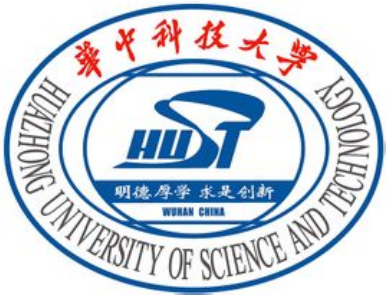
XORInc: Optimizing Data Repair and Update for Erasure-Coded Systems with XOR-Based In-Network Computation

Yingjie Tang

Huazhong University of Science & Technology



Outline

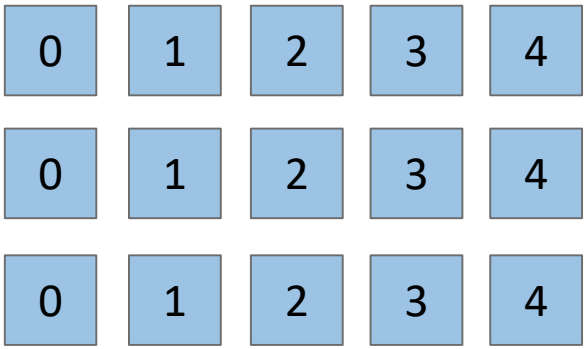
- **Introduction**
- **Motivation**
- **Design**
- **Evaluation**
- **Conclusion**

Outline

- Introduction
- Motivation
- Design
- Evaluation
- Conclusion

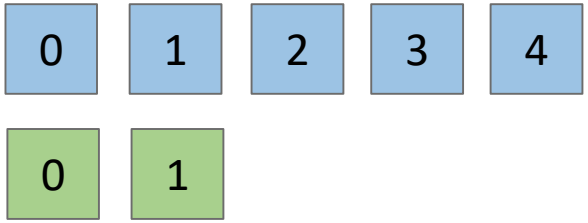
Replication vs erasure coding

- **Multiple replications**
  - **High performance**
  - **Fast recovery**
  - **High storage overhead**



Three replications

- **Erasure coding**
  - **Low storage overhead**
  - **High network traffic** — **Data repair**, **Data update**



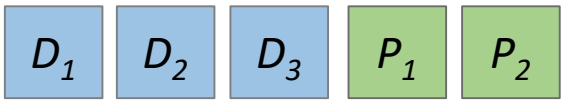
RS(7, 5) $k = 5, r = 2$

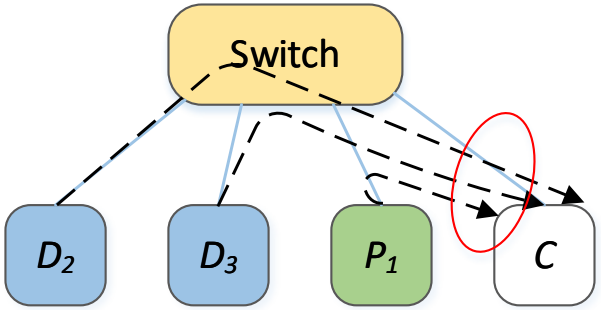
Conventional repair

- **Repairing a single failed data block needs to read other *k* available blocks from multiple nodes**

$$d^* = \alpha_1 b_1 + \alpha_2 b_2 + \cdots + \alpha_k b_k$$

- **The larger the *k* is, the worse performance conventional repair scheme has (*k* = 12 in Azure and *k* = 10 in Facebook)**





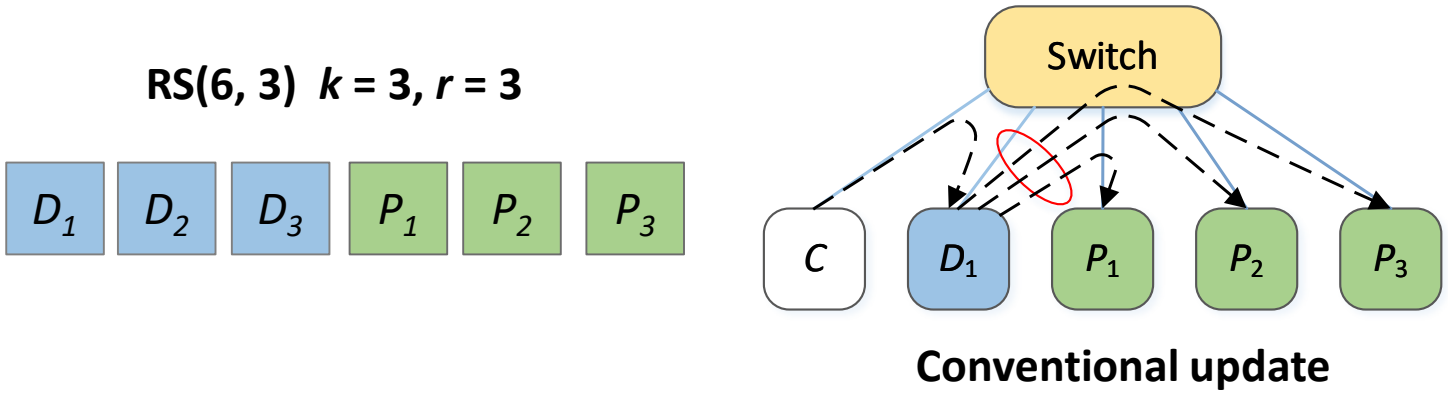
Conventional repair

Conventional update

- **Updating a data block triggers parity updates for *r* parity blocks**

$$p_i = \beta_{i,1} d_1 + \beta_{i,2} d_2 + \cdots + \beta_{i,k} d_k, \quad i = 1, 2, \cdots r$$

