Scalable and Reliable Data Broadcast with Kascade

Stéphane Martin, Tomasz Buchert, Pierric Willemet, Olivier Richard[(2)], Emmanuel Jeanvoine, Lucas Nussbaum

Algorille Team (Inria-Loria, F-54500, France)
[(2)]MESCAL Team (LIG, F-38000,France)

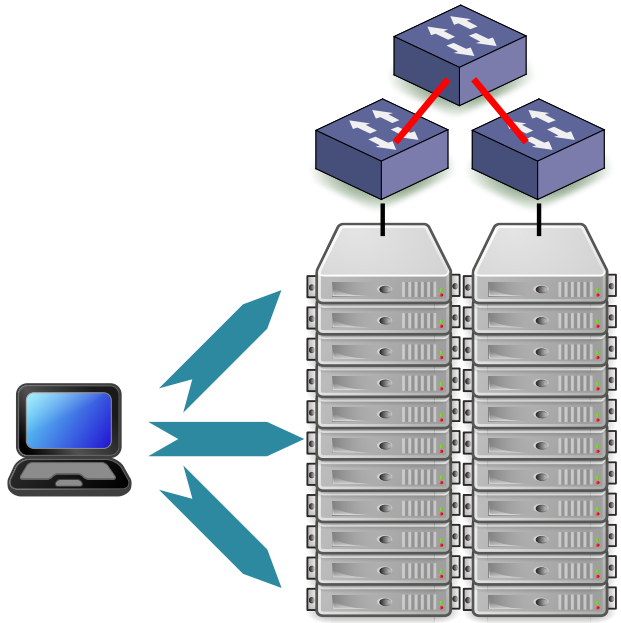
Context Big Data

Broadcast:

- Large amount of data
- From one storage
- To large number of nodes
- Fault tolerant

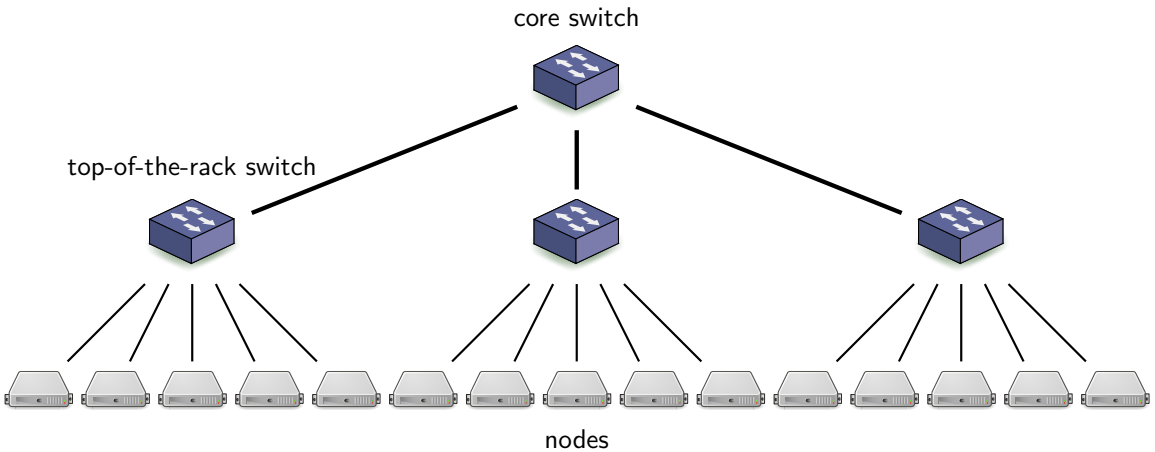
Useful to:

- Distribute big data before analysis
- System image deployment in HPC Cluster



Challenges

- Efficient use of fat tree network
  - Present in most cluster
  - Cost-efficient
- Fault-tolerance
  - One computer can have a problem
    $\rightarrow$ Many computers have problems
- Stream capability



Related work

- Network Layer multicast
- Binomial tree
- BitTorrent
- Pipelined Broadcast

Network Layer multicast

- IP multicast
  - Support is usually disabled in network equipment
  - Find a group address or method to share it
  - High-throughput protocol UDP is unfair to another protocols
  - Message delivering is not guaranteed !
    $\rightarrow$UDPCast provides:
    - Acknowledgment
      $\rightarrow$ Generate too many packets to the master, it's not scalable
    - Forward Error Correction (FEC)
      $\rightarrow$ Configuration depends of packets lost amount
- InfiniBand multicast
  - Not installed on all nodes (expensive)
  - Same problem

Conclusions

- Best in theory
- Not possible or efficient in practice

Protocols

Binomial tree

- Transfer random part of file (entire file is in memory or hard drive)

