Hurricane

Master semester project – IC School
Operating Systems Laboratory

Author

Diego Antognini

Supervisors

Prof. Willy Zwaenepoel

Laurent Bindschaedler

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Outline

- Motivation
- Hurricane
- Experiments
- Future work
- Conclusion

Motivation

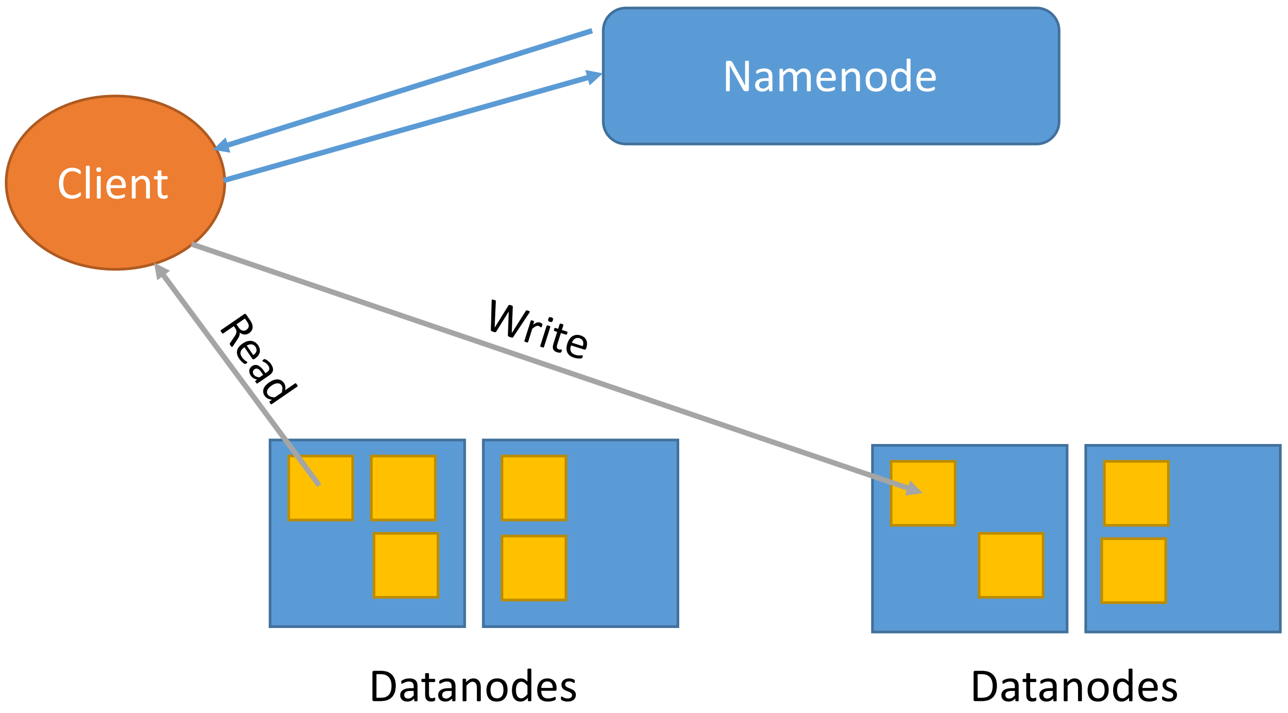
Original goal of the project

- Implement Chaos on top of HDFS !
- How ?
  - Replace storage engine by HDFS
- Why ?
  - Industry interested by systems running on Hadoop
  - Handling cluster easily
  - Distributed file systems
  - Fault-tolerance (but at what price ?)

Chaos

- Scale-out graph processing from secondary storage
  - Maximize sequential access
- Stripes data across secondary devices in a cluster
- Limited only by :
  - aggregate bandwidth
  - capacity of all storage devices in the entire cluster

Hadoop Distributed File System



Experiment : DFSIO

- Measure aggregate bandwidth on a cluster when writing & reading 100 GB of data in X files :

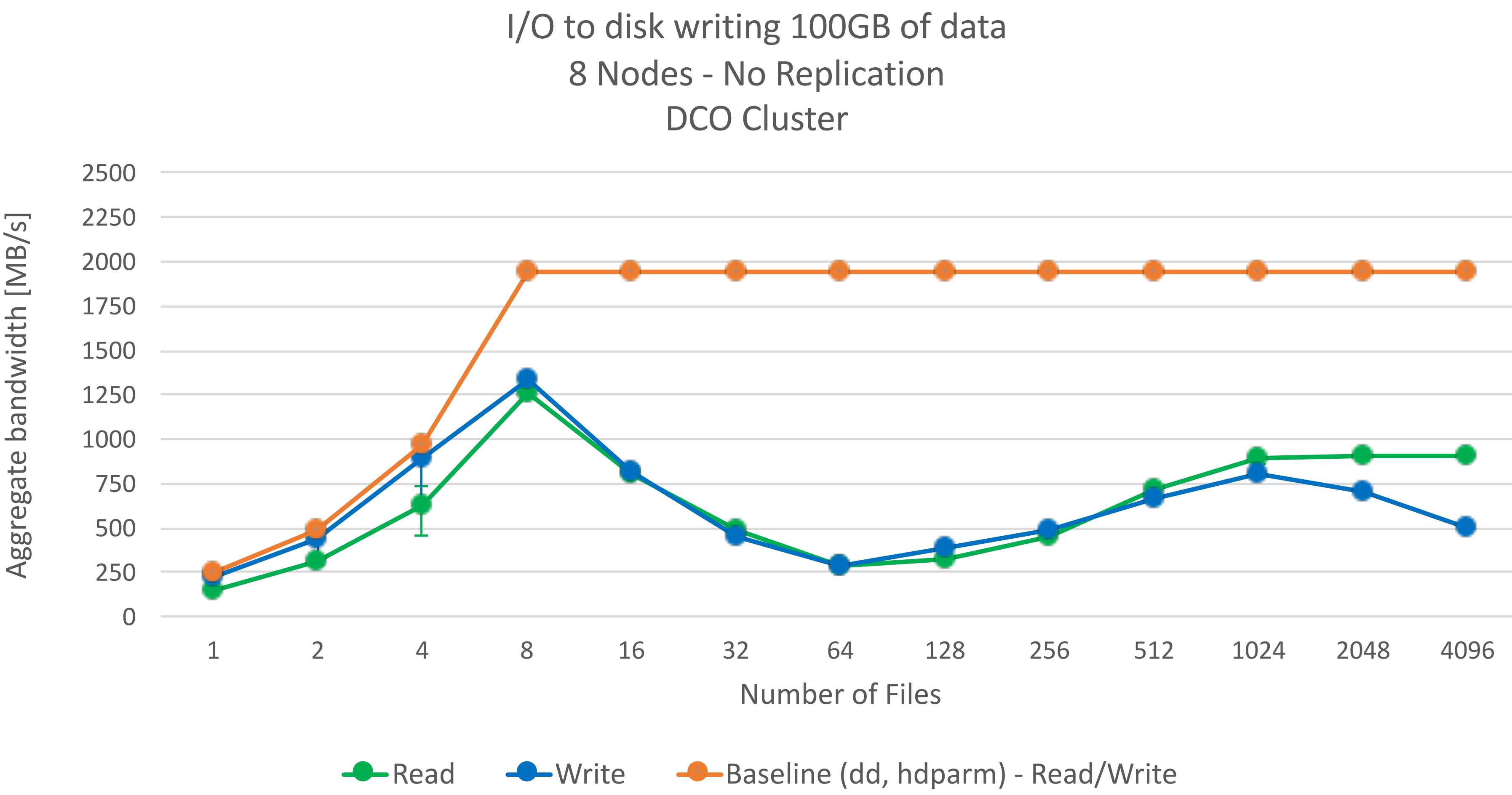
# Files	Size
1	100 GB
2	50 GB
…	…
4096	25 MB

- Use DFSIO benchmark
  - Each task operates on a distinct block
  - Measure disk I/O

Clusters

	DCO
OS	Ubuntu 14.04.01
# Cores	16
Memory	128 GB
Storage	HDD : 140 MB/s SSD : 243 MB/s
Network	10 Gbit/s

Results DFSIO – DCO cluster



Observations: DFSIO

- Somewhat lackluster performance
- Hard to tune !

HDFS doesn’t fit the requirements

Our solution

- Create a standalone distributed storage system based on Chaos storage engine
- Give it an HDFS-like RPC interface

Actual project !

