improved

Example…

- Regenerating code that repairs systematic nodes efficiently
- Parity node repair performed by downloading all data



systematic node repair:
download & IO = 1.5x

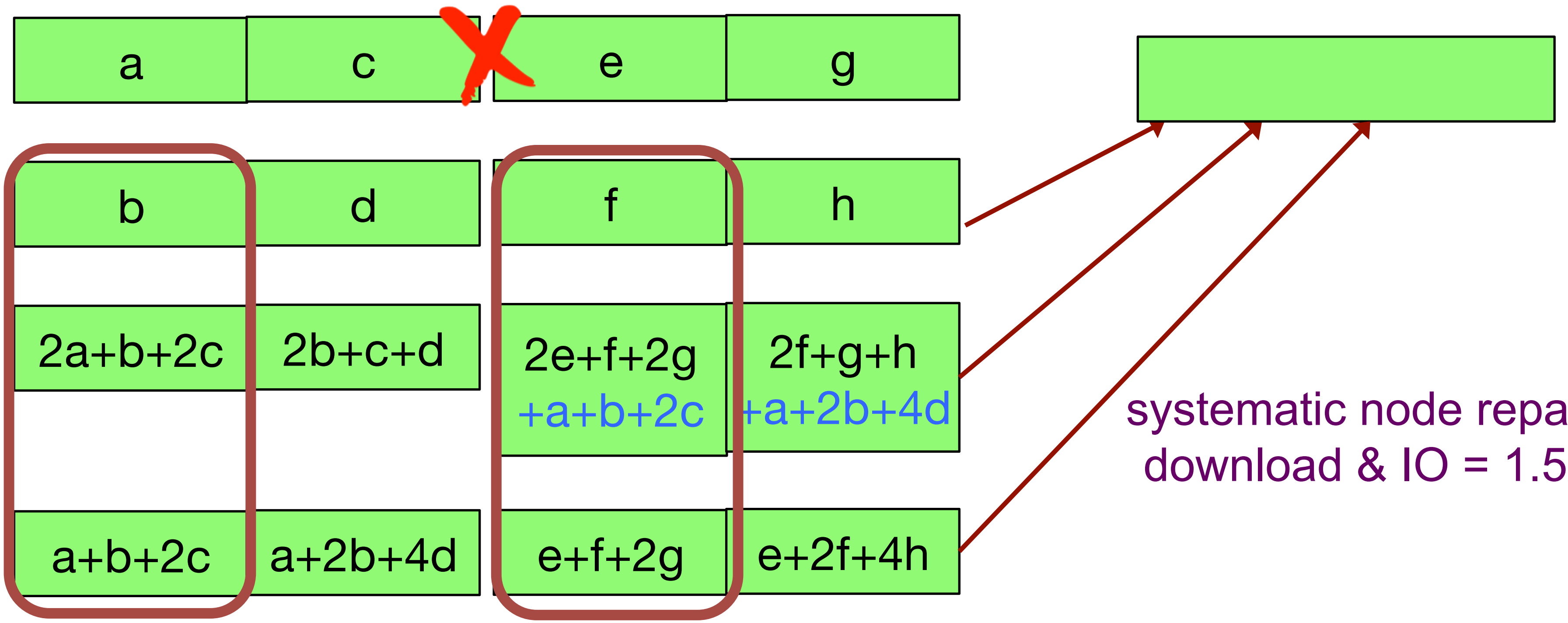
- Take two stripes of this code

a	c	e	g
b	d	f	h
2a+b+2c	2b+c+d	2e+f+2g	2f+g+h
a+b+2c	a+2b+4d	e+f+2g	e+2f+4h

- Add Piggybacks of parities from 1st stripe to 2nd stripe

a	c	e	g
b	d	f	h
2a+b+2c	2b+c+d	2e+f+2g +a+b+2c	2f+g+h +a+2b+4d
a+b+2c	a+2b+4d	e+f+2g	e+2f+4h