BitTorrent

- Verbose protocol
- Transfer random part of file

Conclusions

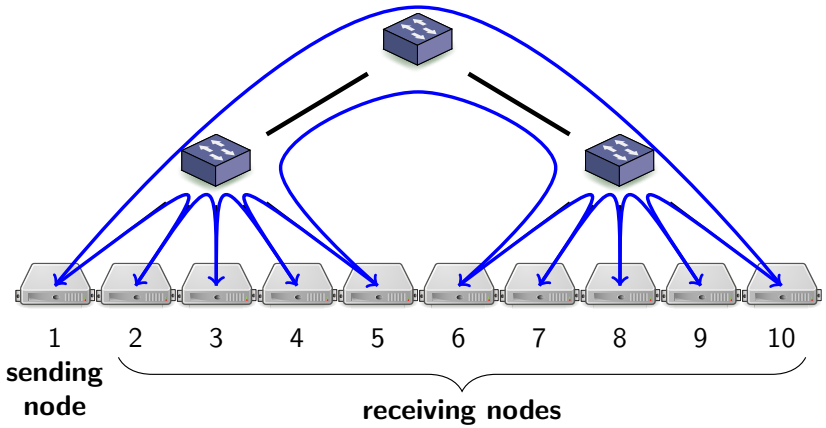
- Not suitable

Pipelined Broadcast

- One time on each direction
- Topology aware

Already existing projects:

- Not fault tolerant:
  - Ka
  - Dolly
  - part of MPI_Bcast primitive (Open MPI)
- Fault tolerant:
  - Dolly+
    $\rightarrow$ unmaintained and FT is not mentioned in their publication.



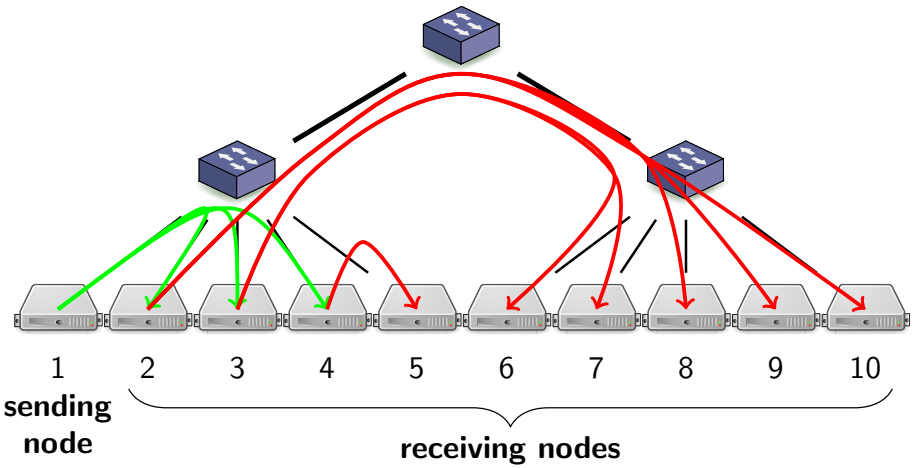
Our contributions: Kascade

How does it work ?

- Overview of pipeline establishment
- Fault detection
- Recovery
- Collecting information
- Protocol is needed

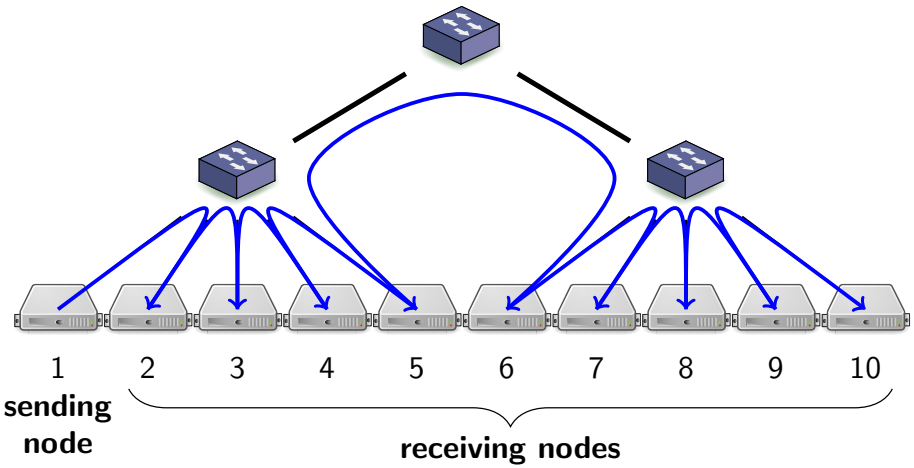
Overview of data transfer pipeline (Kascade)

1. Order the nodes
2. Deploy itself
   $\rightarrow$ Efficient help with Taktuk (Parallel launcher)
3. Establish the pipeline (open TCP/IP connections)
4. Transfer the data
5. Send report to the master