Hurricane

Hurricane

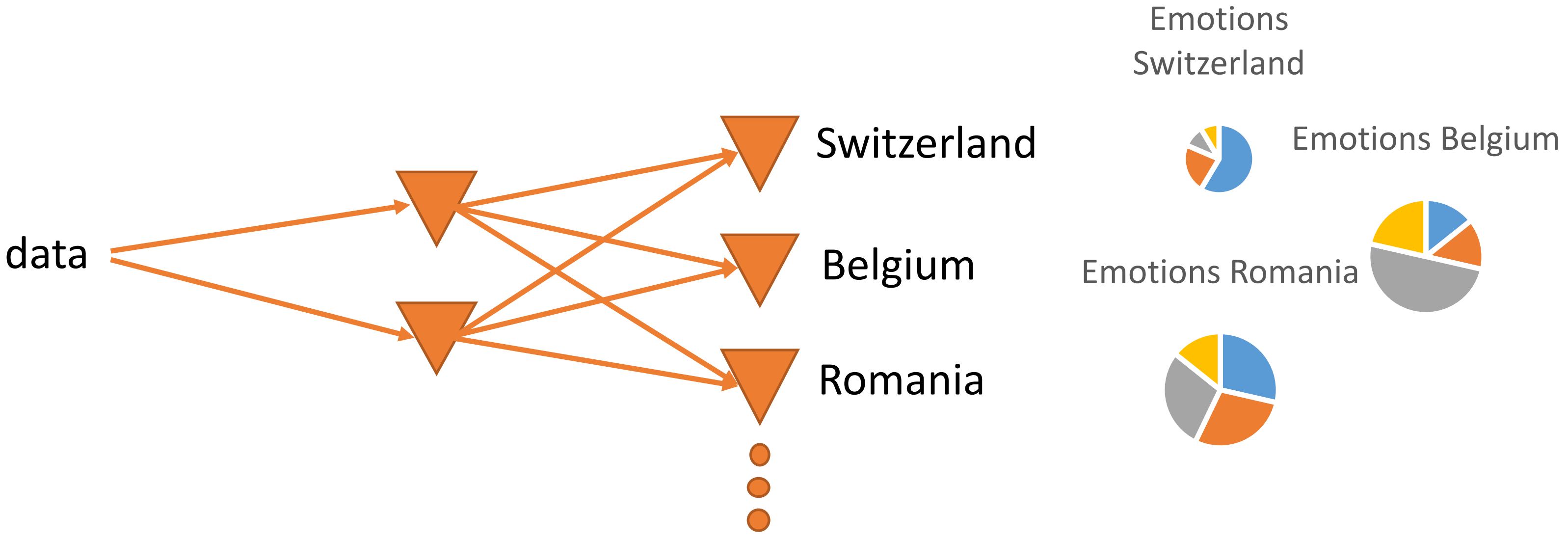
- Scalable decentralized storage system based on Chaos
- Balance I/O load randomly across available disks
- Saturate available storage bandwidth
- Target rack-scale deployment

Real life scenario

- Chaos using Hurricane

Real life scenario

- Measuring emotions of countries during Euro 2016



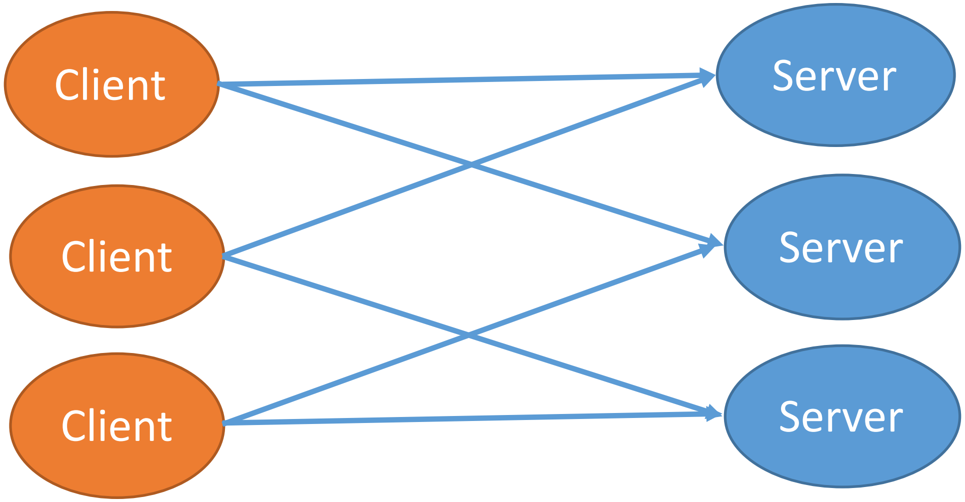
- And much more !

Locality does not matter !

- Remote storage bandwidth = local storage bandwidth
  - Clients can read/write to any storage device
- Storage is slower than network
  - Network not a bottleneck !
- Realistic for most clusters at rack scale or even more

Maximizing I/O bandwidth

- Clients pull data records from servers

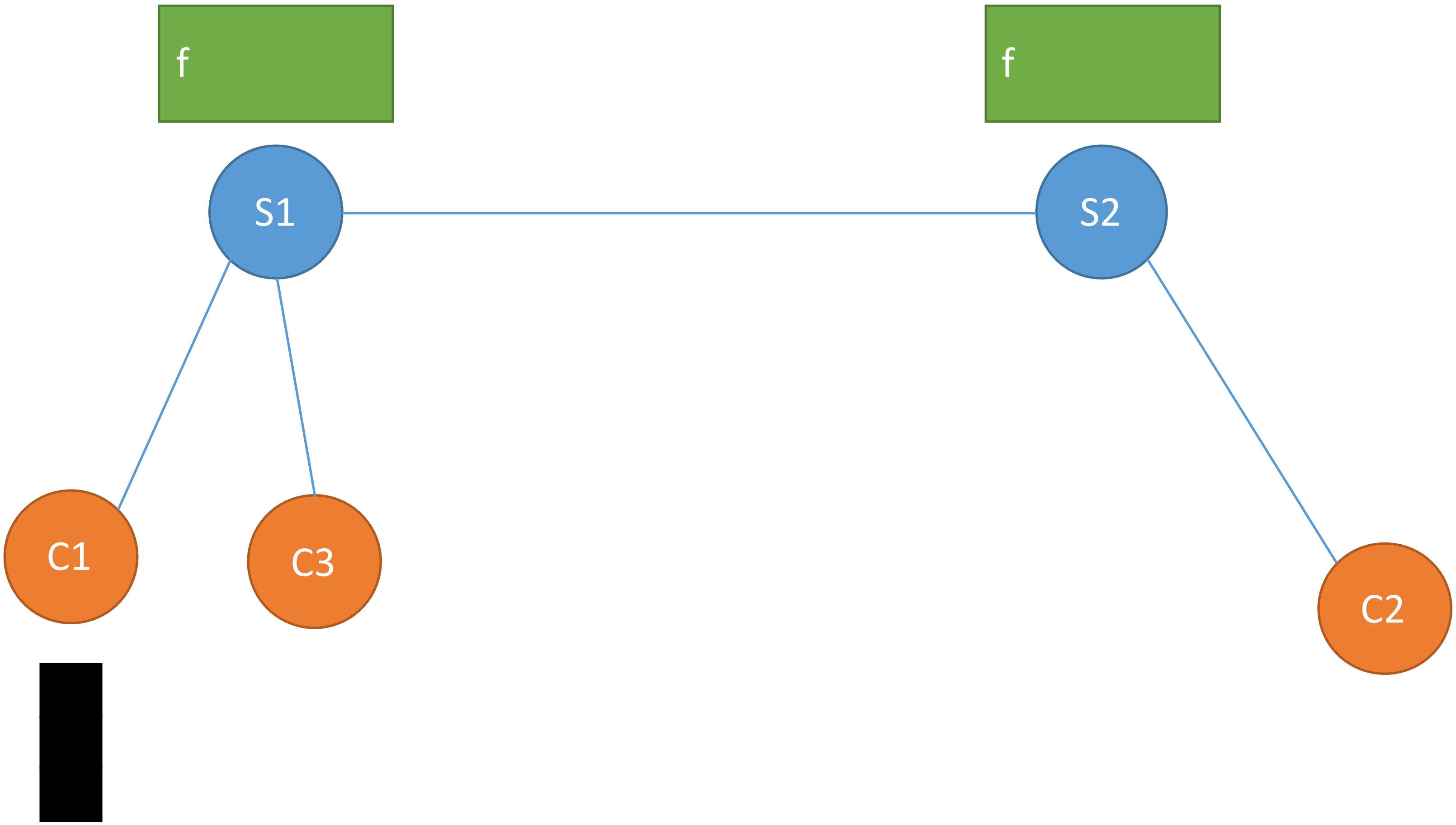


- Batches requests to prevent idle servers (prefetching)