- **The larger the *r* is, the worse performance conventional update scheme has (*r* = 4 in Azure and Facebook)**



Outline

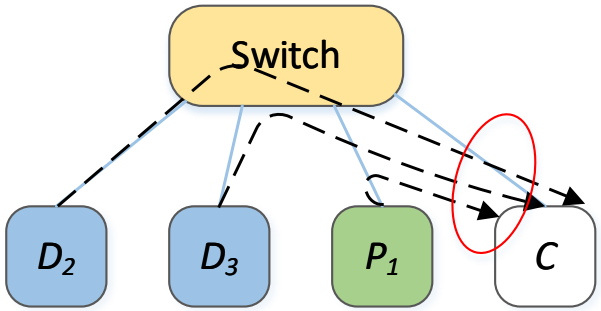
- Introduction
- Motivation
- Design
- Evaluation
- Conclusion

Existing repair schemes

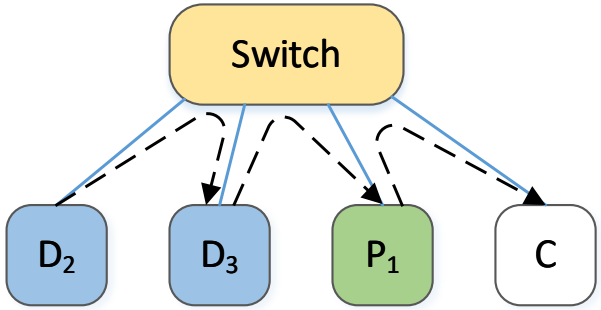
- **Conventional repair: A star-structured repair scheme**
  - **Shortage: A single bottleneck link = O(*k*) repair performance**

- **Repair pipelining (ATC'17): A chain-structured repair scheme**
  - **Optimization: No bottleneck = O(1) repair performance**
  - **Shortage: Transferring traffic instead of reducing it**

$$d^* = \alpha_1 b_1 + \alpha_2 b_2 + \cdots + \alpha_k b_k$$



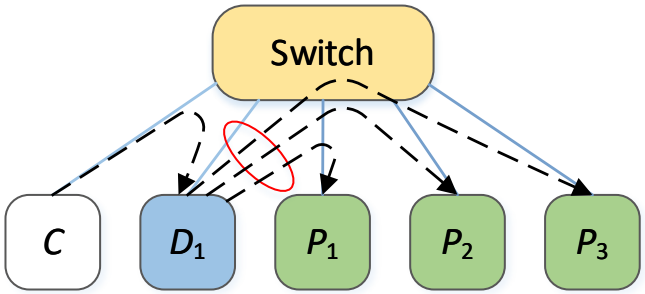
Conventional repair



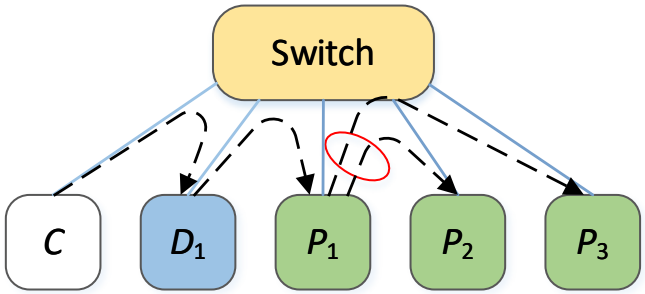
Repair pipelining

Existing update schemes

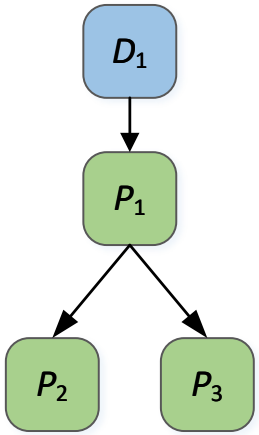
- **Conventional update: A star-structured update scheme**
  - **Shortage: A single bottleneck link = O(*r*) update performance**
- **T-Update (INFOCOM'16): A tree-structured update scheme**
  - **Optimization: O(*t*) update performance (*t* < *r*)**
  - **Shortage: No reduction in traffic & Failure to achieve O(1) performance**