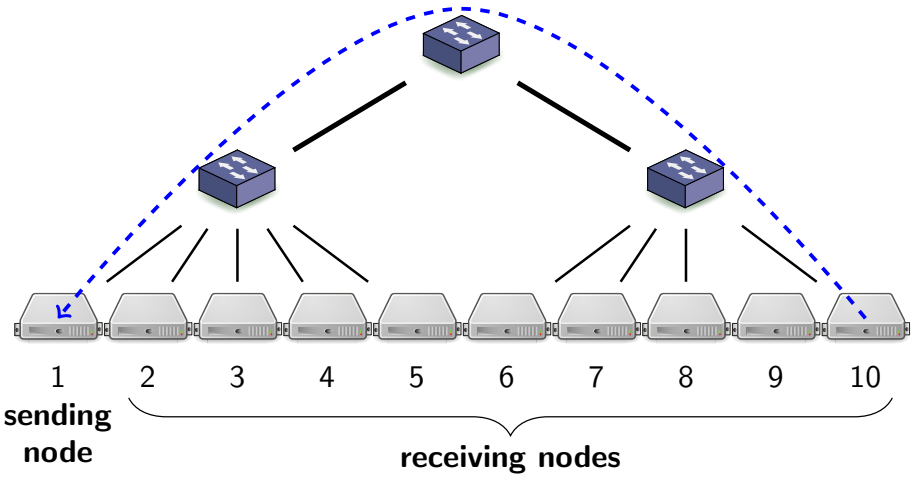
Overview of data transfer pipeline (Kascade)

1. Order the nodes
2. Deploy itself
   $\rightarrow$ Efficient help with Taktuk (Parallel launcher)
3. Establish the pipeline (open TCP/IP connections)
4. Transfer the data
5. Send report to the master



Overview of data transfer pipeline (Kascade)

1. Order the nodes
2. Deploy itself
   $\rightarrow$ Efficient help with Taktuk (Parallel launcher)
3. Establish the pipeline (open TCP/IP connections)
4. Transfer the data
5. Send report to the master



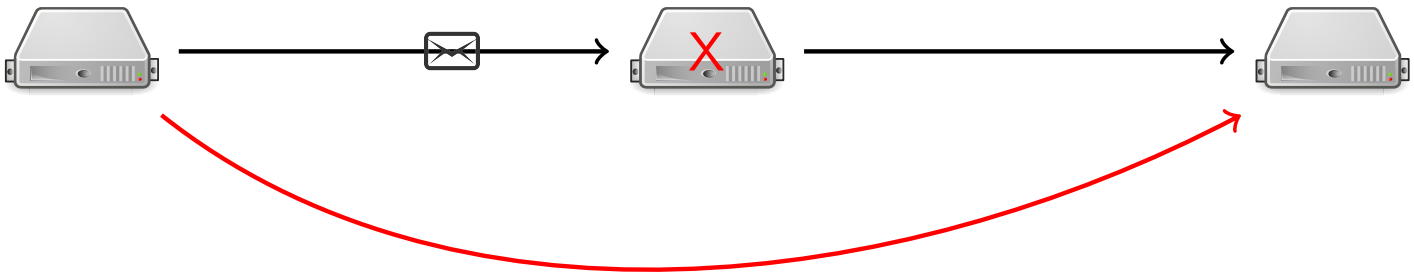
Fault detection

- Two case of faults:
  - Stream closed (error packet received)
  - Black hole (nothing comes back)

- The sender handles error
- When the next node stops to read the stream
  $\rightarrow$ ping the next node

Recovery

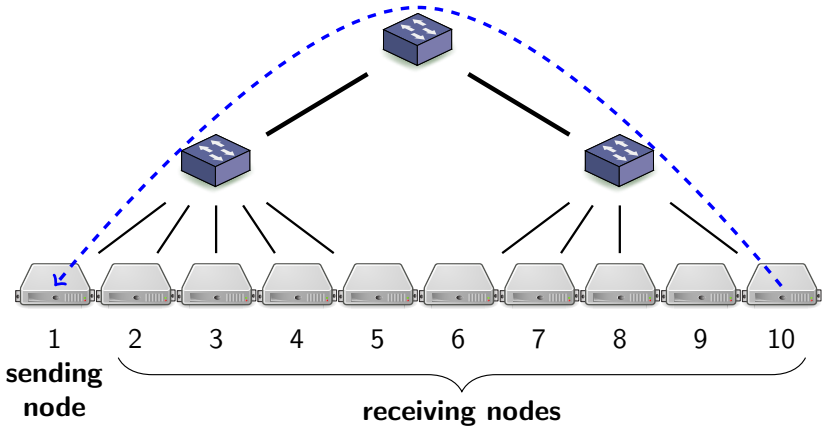
- Add error to the report
- Try to connect to the next node
- Replay the lost messages
  - Use buffer to resend lost message
  - Ask the missing part to the master in worst case



Collecting status at end of transfer

- Connecting to the master directly is not scalable or fault tolerant
- Using the pipeline to transmit a report is scalable and fault tolerant