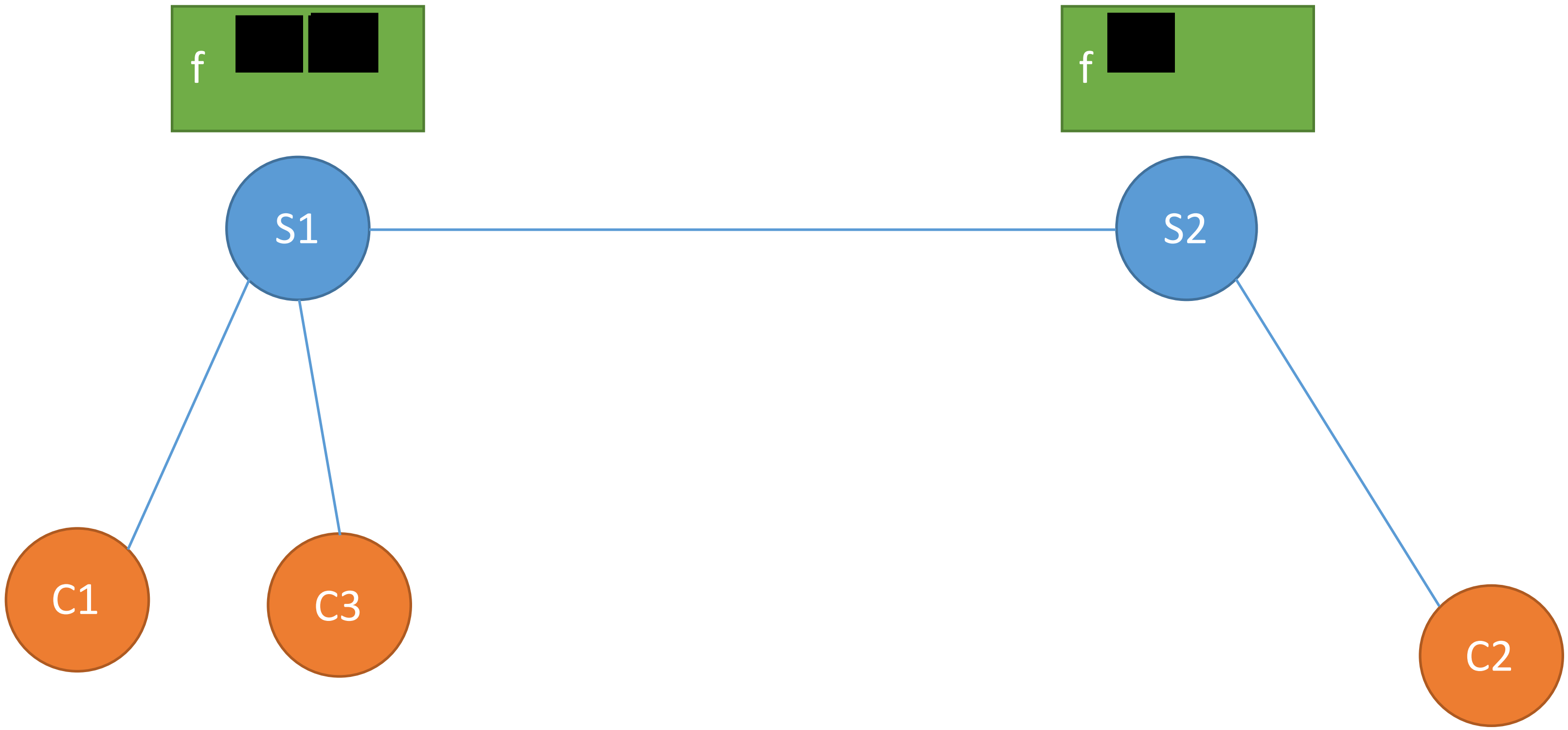
Features

- Global file handling (global_*)
  - Create, exists, delete, fill, drain, rewind etc …
- Local file handling (local_*)
  - Create, exists, delete, fill, drain, rewind etc ...
- Add storage nodes dynamically

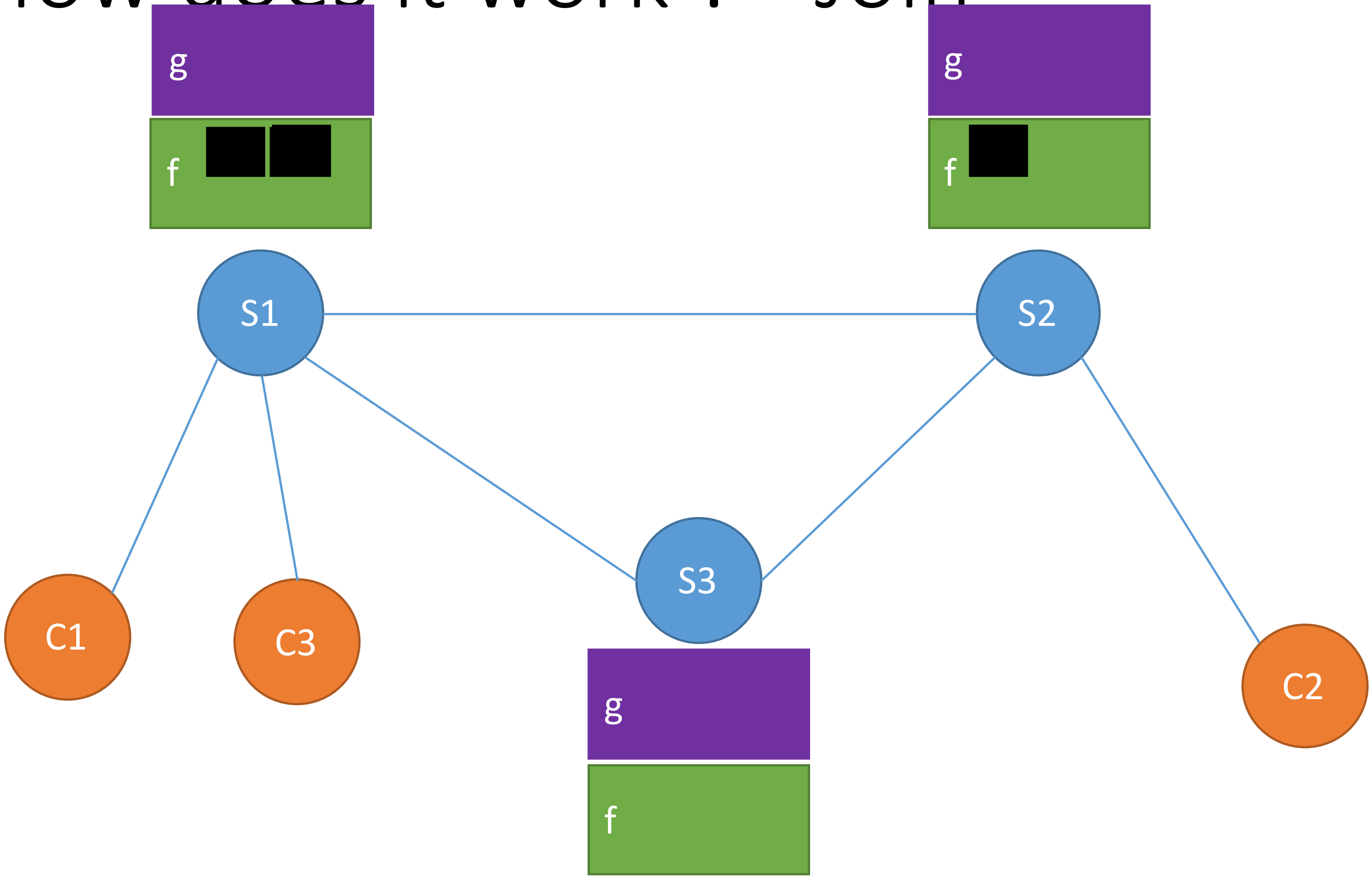
How does it work ? – Writing files



How does it work ? – Reading files



How does it work ? - Join



Experiments

Clusters

	LABOS	DCO	TREX
OS	Ubuntu 14.04.1	Ubuntu 14.04.01	Ubuntu 14.04.01
# Cores	32	16	32
Memory	32 GB	128 GB	128 Gb
Storage	HDD : 474 MB/s	HDD : 140 MB/s SSD : 243 MB/s	HDD : 414 MB/s SSD : 464 MB/s
Network	1 Gbit/s	10 Gbit/s	40 Gbit/s

