ICPP/2020/EDMONTON/CANADA

TEMPLE UNIVERSITY

A Rack-Aware Pipeline Repair Scheme for Erasure-Coded Distributed Storage Systems

Tong Liu
tongliu@temple.edu

Shakeel Alibhai
shakeel.alibhai@temple.edu

Xubin He
xubin.he@temple.edu

Storage Technology and Architecture Research (STAR) Lab

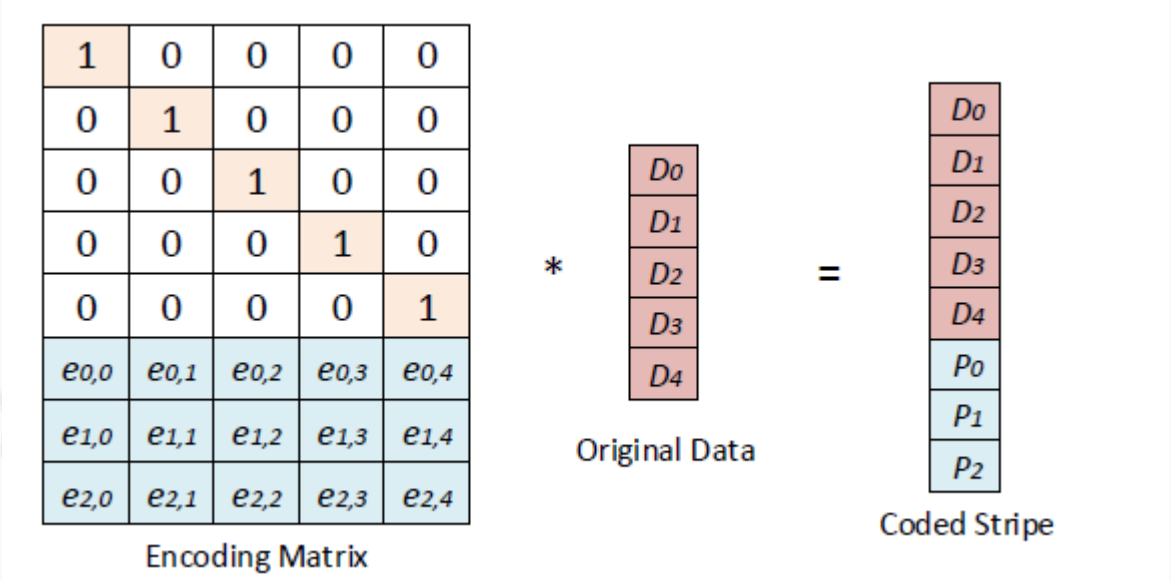
Dept. of Computer and Info. Sciences

Temple University, Philadelphia, USA

** This work was supported by the National Science Foundation (CCF-1717660, CCF-1813081 and CNS-1828363).*

Introduction & Background

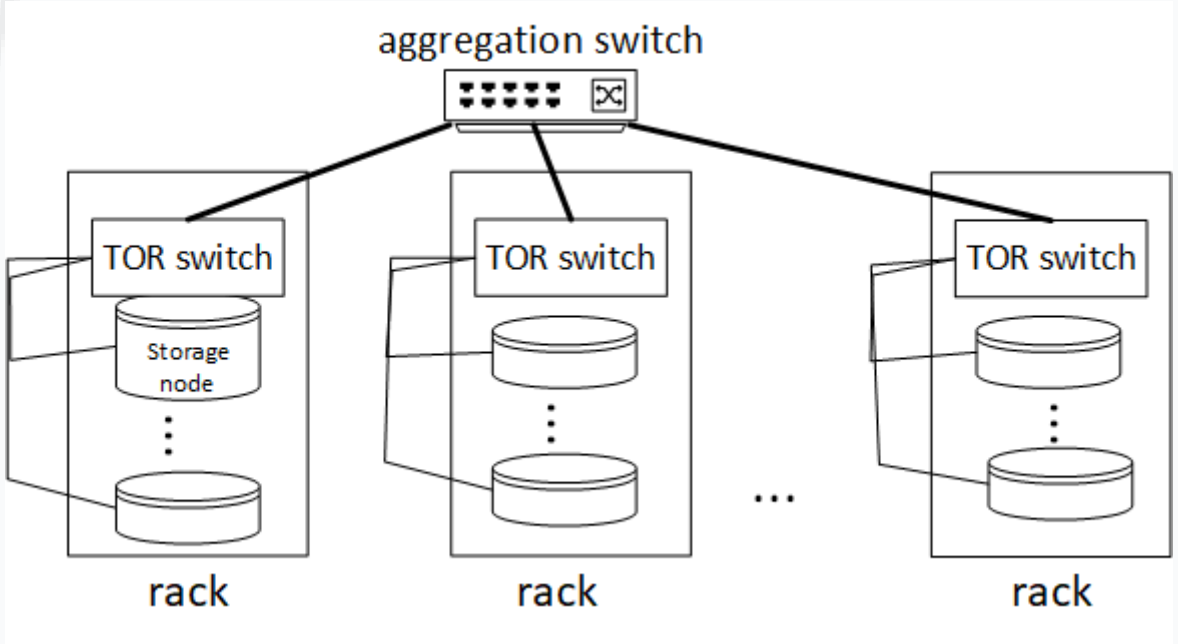
- Erasure coding
  - A popular fault-tolerant scheme which provides data reliability by adding data redundancy.
  - The Reed-Solomon (RS) code [1] is the most widely used erasure code in practice.
  - An RS (***n***, ***k***) code has ***n*** original data chunks and ***k*** parity chunks, which are the coded results from the original ***n*** data chunks.
  - Any amount of failed chunks less than or equal to ***k*** can be recovered by decoding any ***n*** of the remaining available chunks in the stripe.