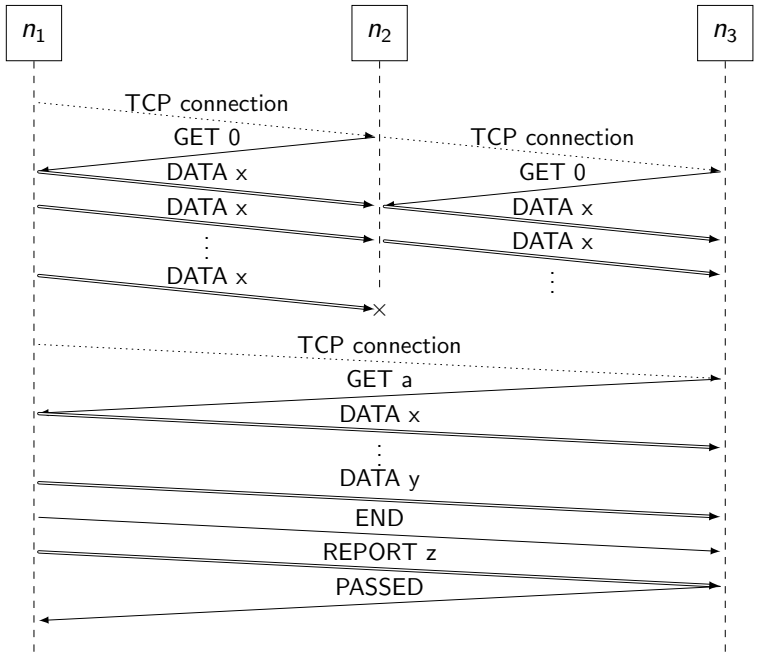
- The last node forwards the report to the master
- The report reception implies the end of transfer



Protocol is needed

The protocol

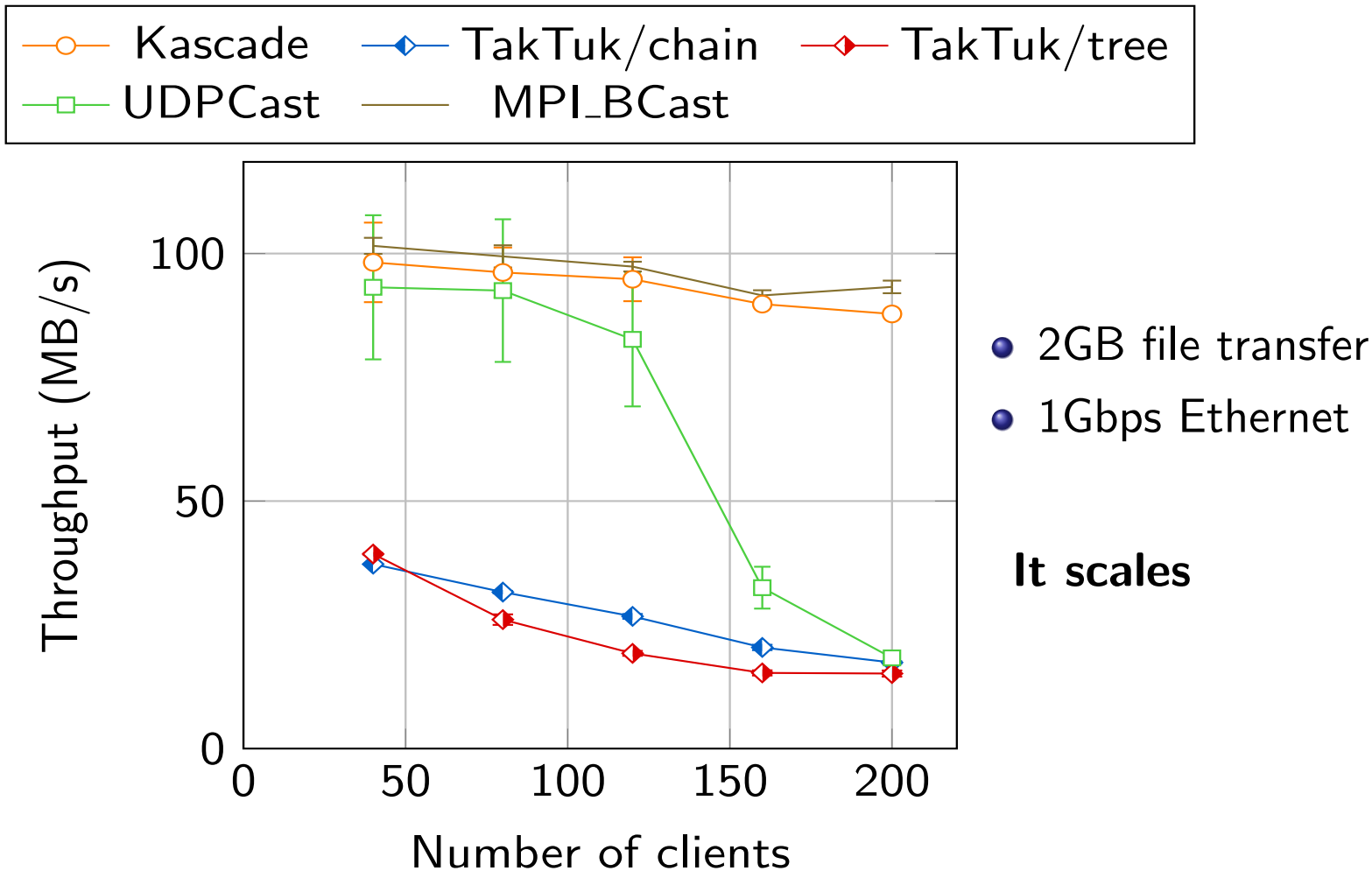
- avoids the data size knowledge (stream capability)
- permits prematurely end requested by user
- distinguishes the report than data
- improves the fault tolerance (correct ending despite failures)