Conventional update



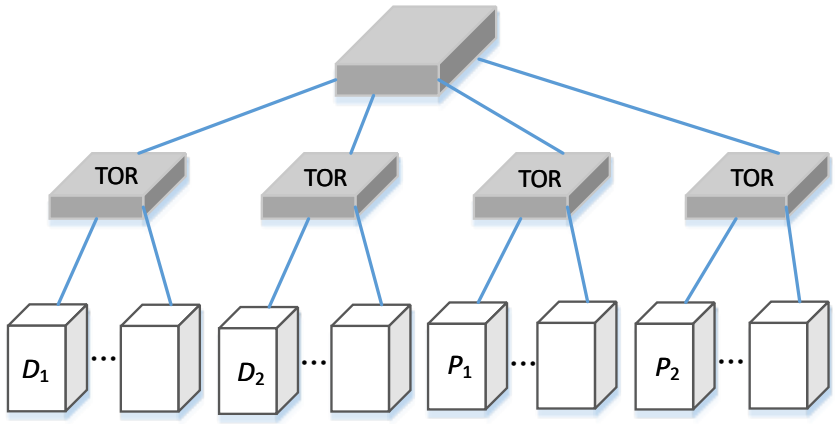
T-Update



Summary

- **Data repair: No reduction in traffic**
- **Data update: No reduction in traffic & Failure to achieve O(1) performance**

For better system reliability, blocks on the same stripe are distributed in different racks. Therefore, the network traffic introduced by repair and update operations is cross-rack traffic.



Outline

- Introduction
- Motivation
- Design
- Evaluation
- Conclusion

Design goal & related technology

O(1) repair and update performance & significant reduction in network traffic

- **The emergency of in-network computation**
  - **Development: Benefit from the rapid development of software-defined networking and programmable network devices**
  - **Main idea: Move the computations originally on the storage nodes to the network**
  - **Advantages for distributed applications: Continue…**

Advantages of in-network computation

- **Remote requests for back-end storage nodes can be responded in advance in the network, thereby reducing latency**
- **Reduce end-to-end traffic and mitigate network congestion**
- **Servers can be put in low-power mode (e.g., Intel C6 state) or even be turned off or removed, thus saving energy costs**

Related works

- **NetCache (SOSP'17)**
  - **Goal: Achieve load balance of in-memory key-value stores**
  - **Main idea: leverage switches to cache a little hot data in the network**
  - **Implementation: Switch data plane is written in P4 and is compiled to Barefoot Tofino ASIC**

- **IncBricks (ASPLOS'17)**
  - **Goal: Reduce the response latency of in-memory key-value stores**
  - **Main idea: leverage switches to cache large amounts of data for improving the cache hit ratio**
  - **Implementation: Cavium XPliant switch and network accelerator (OCTEON or LiquidIO)**

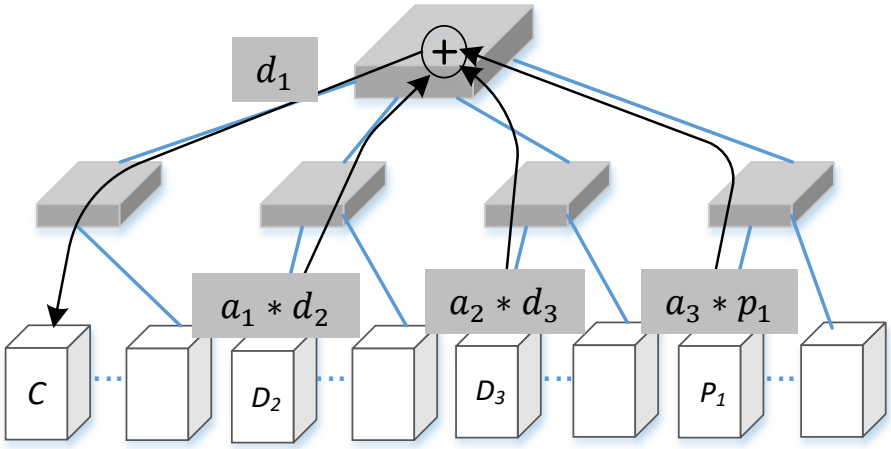
Related works

- **NetRS (ICDCS'18)**
  - **Goal: Improve replica selection effectiveness for key-value stores to reduce response latency**
  - **Main idea: Enable in-network replica selection and support diverse algorithms of replica selection (high computation overhead)**
  - **Implementation: Simulation (programmable switches and network accelerators)**

Related works focused on in-memory key-value store

Whether in-network computation can be applied to other distributed applications, such as erasure coding?

NetRepair

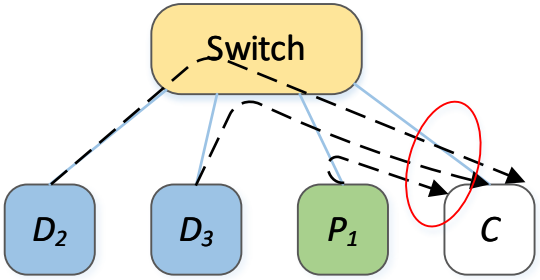


$$d_1 = [a_1 \quad a_2 \quad a_3] \begin{bmatrix} d_2 \\ d_3 \\ p_1 \end{bmatrix} = a_1 * d_2 + a_2 * d_3 + a_3 * p_1$$

Since the computations are done in the network, end-to-end traffic is reduced and there is no bottleneck link.