[1] Reed, I. S., and Solomon, G. Polynomial codes over certain finite fields. Journal of the society for industrial and applied mathematics 8, 2 (1960), 300–304.

Introduction & Background

- Typical top-of-rack (TOR) network connectivity architecture.
  - Reconstruction requires heavy data download and consumes a large amount of disk and cross-rack bandwidth.
  - According to Facebook's report [2], a median of more than 180 Terabytes (TB) of data is transferred through the top-of-rack switches every day for this purpose.
  - The cross-rack bandwidth in production is around 1Gb/s, while the inner-rack bandwidth is 10Gb/s.



[2] K.V. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran. A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the Facebook warehouse cluster. In Proc. of USENIX HotStorage, 2013.
[3] Sathiamoorthy, M., Asteris, M., Papailiopoulos, D., Dimakis, A. G., Vadali, R., Chen, S., and Borthakur, D. Xoring elephants: Novel erasure codes for big data. In Proceedings of the VLDB Endowment (2013), vol. 6, VLDB Endowment, pp. 325–336.

RPR: inner-rack partial decoding

- Partial decoding
  - Matrix encoding/decoding process: Galois Field (GF) arithmetic.
    - the results of any operation still lie in the field.
    - addition is equivalent to XOR.
  - Consider an RS (4, 2) code
    - With four data chunks {D_0, D_1, D_2, D_3} and two parity chunks {P_0, P_1}. When D_2 fails, assume that D_0, D_1, D_3, and P_0 are selected to recover the failed chunk in the recovery node. Then the recover equation of D_2 would be: $D_0 \oplus D_1 \oplus D_3 \oplus P_0 = D_2$.
    - With partial decoding deployed, the default recovery can be divided into two parts: $D_0 \oplus D_1 = I_0$, $D_3 \oplus P_0 = I_1$, $I_0 \oplus I_1 = D_2$.
    - With partial decoding, the lost data can be decoded partially and in parallel, thus mitigating the transfer bottleneck and load imbalance issues.

RPR: inner-rack partial decoding

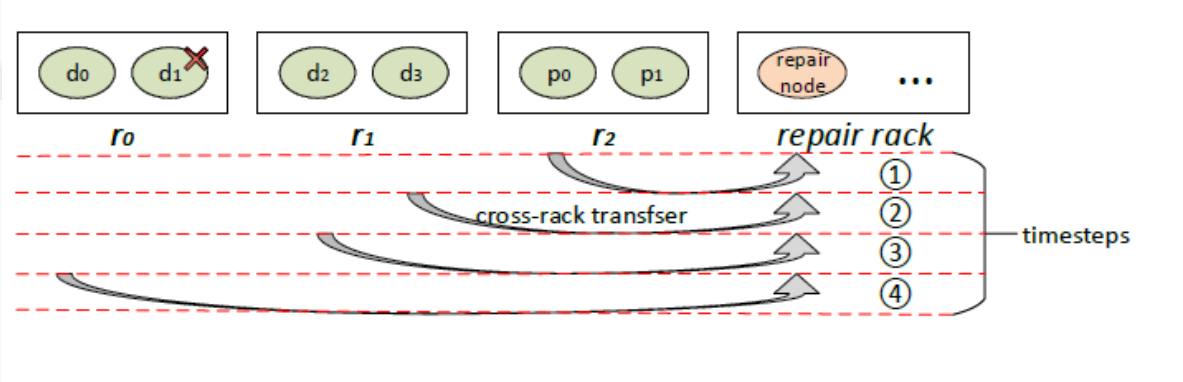
➢The ***Inner*** algorithm and an example