Validation questions

- How do the various solutions perform and scale up to large number of nodes?
- How does Kascade perform on high-performance networks (10 Gbps Ethernet, IP over InfiniBand)?
- What is the impact of network topology and communication structure on performance?
- What it the impact of I/O performance on the overall performance?
- How does Kascade perform on large-scale (Internet-like) setups?
- How does Kascade perform on smaller files?
- How well does Kascade's fault tolerance mechanism perform?

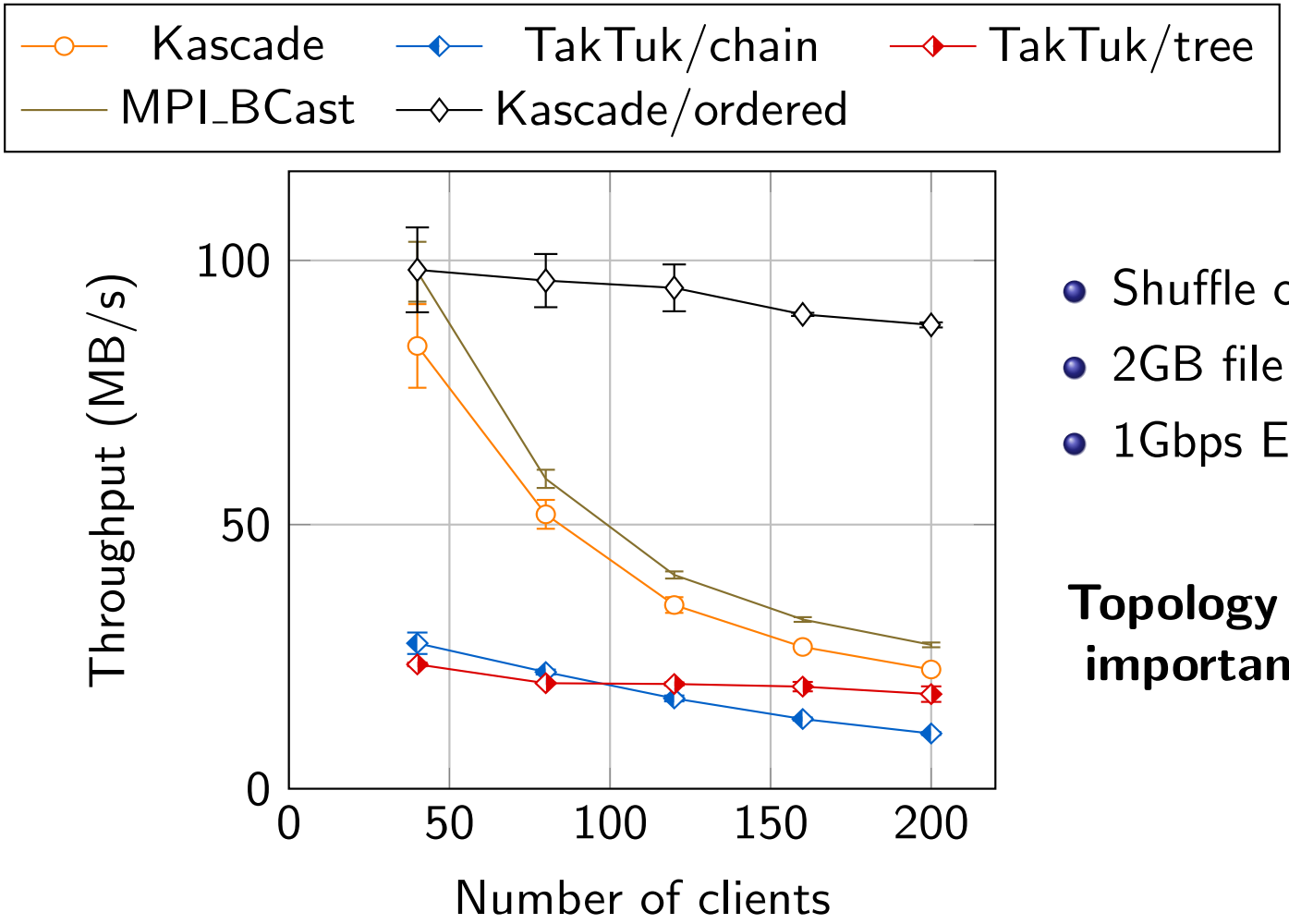
How do the various solutions perform and scale up to large number of nodes ?