1. **Each node performs multiplication before sending data**
2. **The nodes send the result of the multiplication to the switch**
3. **After the switch receives the data, it performs XOR on the data from multiple nodes**
4. **The switch sends the result back to the client**

Performance analysis

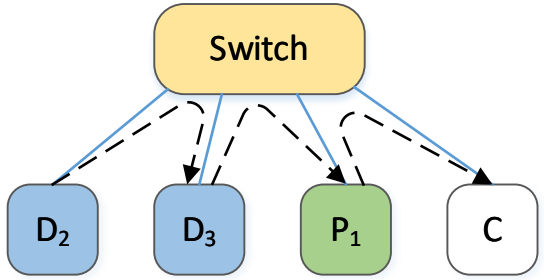


Conventional repair

$\{D_2 \rightarrow S \rightarrow C, D_3 \rightarrow S \rightarrow C, P_1 \rightarrow S \rightarrow C\}$

Repair performance: O(*k*)

Network traffic: 2k block (k = 3)***

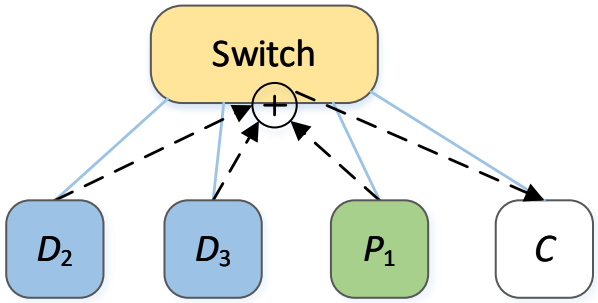


Repair pipelining

$D_2 \rightarrow S \rightarrow D_3 \rightarrow S \rightarrow P_1 \rightarrow S \rightarrow C$

Repair performance: O(*1*)

Network traffic: 2k block (k = 3)***



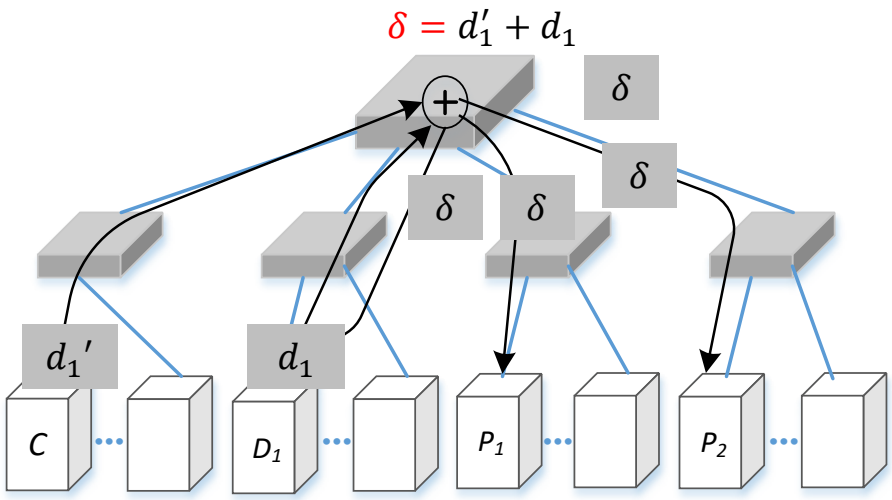
NetRepair

$\{D_2, D_3, P_1\} \rightarrow S \rightarrow C$

Repair performance: O(*1*)

Network traffic: *k + 1 block (k = 3)*

NetUpdate



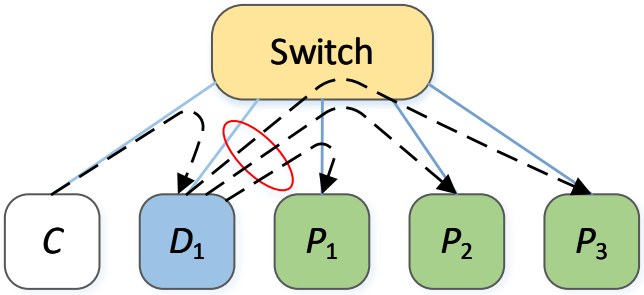
$$p_1' = a_1 * \delta + p_1$$

$$p_2' = a_2 * \delta + p_2$$

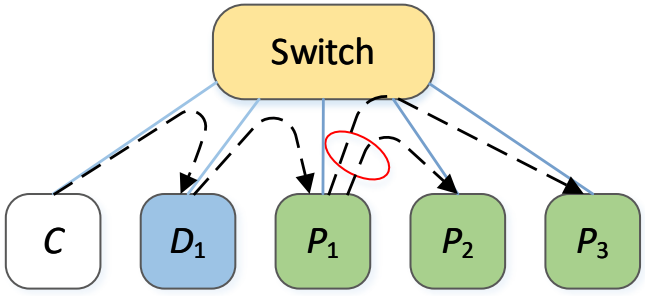
1. Data node D_1 and client send data to the switch
2. After the switch receives the data, it performs XOR to obtain the δ
3. The switch sends the δ to the data node D_1 and all parity nodes

Since the computations are done in the network, end-to-end traffic is reduced and there is no bottleneck link.