Algorithm 1: ***Inner*** $(d_0, d_1, \ldots, d_{n-1})$

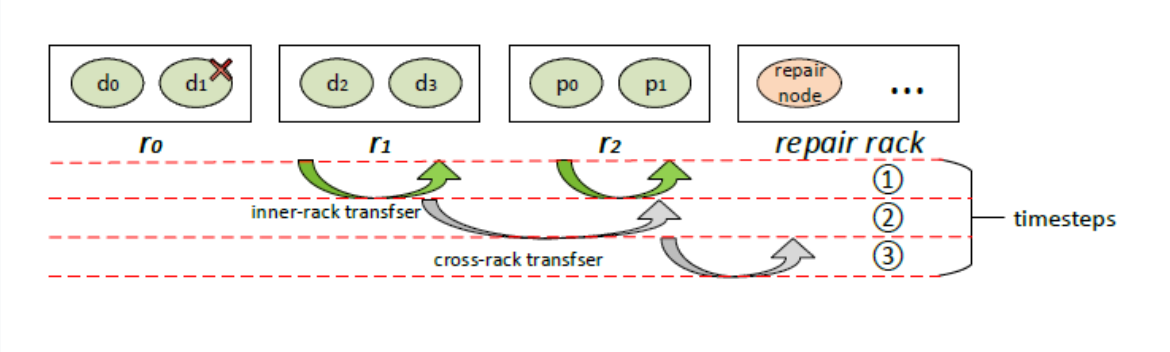
input : Selected data blocks in the rack involved in the repair process, $\{d_0, d_1, \ldots, d_{n-1}\}$.

output: An intermediate block I that will be cross-rack transferred for further decoding.

```
if n == 0 then
    return I
if n%2 == 0 then
    for a ← 0 to (n−2)/2 do
        i_a = d_2a ⊕ d_2a+1;
    Inner (i_0, i_1, ..., i_(n−2)/2)
else
    for a ← 0 to (n−3)/2 do
        i_a = d_2a ⊕ d_2a+1;
    Inner (i_0, i_1, ..., i_(n−3)/2 ⊕ d_n−1)
```



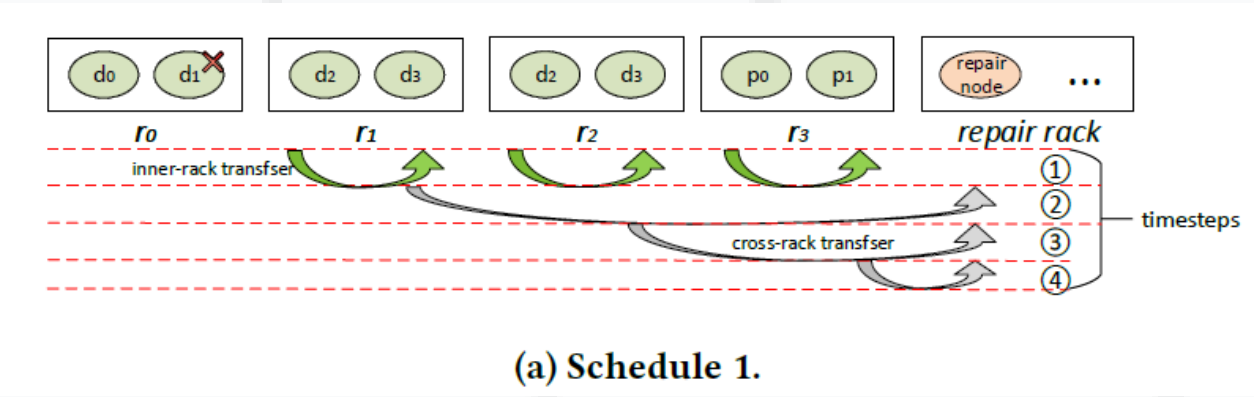
Traditional repair with an RS (4, 2) code when single-block failure occurs



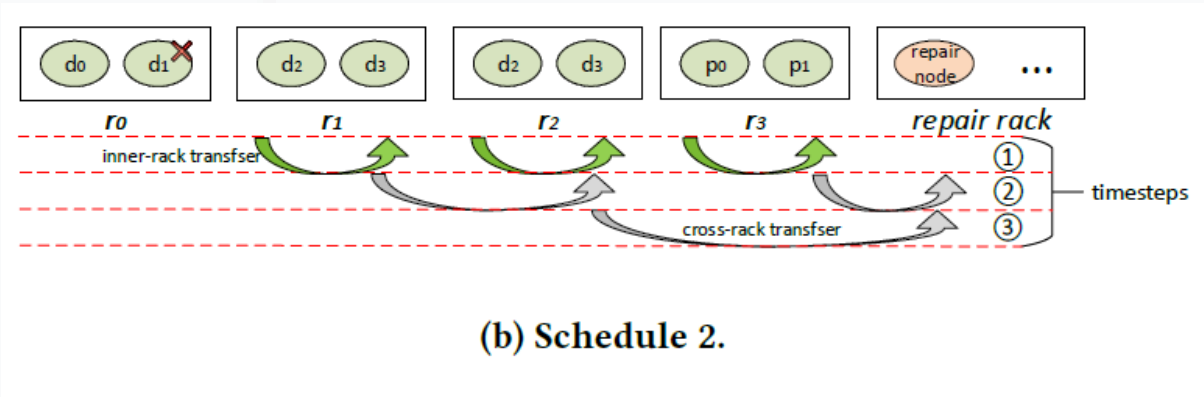
Repair with inner-rack partial decoding

Different schedules in more complicated scenario

- For simple RS code, after each intermediate block is generated, the cross-rack transfer schedule would be straightforward.
- However, when multiple failures occur, or when there are more nodes involved in the repair process, the inner-rack and cross-rack transfer schedules would be much more complex.