- 2GB file transfer
- 1Gbps Ethernet

It scales

What is the impact of network topology and communication structure on performance ?



- Shuffle order of nodes
- 2GB file transfer
- 1Gbps Ethernet

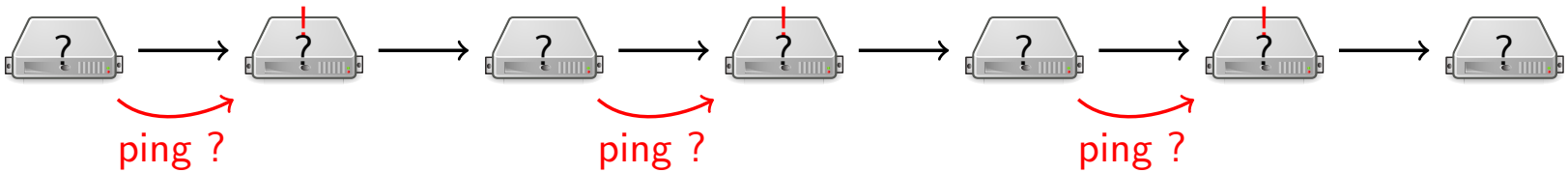
Topology awareness is important

How well does Kascade's fault tolerance mechanism perform?

Simultaneous

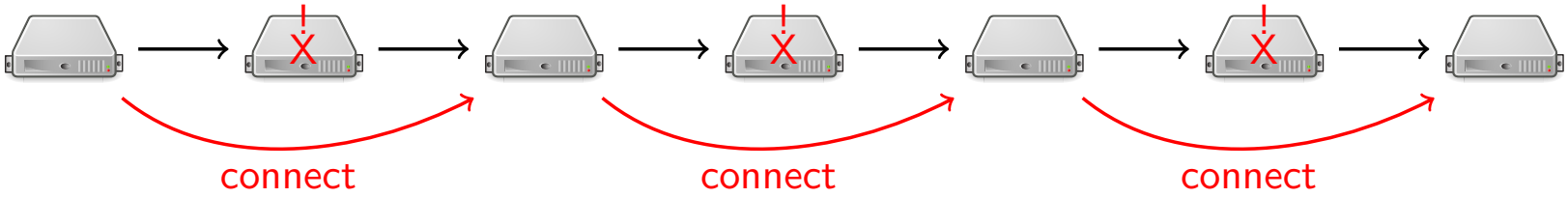
How well does Kascade's fault tolerance mechanism perform?

Simultaneous



How well does Kascade's fault tolerance mechanism perform?

Simultaneous

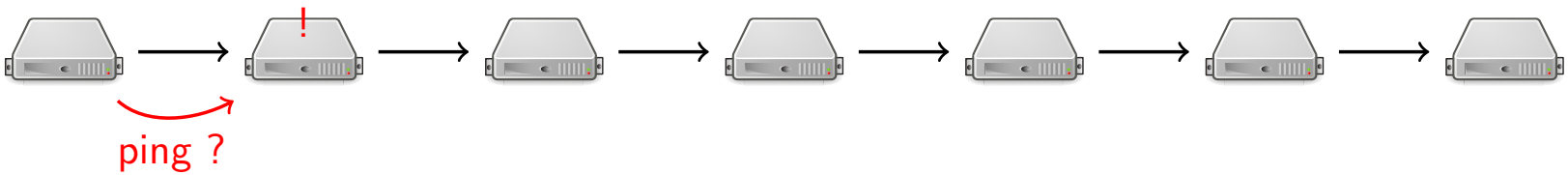


How well does Kascade's fault tolerance mechanism perform?

Sequencial

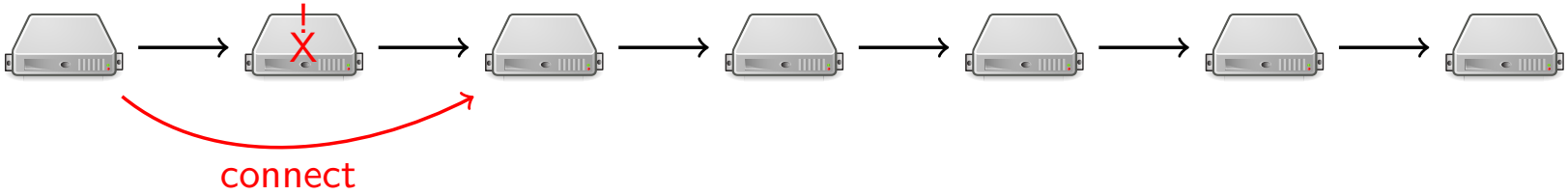
How well does Kascade's fault tolerance mechanism perform?