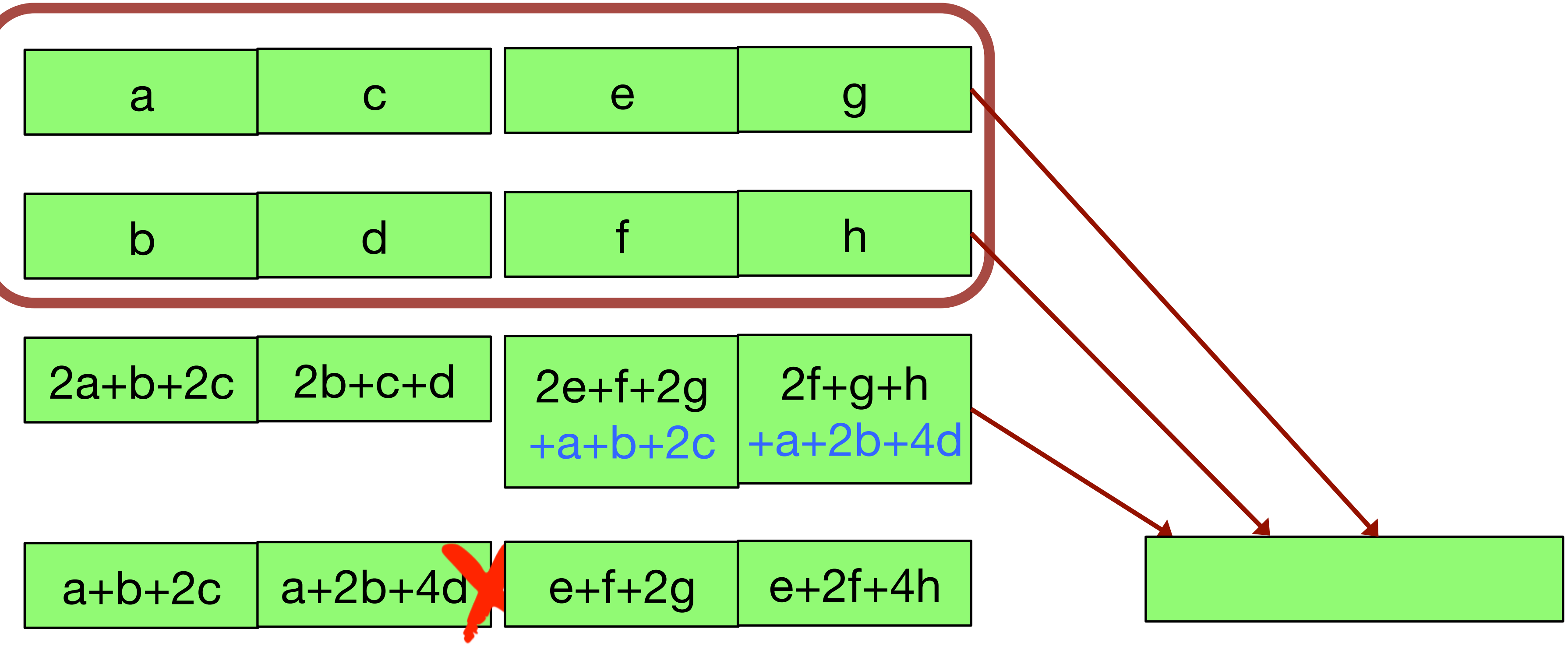
- Systematic repair: same efficiency as original code (Piggyback subtracted out in downloaded data)



a	c	e	g
b	d	f	h
2a+b+2c	2b+c+d	2e+f+2g +a+b+2c	2f+g+h +a+2b+4d
a+b+2c	a+2b+4d	e+f+2g	e+2f+4h

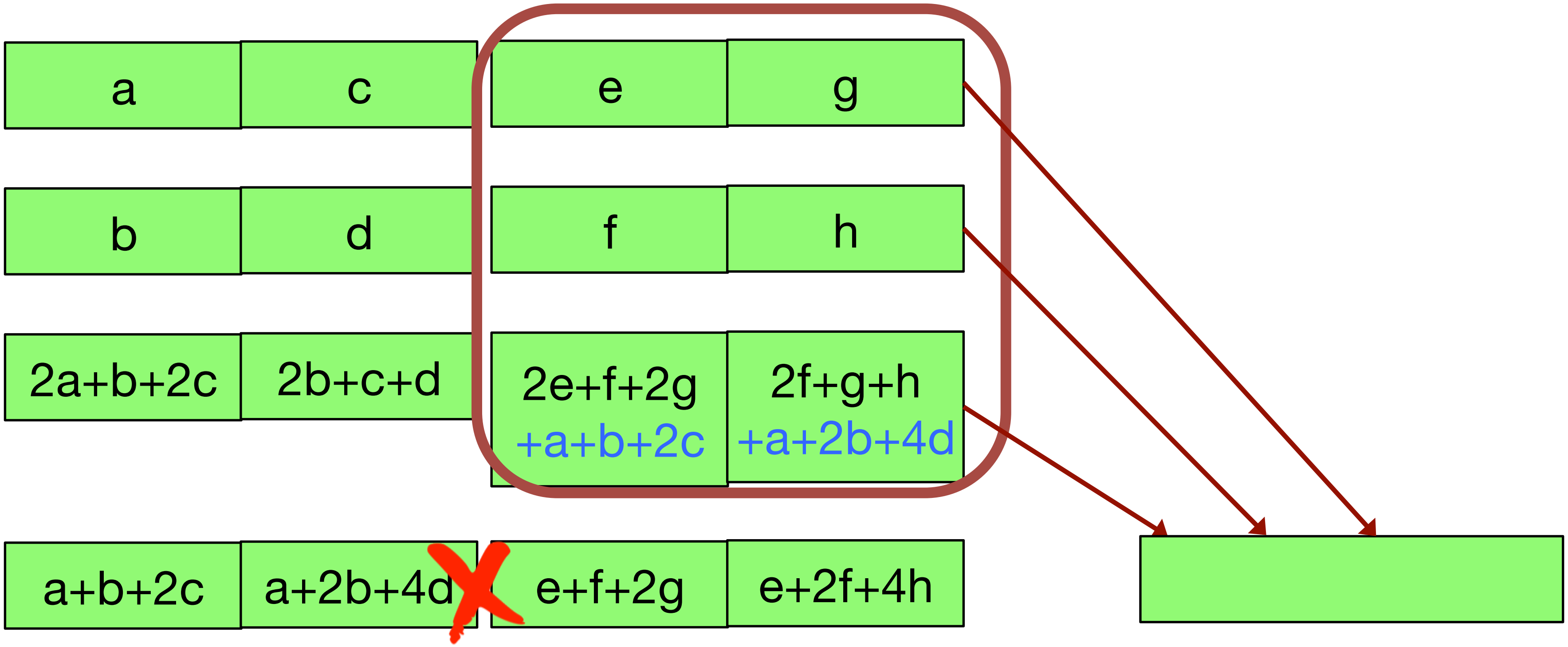
systematic node repair:
download & IO = 1.5x