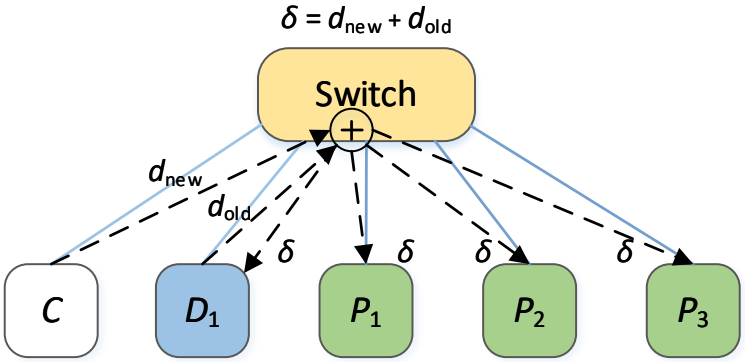
Performance analysis



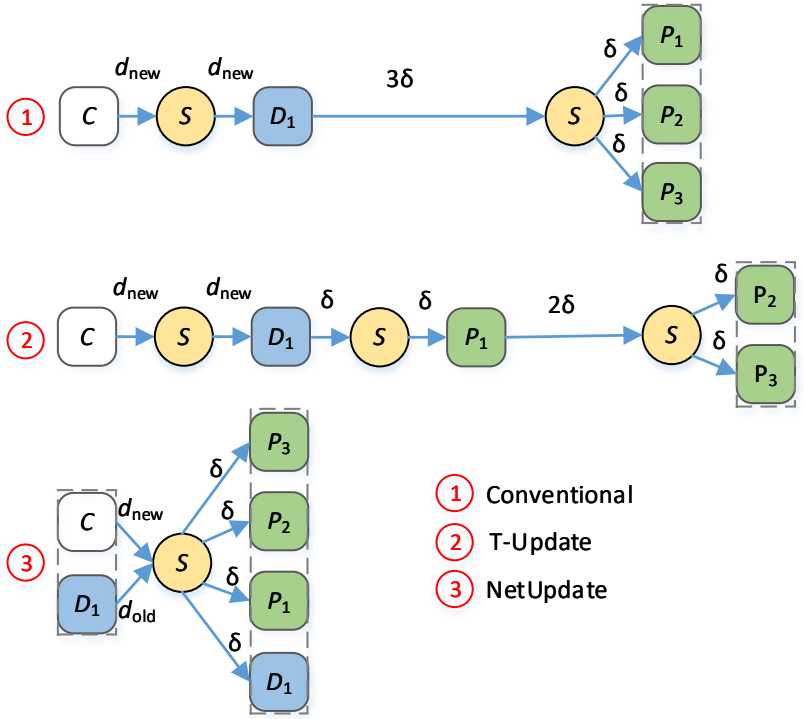
Conventional update



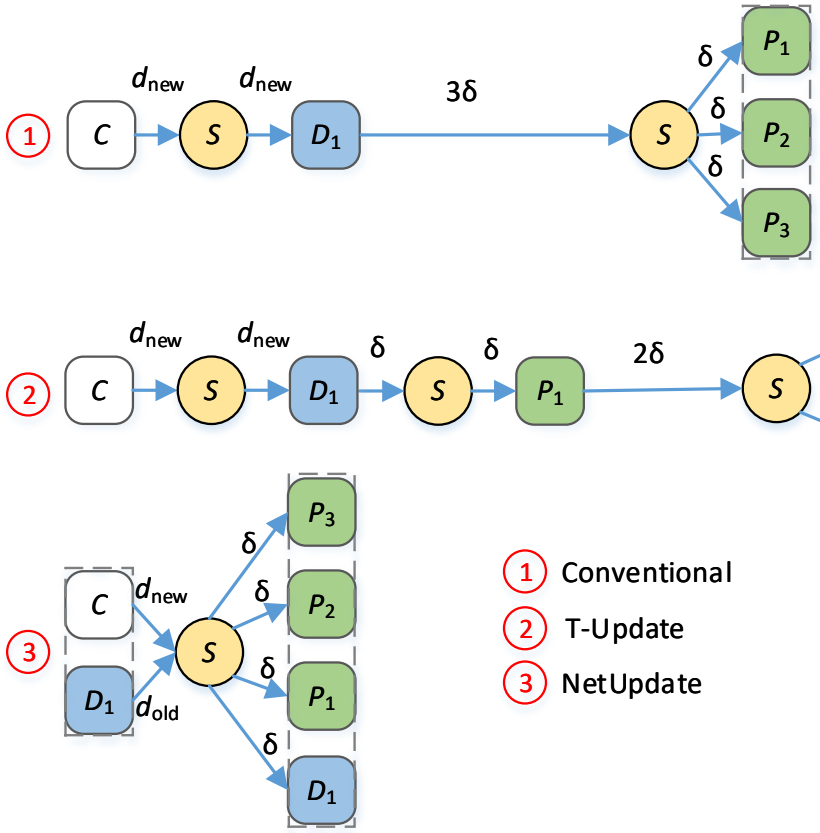
T-Update



NetUpdate



Performance analysis



Update performance: O(*r*)
Network traffic: 2*r* + 2 (*r* = 3)