Sequencial



How well does Kascade's fault tolerance mechanism perform?

Sequencial

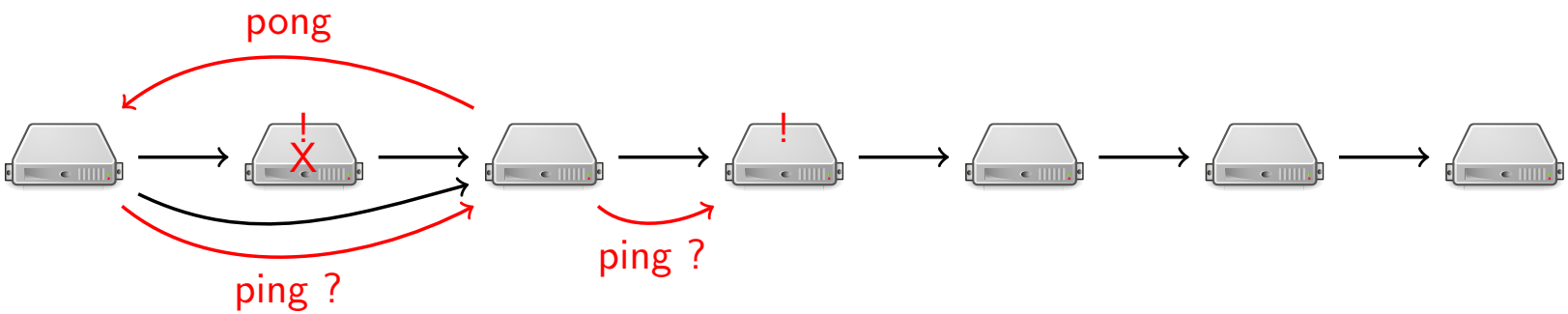


How well does Kascade's fault tolerance mechanism perform?

Sequencial

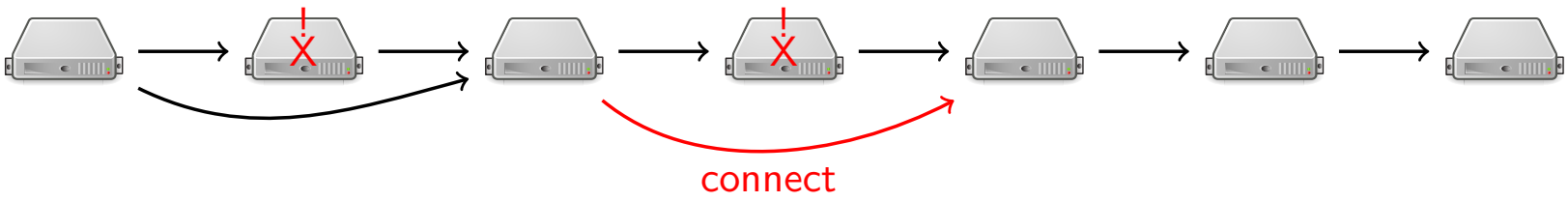
How well does Kascade's fault tolerance mechanism perform?

Sequencial



How well does Kascade's fault tolerance mechanism perform?

Sequencial

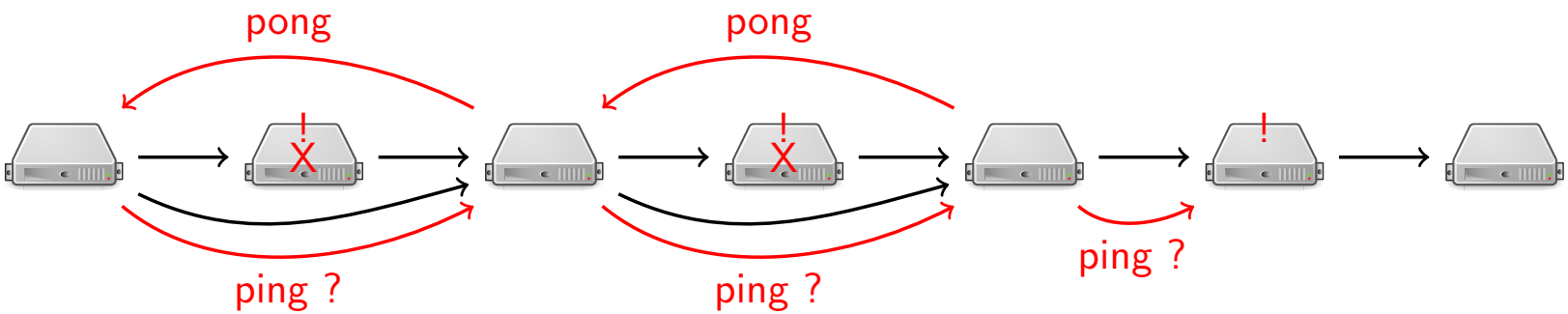


How well does Kascade's fault tolerance mechanism perform?