(a) Schedule 1.



(b) Schedule 2.

RPR: cross-rack pipeline scheduling

- To address this issue, we propose a pipeline-based greedy algorithm ***Cross***, which aims to maintain the data consistency during the entire repair process as well as achieve the optimal total repair time.
- The ***Cross*** algorithm works together with the ***Inner*** algorithm: whenever ***Inner*** finishes the inner-rack data transfer and produces the intermediate data, ***Cross*** will give the optimal cross-rack transfer schedule based on the current node and rack transfer status.

RPR: cross-rack pipeline scheduling

Algorithm 2: *Cross* (nodes and racks)

input : Failed block and its corresponding rack ID; nodes in each rack that are in the same stripe and their corresponding rack IDs.

output: The reconstructed data block d_r.

if *recovery node gets all intermediate data* **then**
    do last partial decoding;
    return d_r;

for *each rack* **do**
    if *inner decoding is possible* **then**
        Inner (nodes in current rack);
    else
        start cross-rack transfer with any other rack which has no inner-rack transfer;
    if *cross-rack transfer with any other rack which has no cross-rack transfer is possible* **then**
        start cross-rack transfer with selected rack;
    else
        wait until the rack which finishes the cross-rack first, then start transfer;

Cross (racks that conduct inner-rack partial decodings in the next timestep)

- Process of ***Cross*** algorithm
  1. For each rack, if IR decoding can be conducted, start the IR transfer and partial decoding.
  2. If an IR transfer cannot be scheduled, then start a CR transfer with any other rack which currently has no IR transfer.
  3. When the IR transfer finishes, start a CR transfer with any other rack that does not currently have a CR transfer.
  4. If every other rack has a CR transfer, then wait until the one that finishes first, and then start the transfer.
  5. Repeat recursively until all the racks finish their CR transfers or the recovery node receives all the intermediate data.

IR = inner-rack, CR = cross-rack

RPR: data pre-placement

- Intuitively, to reduce the cross-rack data transfer, it would be better to transfer as much information as possible in each cross-rack transfer.
- Thus, to reduce the amount of cross-rack transfers and avoid building the decoding matrix, the best way is to put P_0 with data blocks instead of parity blocks in the same rack.
- For an RS (n, k) code, if the first parity block P_0 is placed with all data blocks in the same rack, then, assuming the block failure rate is same for all blocks, there is a $1/n$ chance that there is no need to build the decoding matrix when a single-block failure occurs.
- Our experiments show that building the decoding matrix takes 80% of the total decoding time.

Extension to multi-block failures

- When multiple failures occur, the repair procedure becomes more complicated since multiple decoding equations are required.
- Existing cross-rack repair scheme, such as CAR [4] can only tolerate single-failure scenario.
- Similar to ***Inner*** and ***Cross*****,** we propose ***Inner-multi*** and ***Cross-multi*** algorithms which can reduce the cross-rack traffic and total repair time when multiple failures occur.

[4] Shen, Z., Shu, J., and Lee, P. P. Reconsidering single failure recovery in clustered file systems. In 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (2016), IEEE, pp. 323–334.

Extension to multi-block failures

- Assume an (*n*, *k*) erasure coded system:

$$\begin{aligned} & e_{0,0} * d_0 \oplus e_{0,1} * d_1 \oplus \ldots \oplus e_{0,n-1} * d_{n-1} = p_0 \\ & \ldots \\ & e_{k-1,0} * d_0 \oplus e_{k-1,1} * d_1 \oplus \ldots \oplus e_{k-1,n-1} * d_{n-1} = p_{k-1} \end{aligned} \quad (7)$$

- Assume there are *k* failed blocks

$$\begin{aligned} & e'_{0,0} * d_0 \oplus e'_{0,1} * d_1 \oplus \ldots \oplus e'_{0,n-1} * p_{k-1} = d_{f,0} \\ & \ldots \\ & e'_{k-1,1} * d_0 \oplus e'_{k,1} * d_1 \oplus \ldots \oplus e'_{k-1,n-1} * p_{k-1} = d_{f,k-1} \end{aligned} \quad (8)$$

- To recover, data on the left side of each equation is needed.

Extension to multi-block failures

- Assume that the stripe is distributed across q racks, then every rack can only transfer an intermediate data block to the recovery node/rack:

$$
\begin{aligned}
&I_{0,0} \oplus I_{0,1} \oplus \ldots \oplus I_{0,q-1} = d_{f,0} \\
&\ldots \\
&I_{k-1,0} \oplus I_{k-1,1} \oplus \ldots \oplus I_{k-1,q-1} = d_{f,k-1}
\end{aligned} \tag{9}
$$

- Similarly, ***Inner-multi*** and ***Cross-multi*** algorithms can be applied.