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

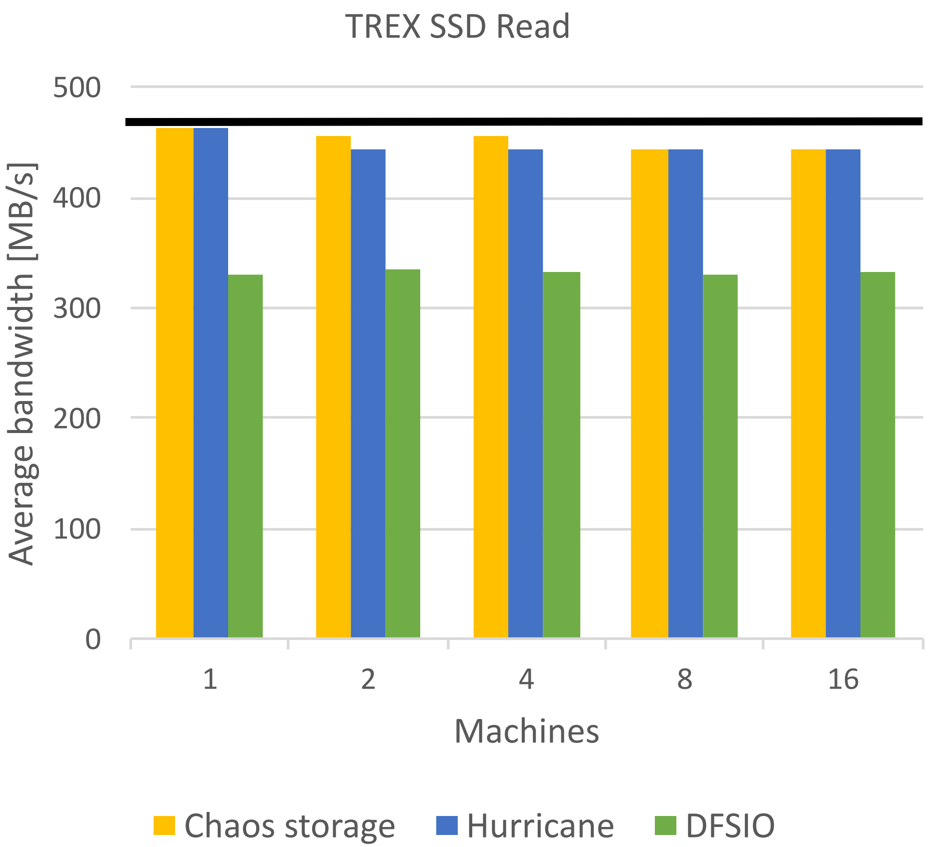
List of experiments

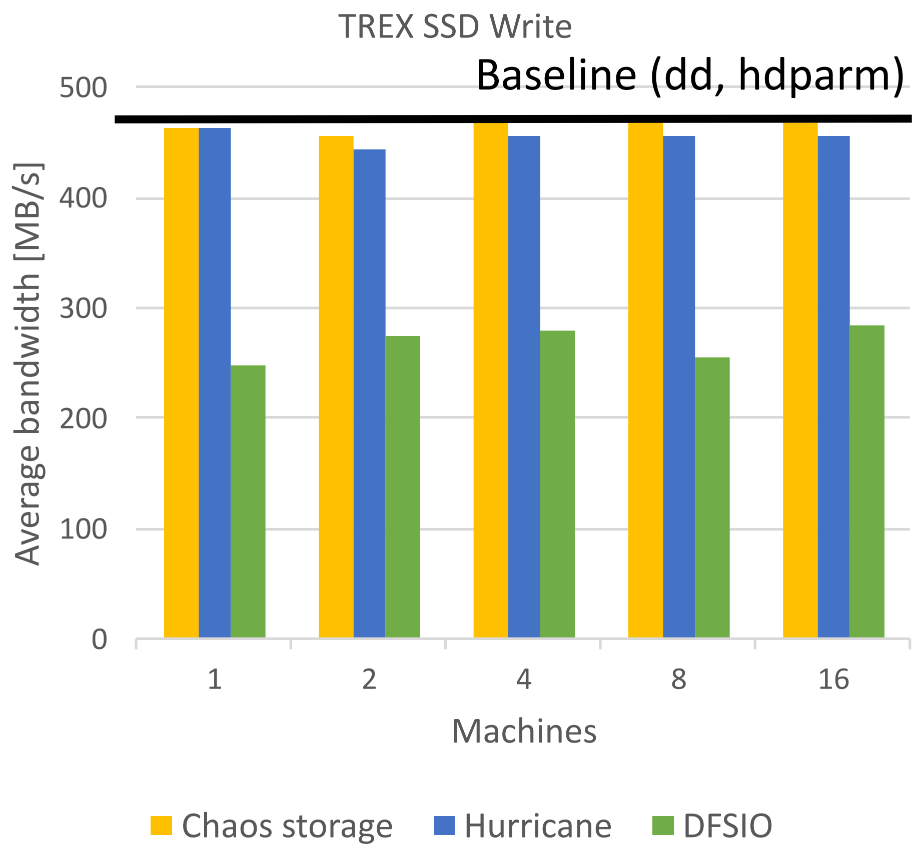
- Weak scaling
- Scalability 1 client
- Strong scaling
- Case studies
  - Unbounded buffer
  - Compression

Weak scaling

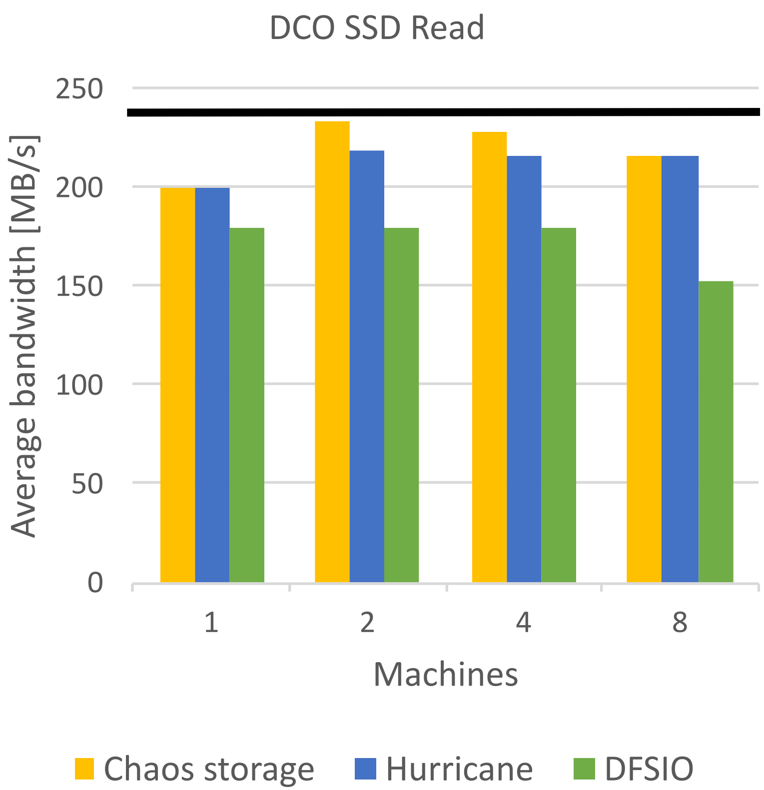
- Each node writes/reads 16 GB of data
- Increasing number of nodes
- N servers, N clients
- Measure average bandwidth
- Compare Chaos storage engine, Hurricane, DFSIO