- Original code:
  Parity repair download & IO = 2x



a	c	e	g
b	d	f	h
2a+b+2c	2b+c+d	2e+f+2g +a+b+2c	2f+g+h +a+2b+4d
a+b+2c	a+2b+4d	e+f+2g	e+2f+4h

- Using Piggybacks:
  Second parity repair download & IO = 1.5x



Via the Piggybacking framework…

④ Have implemented (14, 10) Piggyback-RS in the Hadoop Distributed File System (HDFS)

- 35% reduction in disk-IO and download
- same storage & fault tolerance
- testing on Facebook’s Warehouse cluster underway

Is connecting to more nodes a concern ?

We performed measurements for various data-sizes in the Facebook Warehouse cluster in production.

Piggyback-RS codes:

- Reduce primary metrics of IO & download
- Time to repair also reduces upon connecting to more

Locality/Connectivity not an issue in this setting

Outline

- Introduction & Motivation
  - Measurements from Facebook's Warehouse cluster

- The Piggybacking framework

- Via the Piggybacking framework
  - Best known codes for several settings
  - Comparison with other codes
  - Preliminary practical experiments

- Summary & future work

Summary

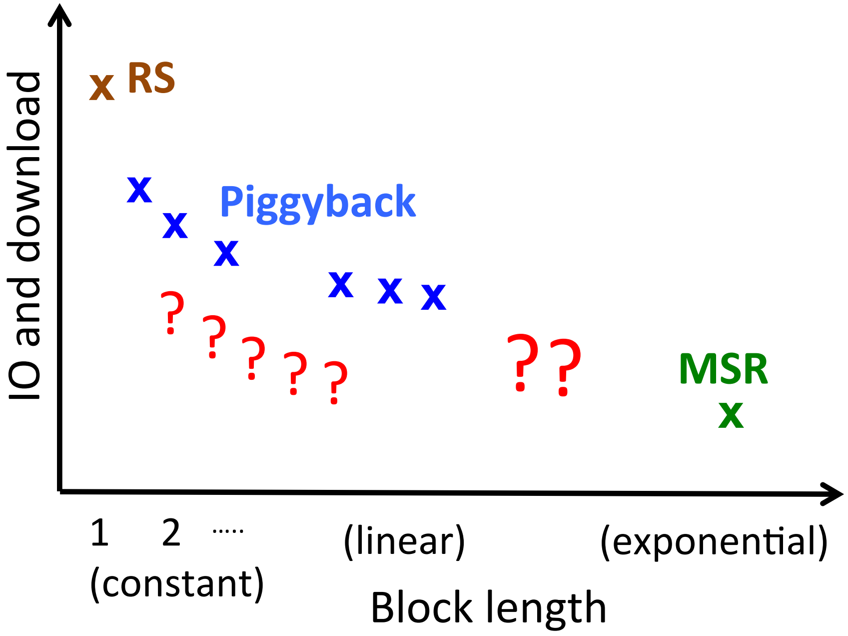
- “Piggybacking” code design framework
- 3 piggyback function designs

- Best known codes for several settings
  - MDS + high-rate + small block length
  - binary MDS (vector)
  - parity repair in regenerating codes

- Implemented in HDFS, testing in Facebook

Future work & open problems

- Comprehensive experiments at Facebook
- Other Piggybacking designs / applications
- Bounds for Piggybacking approach ?
- High-rate MDS: Tradeoff between block length & IO/download



Future work & open problems

- Comprehensive experiments at Facebook
- Other Piggybacking designs / applications
- Bounds for Piggybacking approach ?

THANKS!

- High-rate MDS: Tradeoff between block length & IO/download