Update performance: O(*t*) (*t* < *r*)
Network traffic: 2*r* + 2 (*r* = 3)

Update performance: O(*1*)
Network traffic: *r* + 3 (*r* = 3)

How to select switches

- **This work is done by the SDN controller**

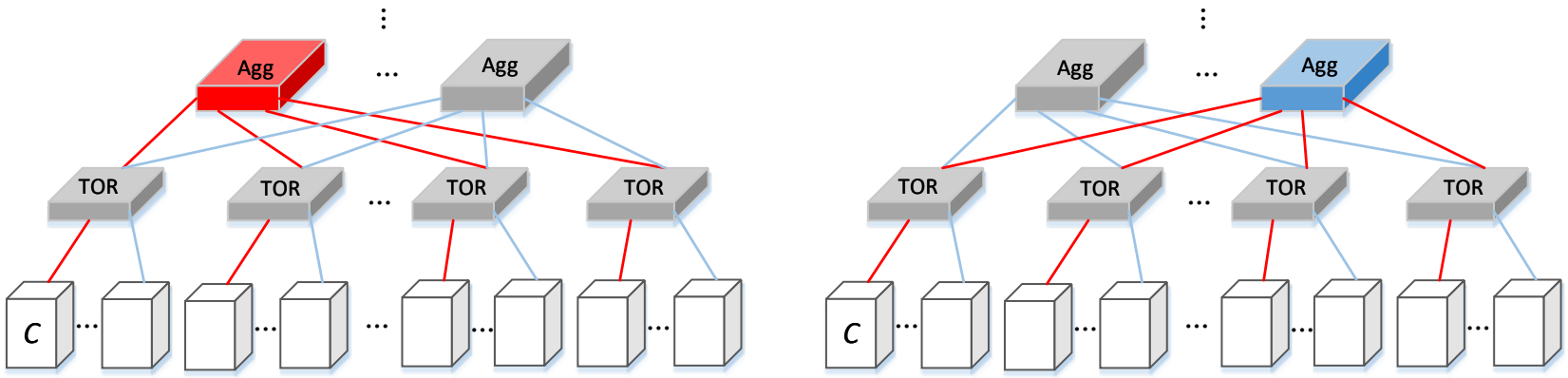
  There are two basis:

  - **Datacenter networks offer multiple paths between storage nodes**
  - **SDN controller maintains the global network topology information**

- **Selection strategy: Three steps**

How to select switches

- **Selection strategy:**
  1. **for the *x* nodes participated in the repair or update operation, the SDN controller finds the *y* shortest paths from each node to the client**
  2. **Each *x* shortest path forms a transmission network, and a total of y^x transmission networks can be found**
  3. **For each transmission network, determine if the switch at the intersection of the paths is overloaded**



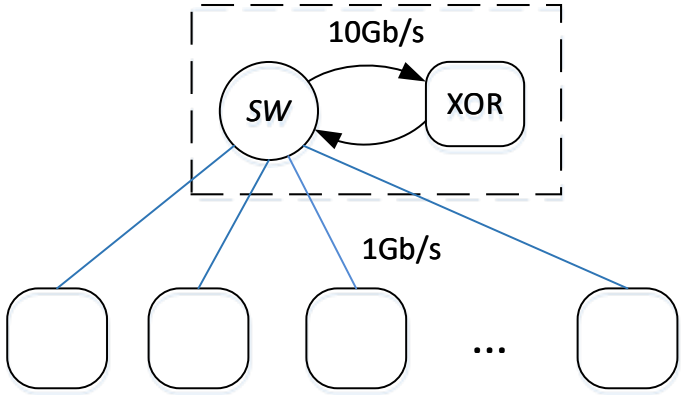
Outline

- Introduction
- Motivation
- Design
- Evaluation
- Conclusion

Evaluation

- **Environment**
  - **Storage cluster: HDFS-RAID with 18 docker containers**
  - **SDN network: floodlight + Open vSwitch (1Gb/s)**
  - **In-network computation: Simulation**
  - **Erasure coding: RS(5,3) (default)**