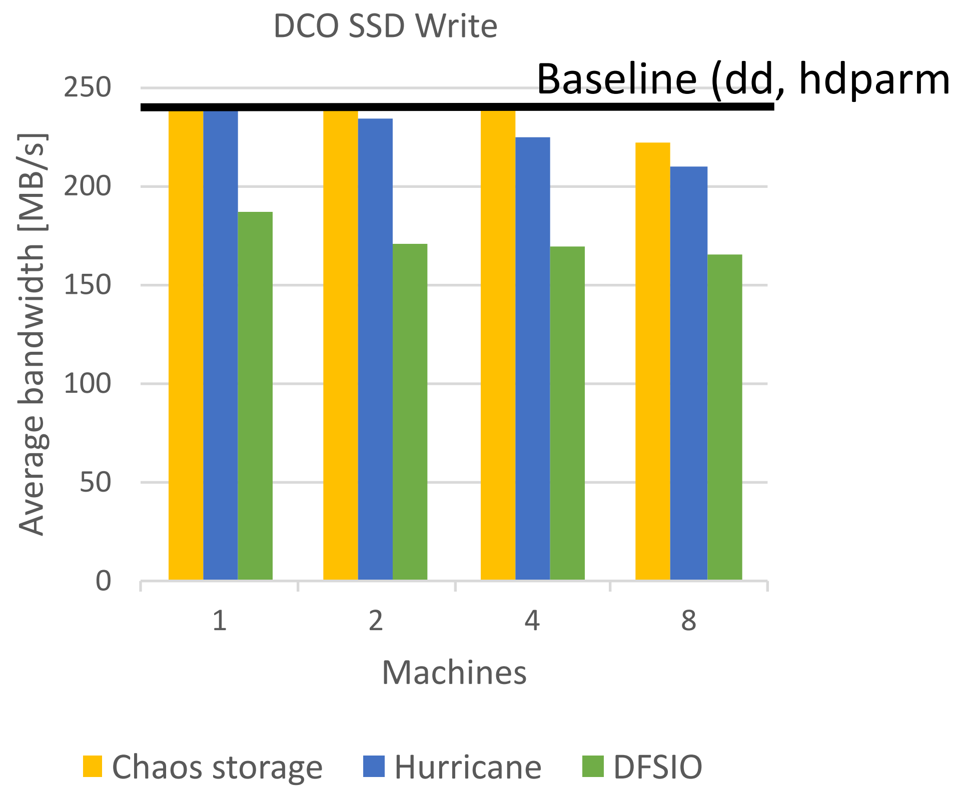
16 GB per node – 40 Gbit/s network



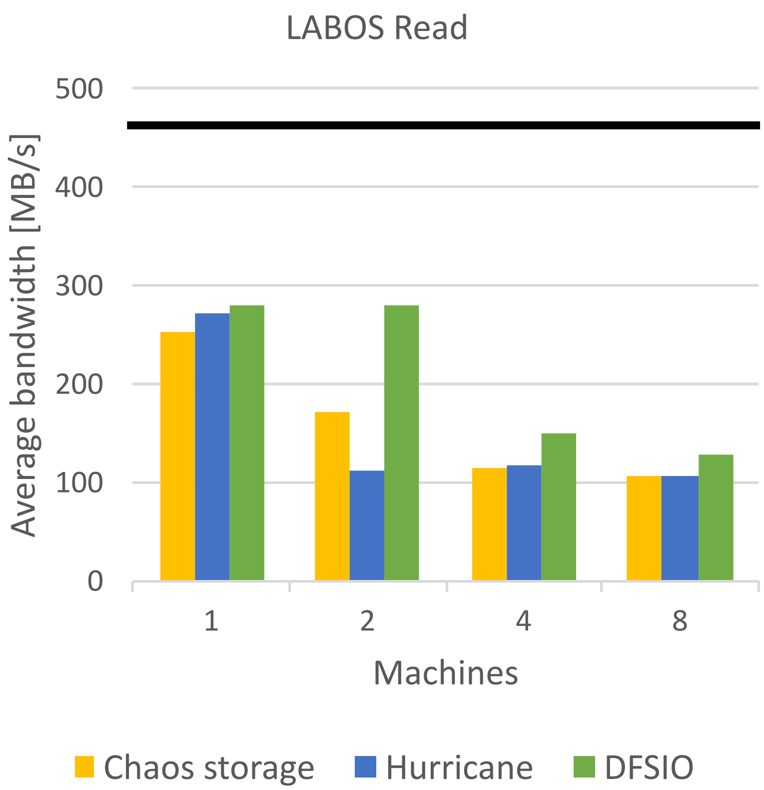


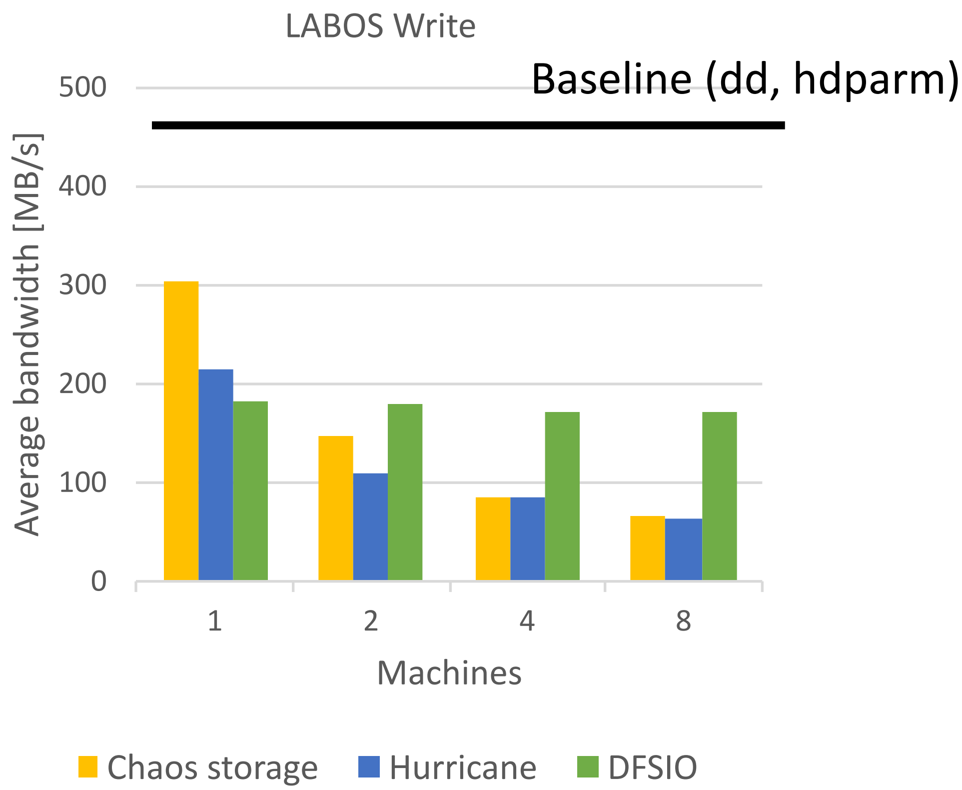
16 GB per node – 10 Gbit/s network





16 GB per node – 1 Gbit/s network

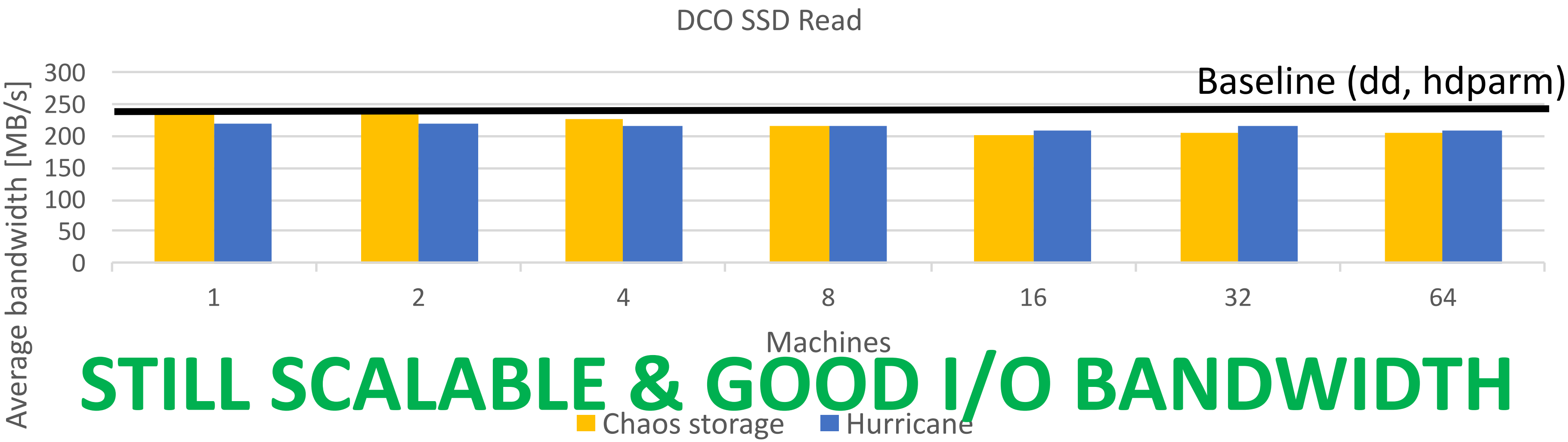




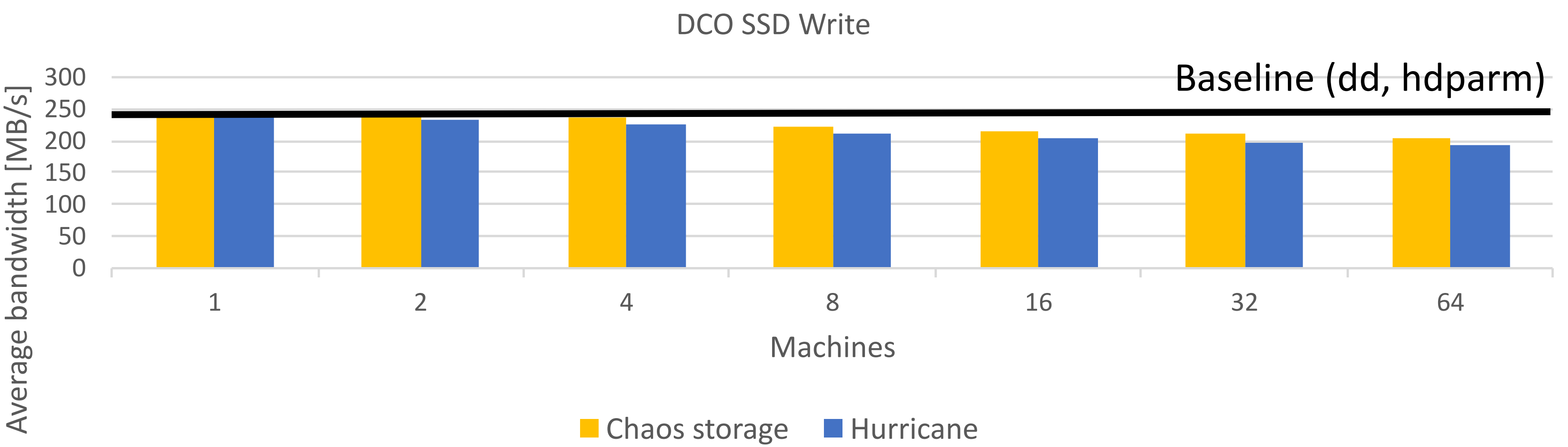
Weak scaling - Summary

- Hurricane similar performance with Chaos storage
- Scalable