Evaluation setup

- Simics simulation
  - Simics is a platform for full-system simulation
  - Each node/server has a single-core Intel(R) Core(TM)-i7 CPU @ 2.00 GHz with 4GB RAM running on Ubuntu 16.04.4 LTS.
  - *Wondershaper* is applied to achieve the different inner-rack transfer and cross-rack transfer bandwidth. Inner-rack bandwidth is set to be 1Gb/s, and cross-rack bandwidth is set to be 0.1Gb/s.
  - The erasure-coded RPR prototype is built based on the *Jerasure* Library v2.0.
  - Each data/parity block is 256MB.

Evaluation setup

➢Amazon Web Service EC2 Evaluation

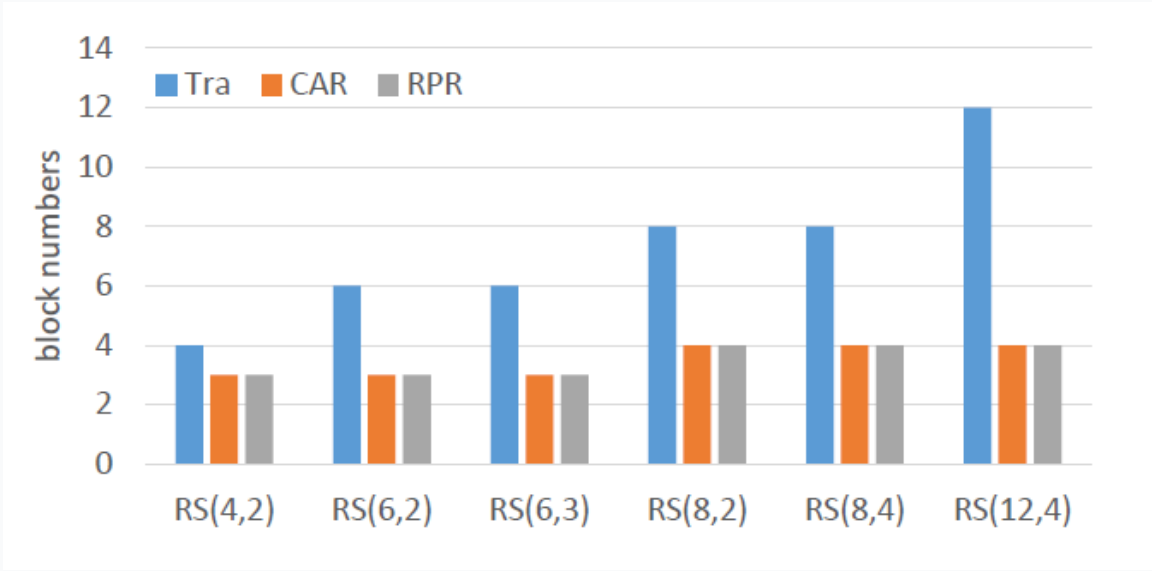
- To validate that RPR works in real-world systems, we also evaluate RPR in AWS EC2.
- To simulate the rack-level data transfer, we launch instances (virtual machines) in five different continents.
- Each virtual machine is created based on a t2.micro type Linux Kernel 4.14 Amazon Linux 2 AMI with 1 vCPU, 1 GB RAM, and 8 GB SSD.
- Other settings same as Simics simulation.

	Ohio	Tokyo	Paris	São Paulo	Sydney
Ohio	583.39	51.798	59.281	67.613	41.4
Tokyo		583.26	45.56	41.605	91.21
Paris			641.403	56.57	40.79
São Paulo				631.416	34.44
Sydney					565.39

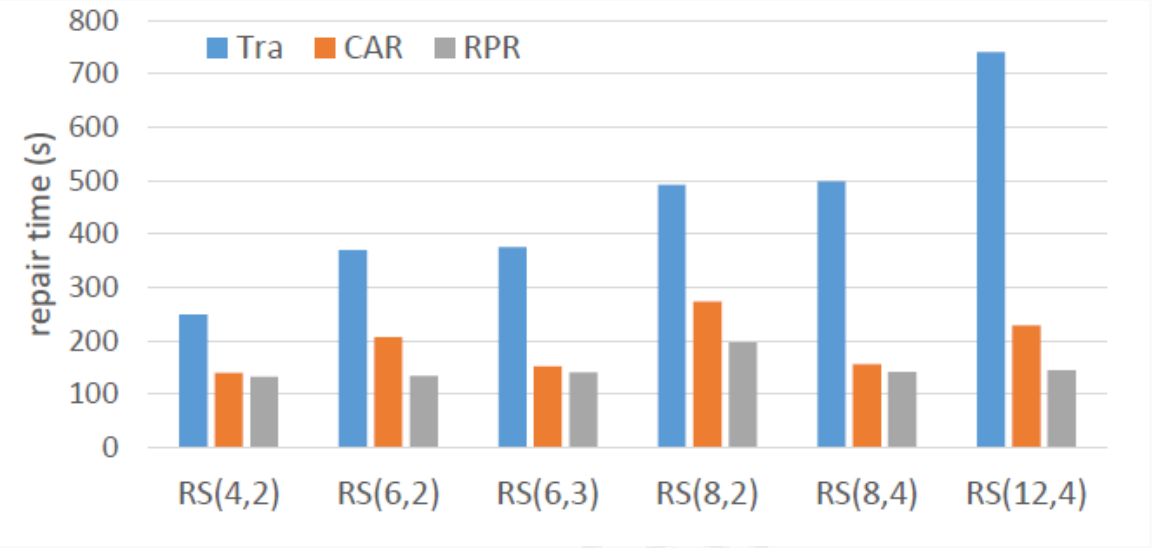
Inter- and intra-bandwidths (Mbps) across regions