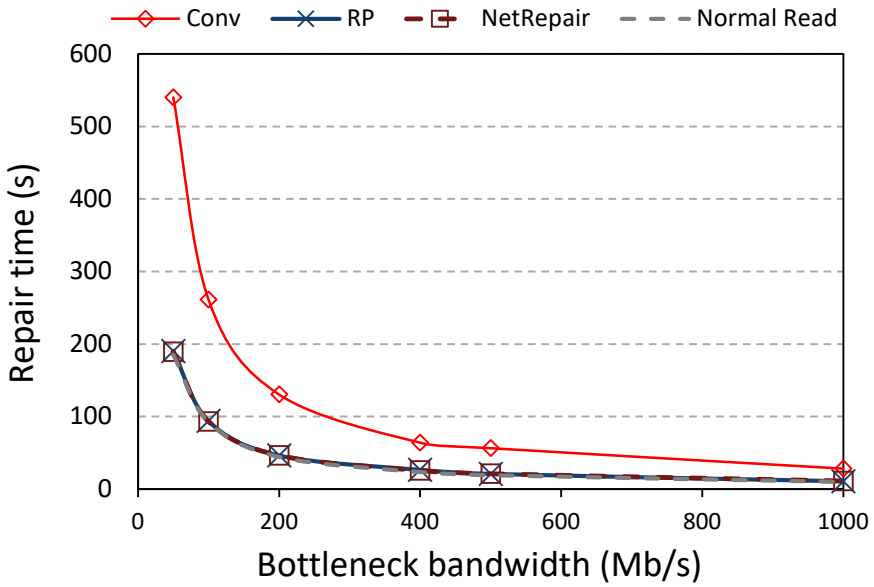
- **Performance metrics**
  - **Repair time for 1GB data**
  - **Update time for 10MB data**
  - **Network Traffic**



Bottleneck bandwidth

- **NetRepair can make the performance of the degraded read the same as the normal read**

Bandwidth (Mb/s)	Repair time (s)			
	Conv	RP	NetRepair	Normal Read
50	540.17	190.11	189.28	185.29
100	261.22	93.86	93.13	92.46
200	130.48	46.12	46.27	44.19
400	64.1	26.29	25.43	23.37
500	56.47	20.67	21.02	18.97
1000	28.31	10.56	11.14	9.77

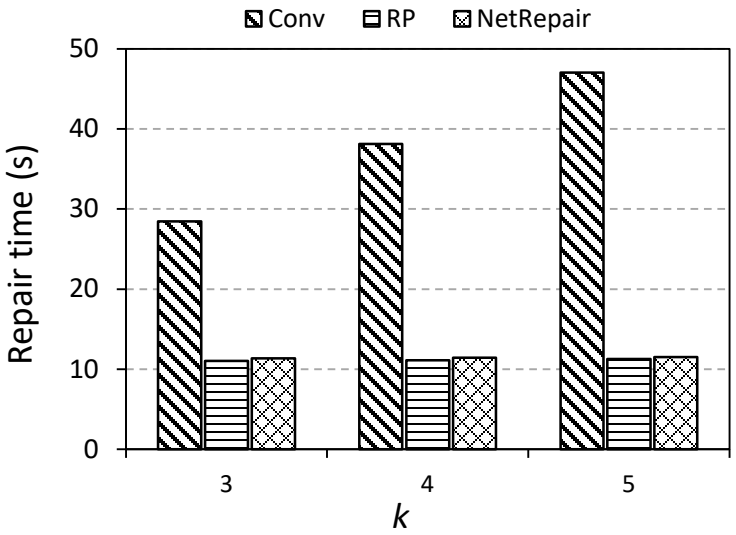


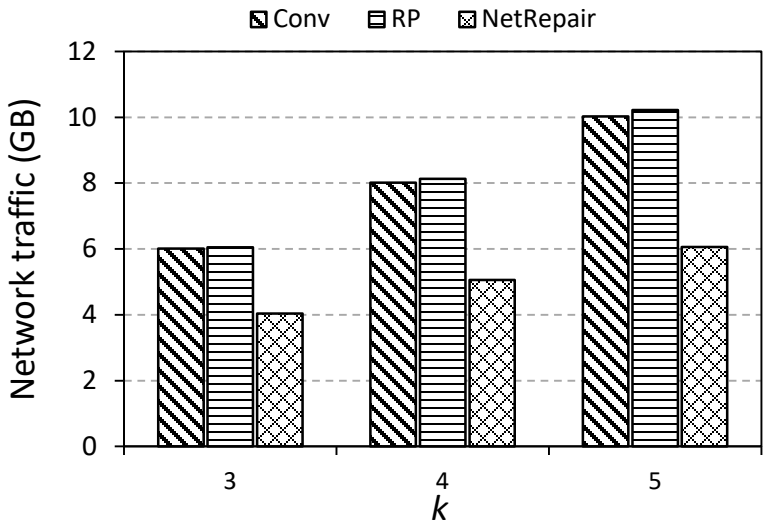
Repair time is inversely proportional to the bottleneck bandwidth