- Outperforms HDFS roughly 1.5x
- Maximize I/O bandwidth

16 GB per node - 64 nodes



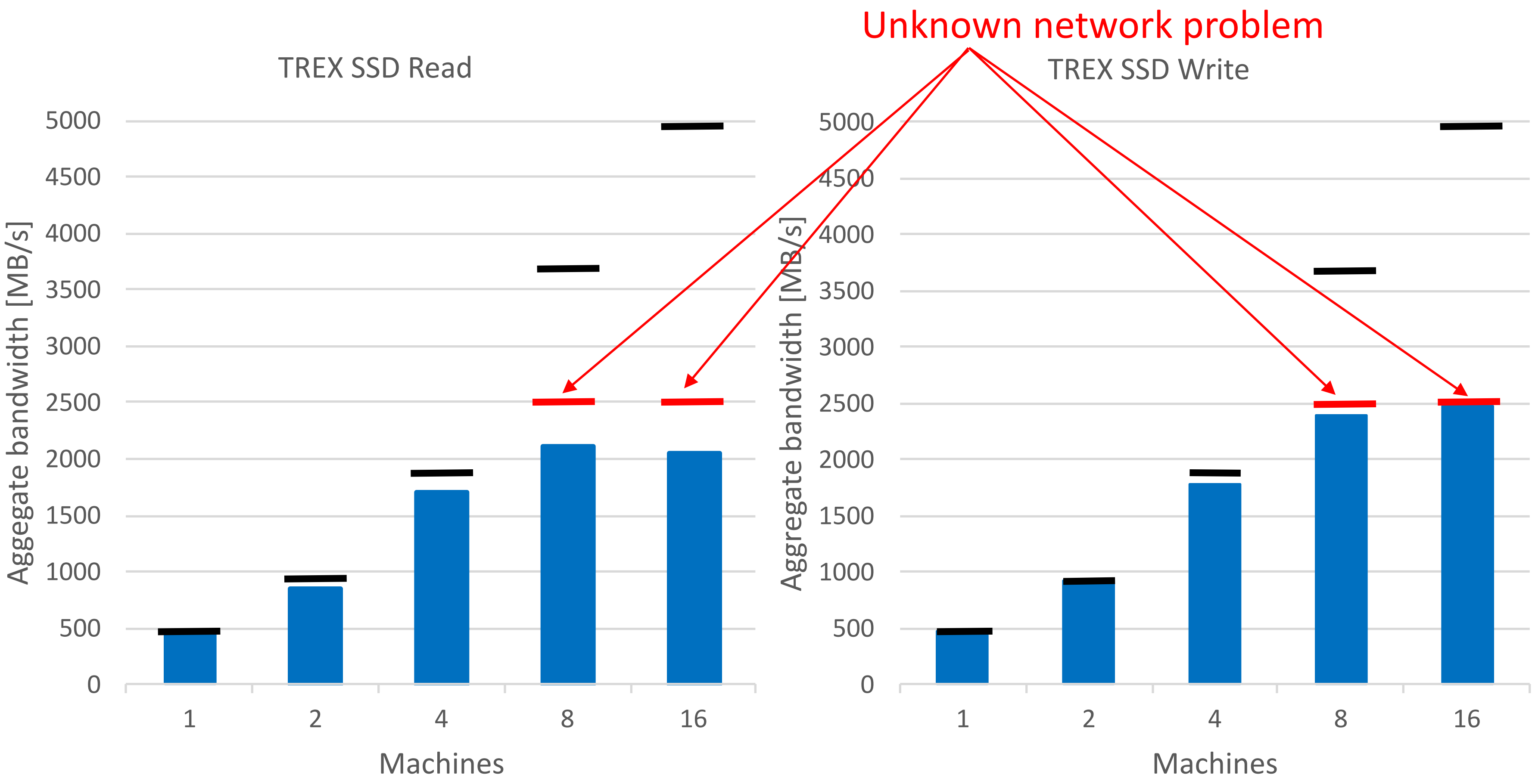
STILL SCALABLE & GOOD I/O BANDWIDTH



Scalability with 1 Client

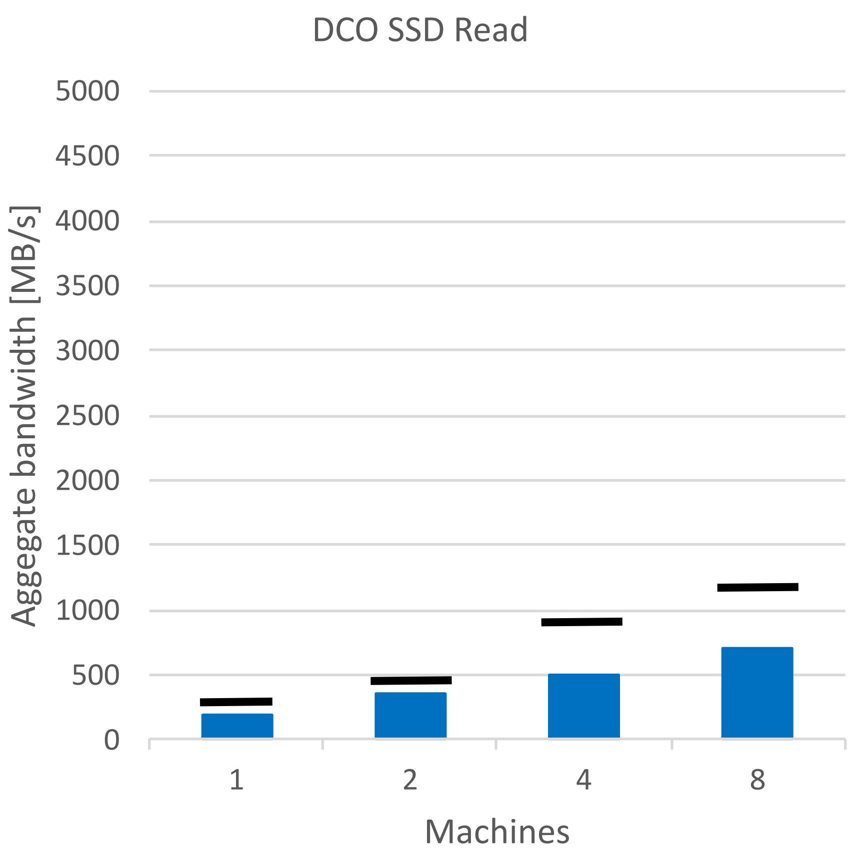
- Client writes/reads 16 GB of data per server node
- Increasing number of server nodes
- N servers, 1 client
- Measure aggregate bandwidth
- Only Hurricane is used