Experimental results

- Single-block failure (Simics simulation)



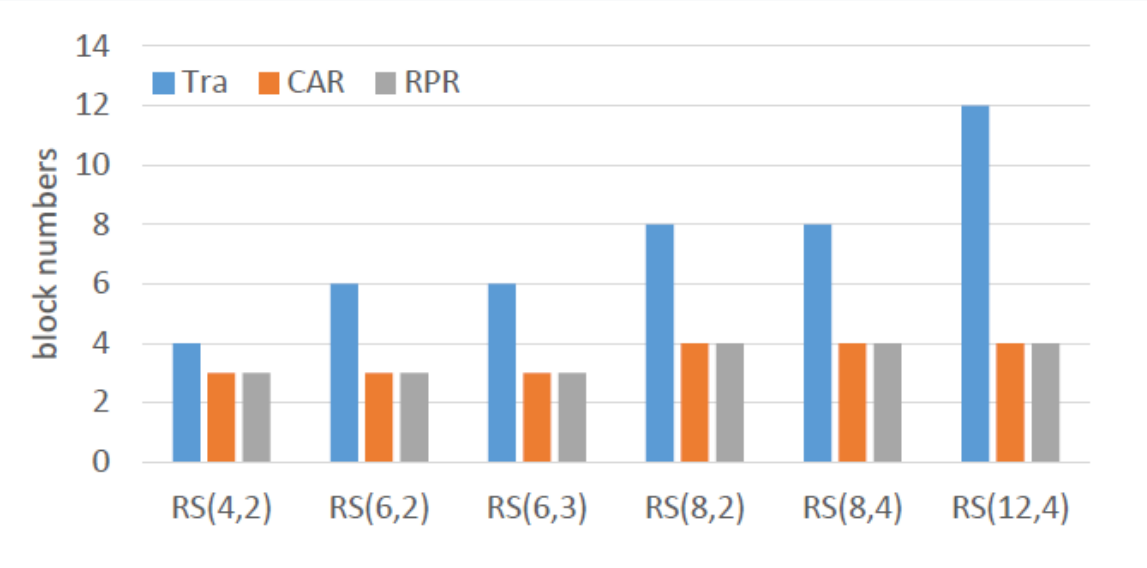
Cross-rack traffic



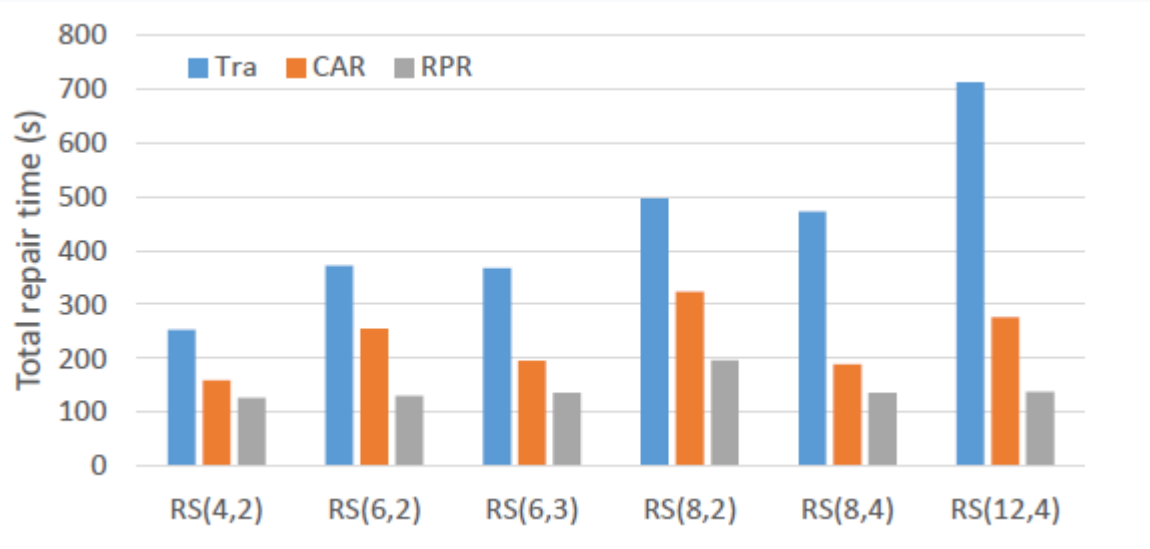
Total repair time

Experimental results

- Single-block failure (EC2 evaluation)