Sequencial

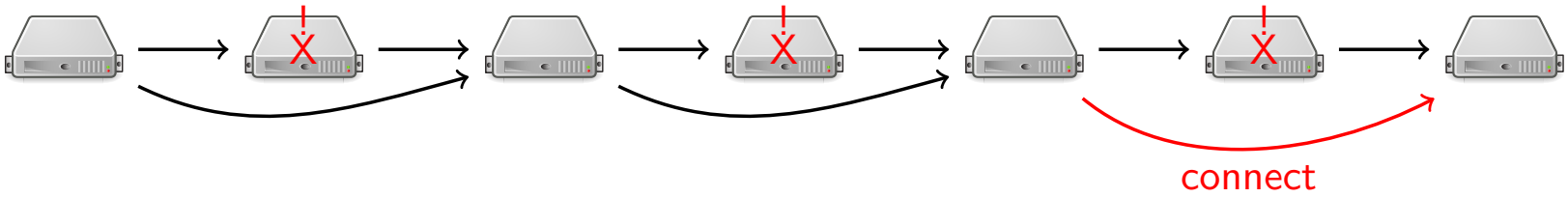
How well does Kascade's fault tolerance mechanism perform?

Sequencial

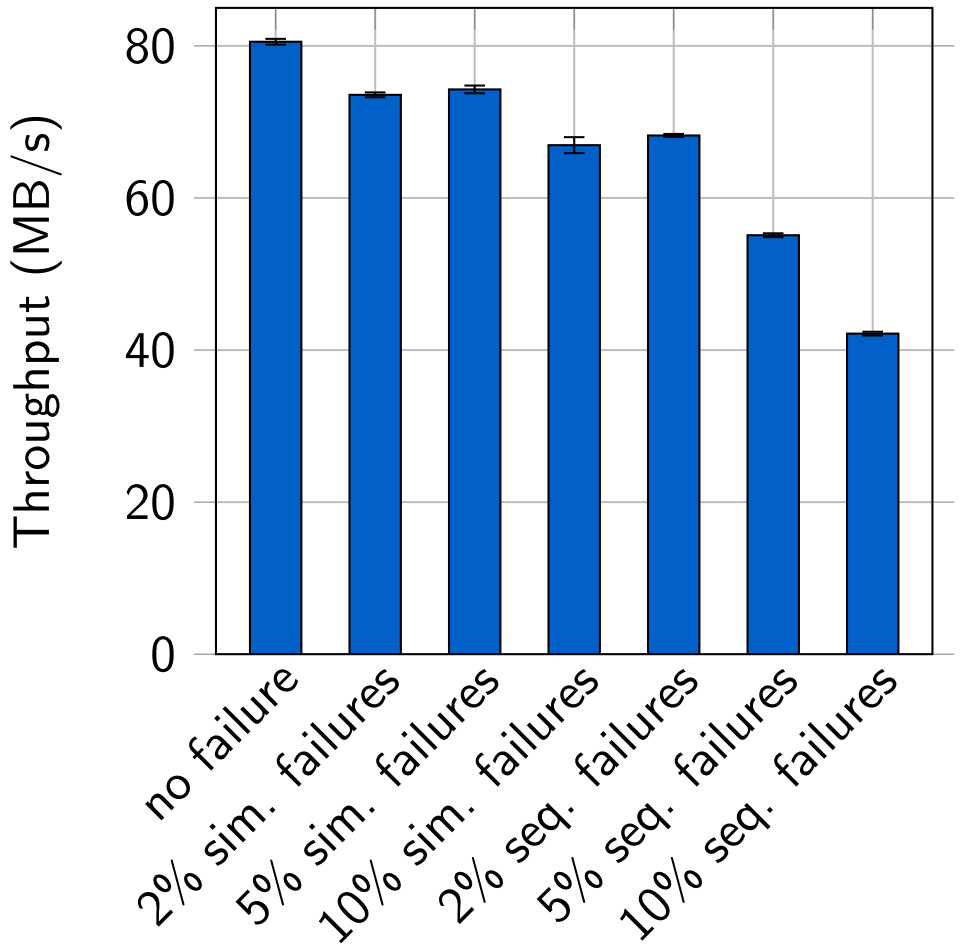


How well does Kascade's fault tolerance mechanism perform?

Sequencial



How well does Kascade's fault tolerance mechanism perform?



- 100 virtual nodes, fault injected with Distem
- 5GB file transfer
- 1Gbps Ethernet

Fault tolerant

Conclusions

- Efficient use of fat tree network
  - Saturate the 1Gbps network
- Fault-tolerance
  - Mechanisms works
  - Performance is acceptable in hostile environment
- Stream capability

Future work:

- Slow node elimination
  $\rightarrow$ when a node in the pipeline slows down the transfer rate because of a malfunction