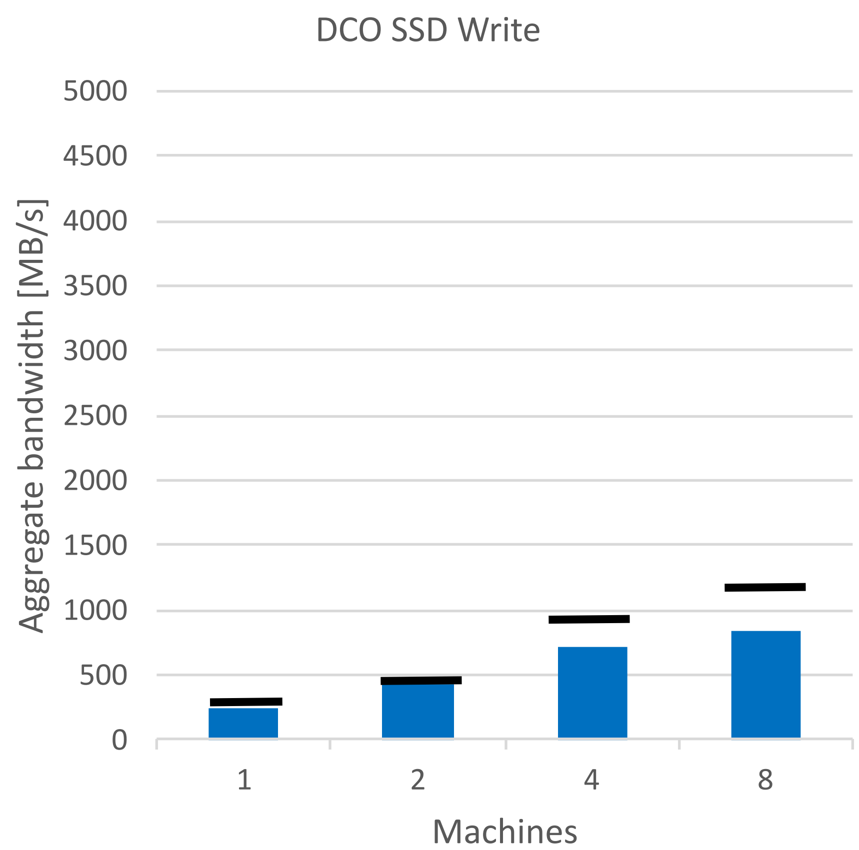
40 Gbit/s network



- Baseline
- Actual bandwidth of the network

10 Gbit/s network





Baseline

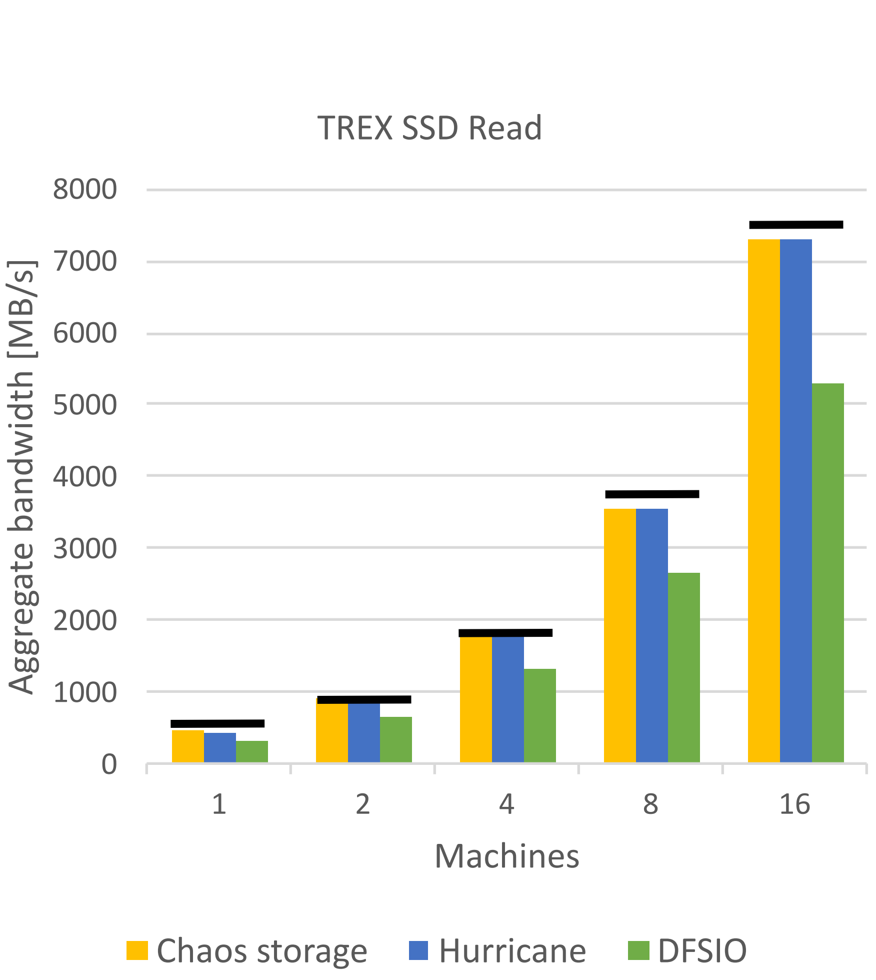
Also scale with only 1 client

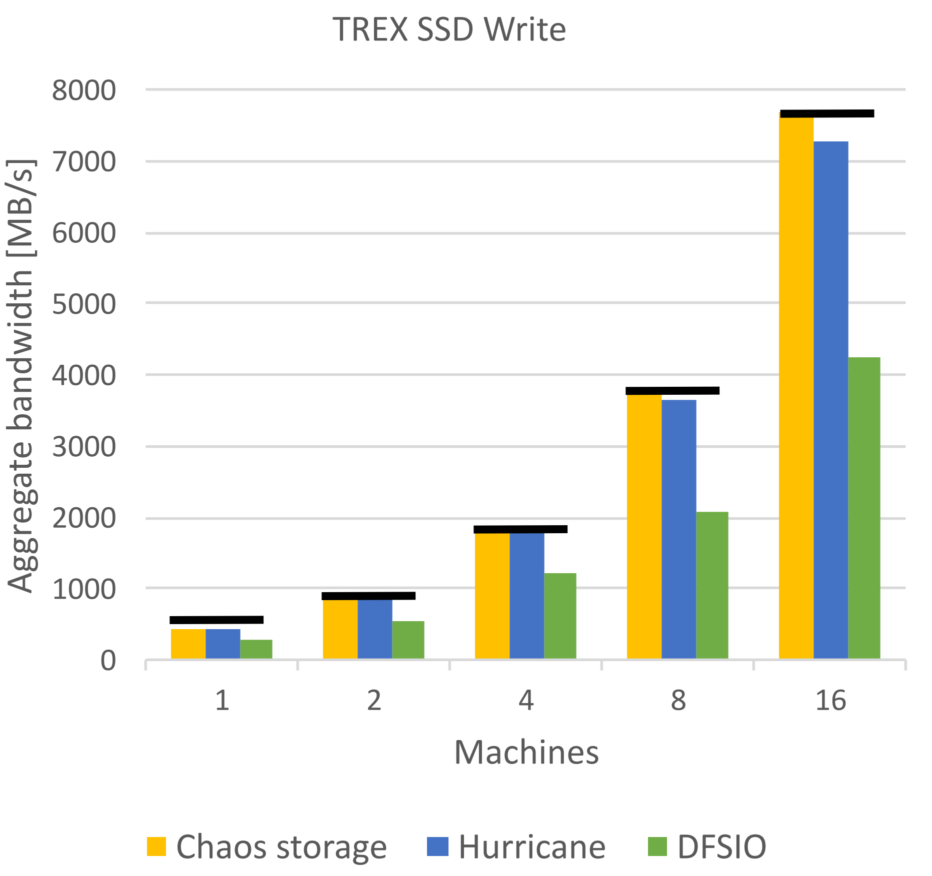
Use the I/O bandwidth of all the server nodes