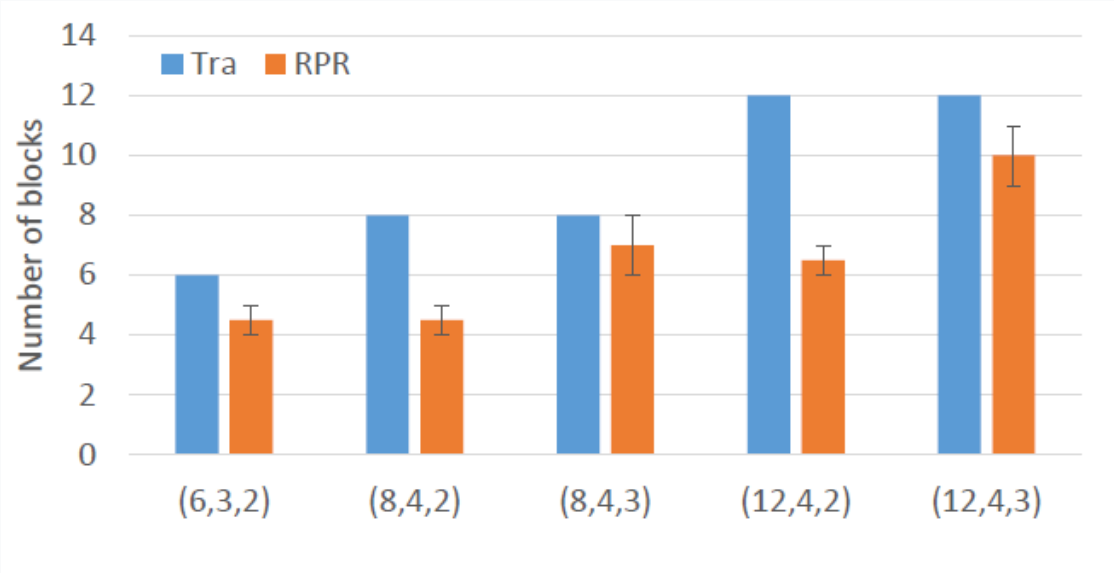
Cross-rack traffic



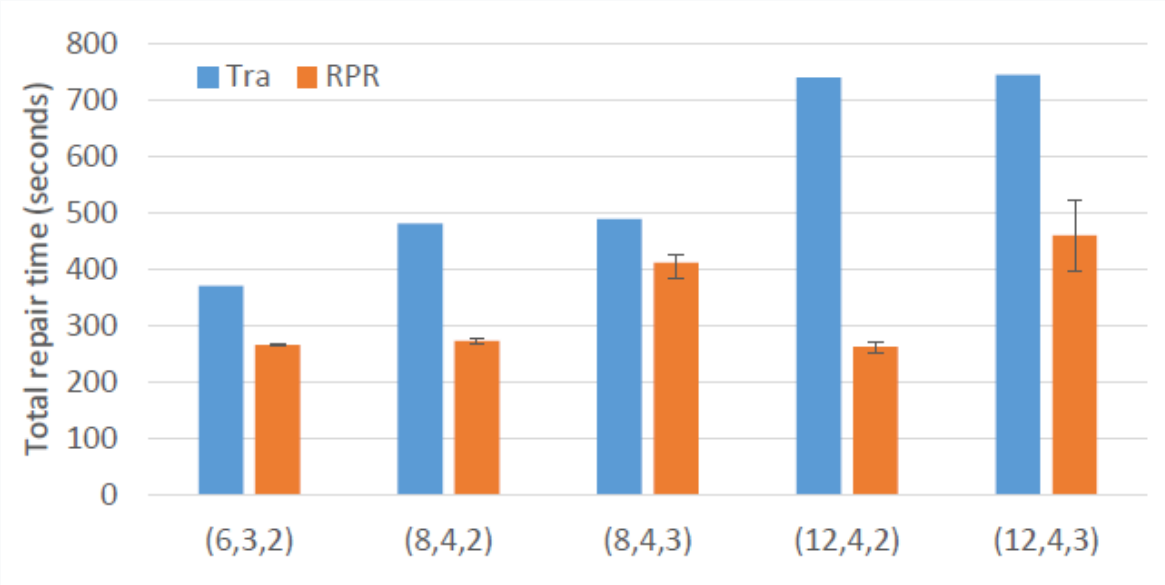
Total repair time

Experimental results

- Multi-block failure (Simics simulation)