Repair time:
Conv ≈ 3*NetRepair
NetRepair ≈ RP≈ Normal Read

Coding parameter

- **Repair performance of NetRepair is independent of *k***
- **The larger the *k* value is, the more traffic that NetRepair reduces**



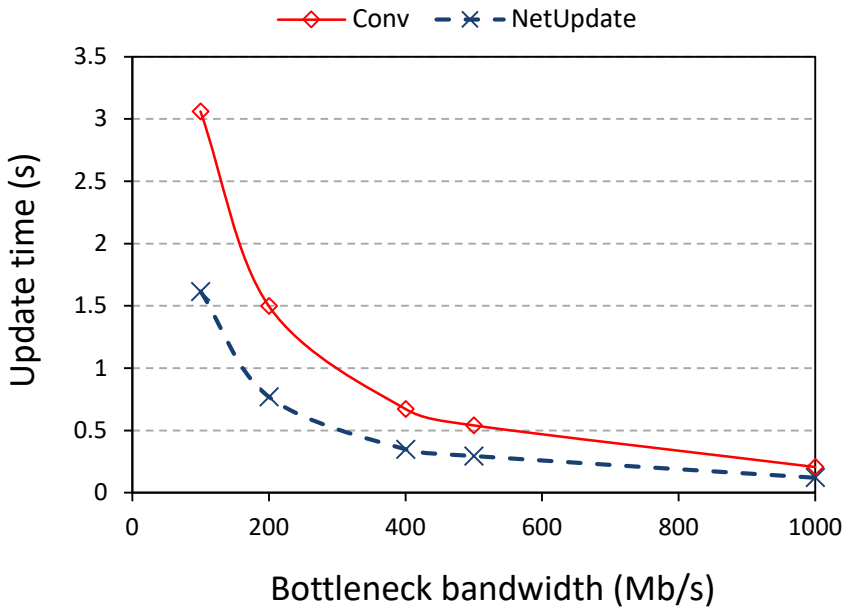


Repair time: NetRepair always guarantees optimal performance

Network traffic: As k increases from 3 to 5, the network traffic reduction of NetRepair increases from 33% to 41%

Bottleneck bandwidth

- **NetUpdate significantly reduces update time**

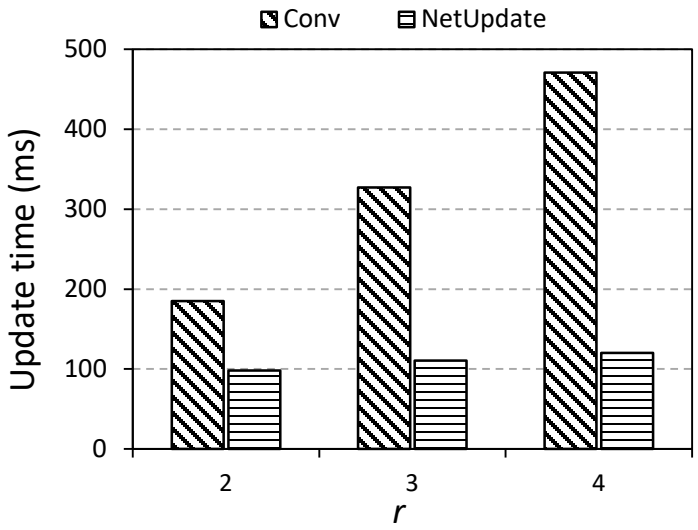


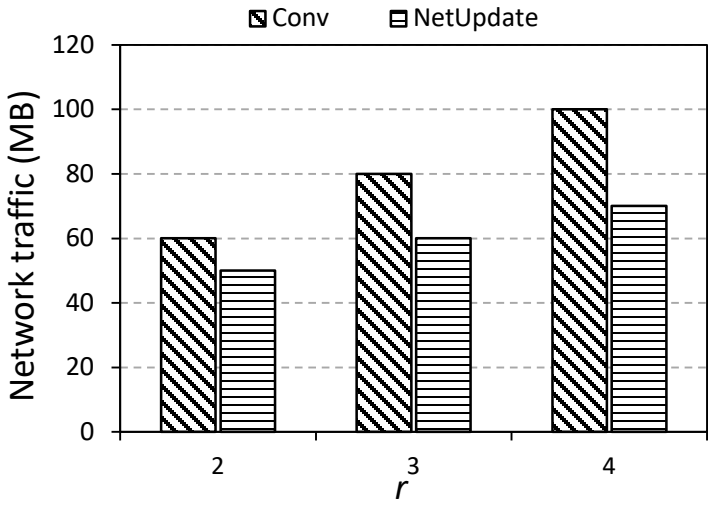
Update time is inversely proportional to the bottleneck bandwidth

Update time: Conv ≈ 2*NetUpdate

Coding parameter

- **Update performance of NetUpdate is independent of *r***
- **The larger the *r* value is, the more traffic that NetUpdate reduces**





Update time: NetUpdate always guarantees optimal performance

Network traffic: As r increases from 2 to 4, the network traffic reduction of NetUpdate increases from 17% to 30%

Thanks for listening
Q&A