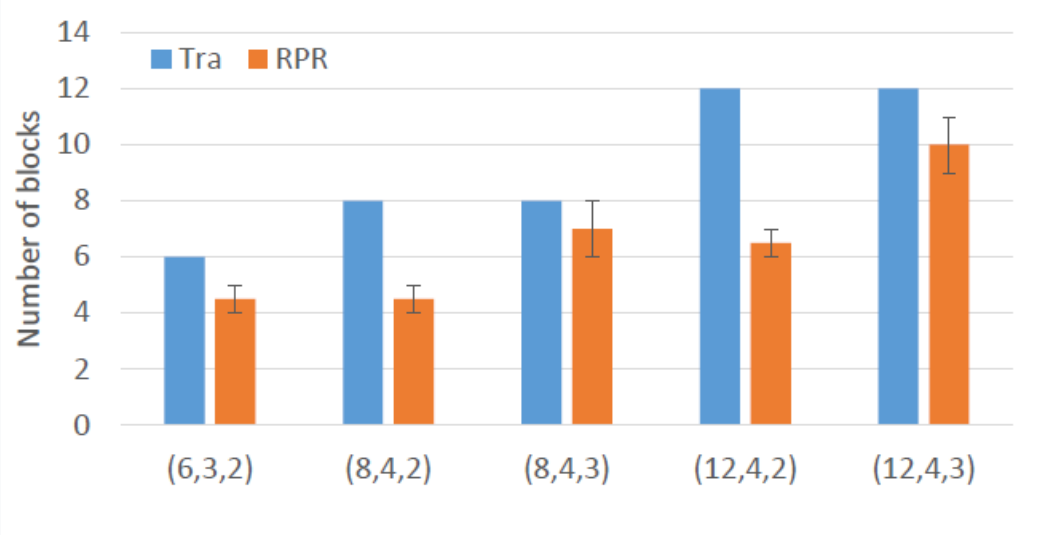
Cross-rack traffic



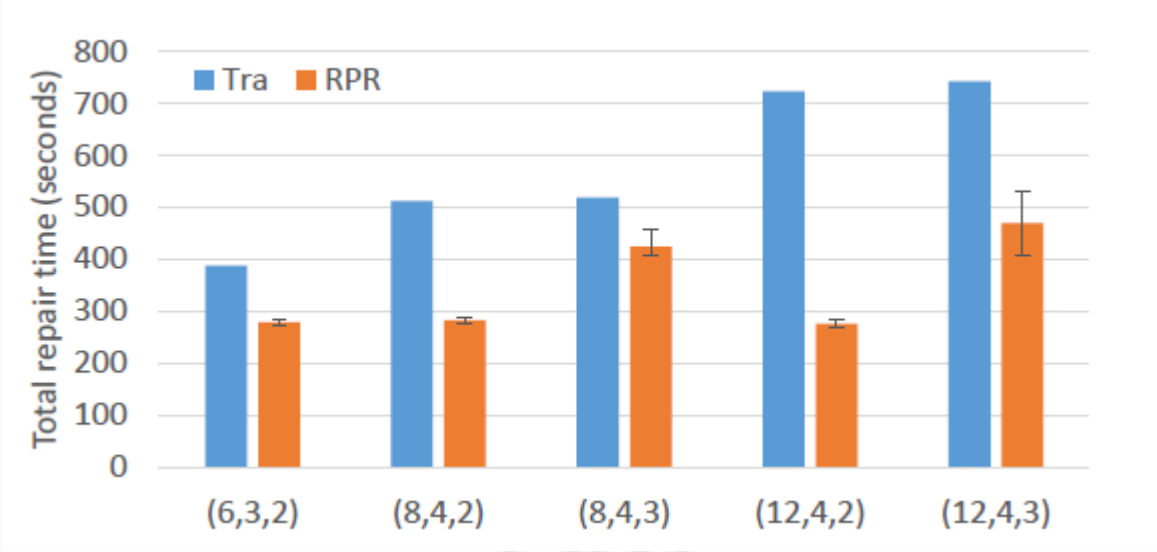
Total repair time

Experimental results

- Multi-block failure (EC2 evaluation)



Cross-rack traffic



Total repair time

Conclusion

- We propose RPR, a rack-aware pipeline repair scheme comprised of three techniques
  - inner-rack partial decoding
  - cross-rack pipeline scheduling
  - data-parity placement
- RPR supports both single-block and multi-block failures.
- We conduct experiments on both Simics and AWS EC2, results from both platforms show that,
  - In single-block failure scenario, RPR significantly improves the repair performance compared to the traditional RS code as well as CAR, with total repair time reductions of up to 81.5% and 50.2%, respectively.
  - When multiple blocks fail, RPR also improves the repair performance compared to the traditional RS code repair with total repair time reductions of up to 64.5% and cross-rack data transfer reductions of up to 50%.

TEMPLE UNIVERSITY

Thank You & Questions?