Strong scaling

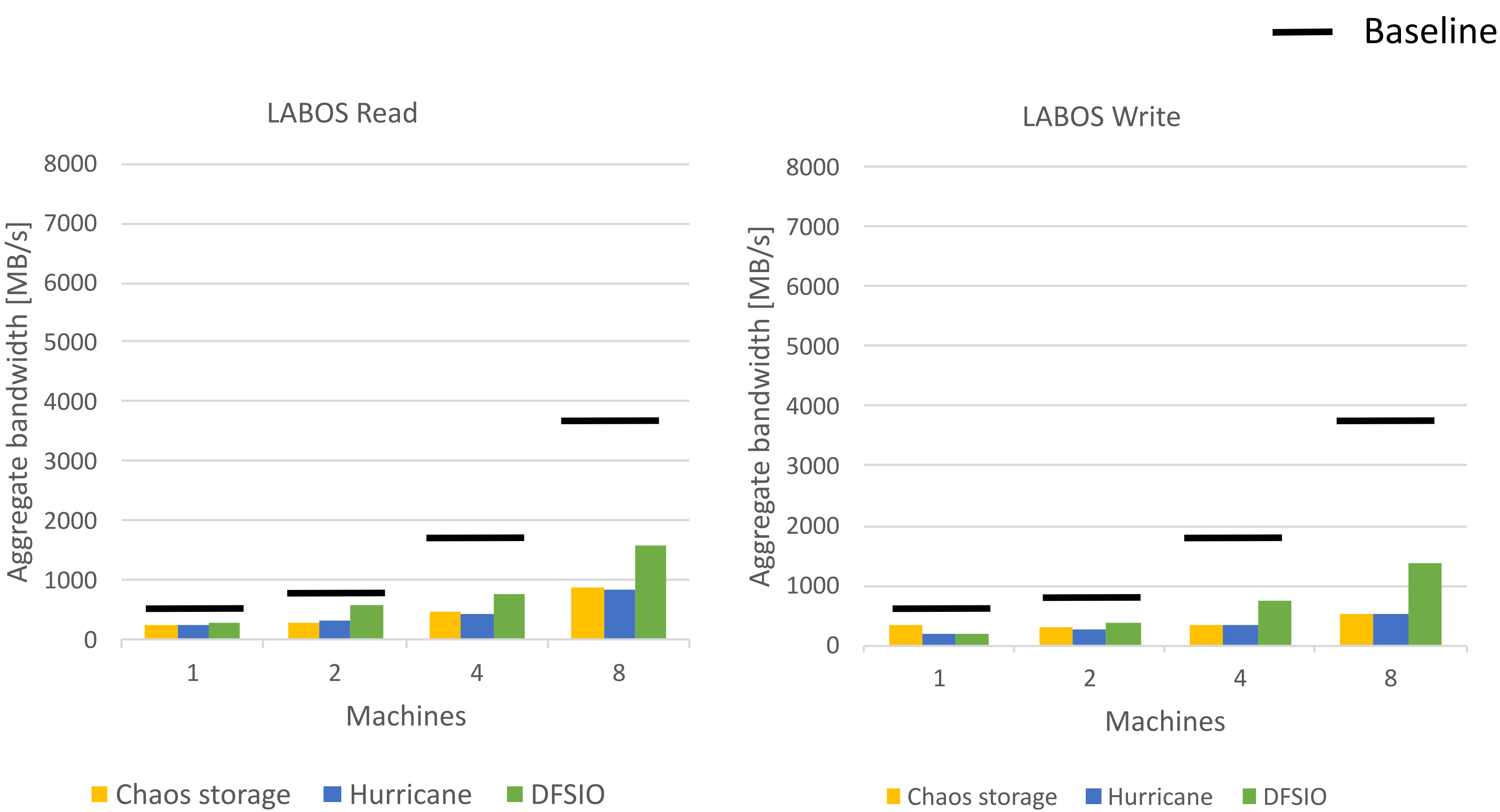
- Read/write 128 GB of data in total
- Increasing number of nodes
- N servers, N clients
- Measure aggregate bandwidth
- Compare Chaos storage engine, Hurricane, DFSIO

40 Gbit/s network





1 Gbit/s network



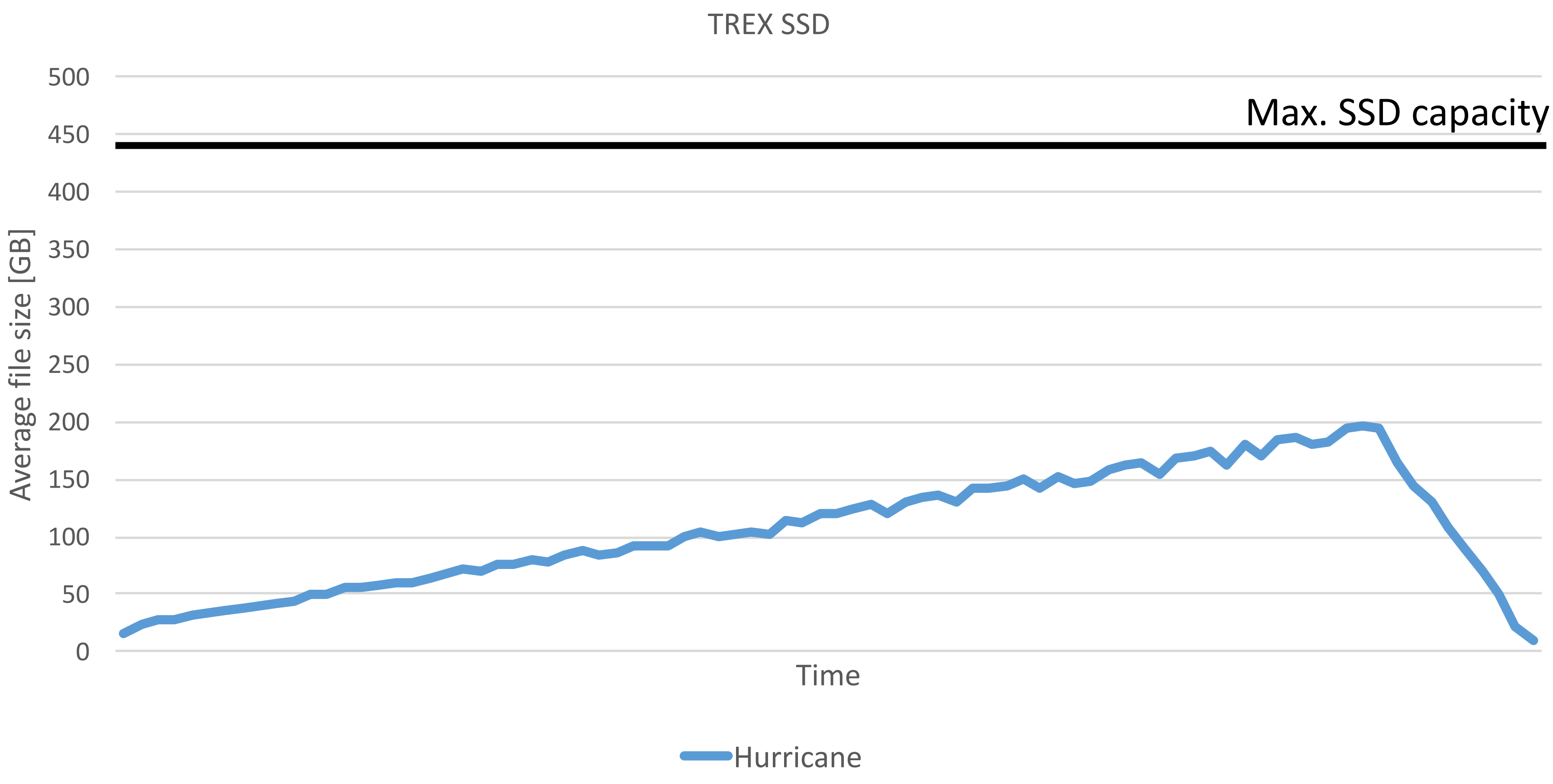
Strong scaling - Summary

- Hurricane similar performance with Chaos storage
- Scalable
- Outperforms HDFS roughly x1.5
- Maximize I/O bandwidth

Case study - Unbounded buffer

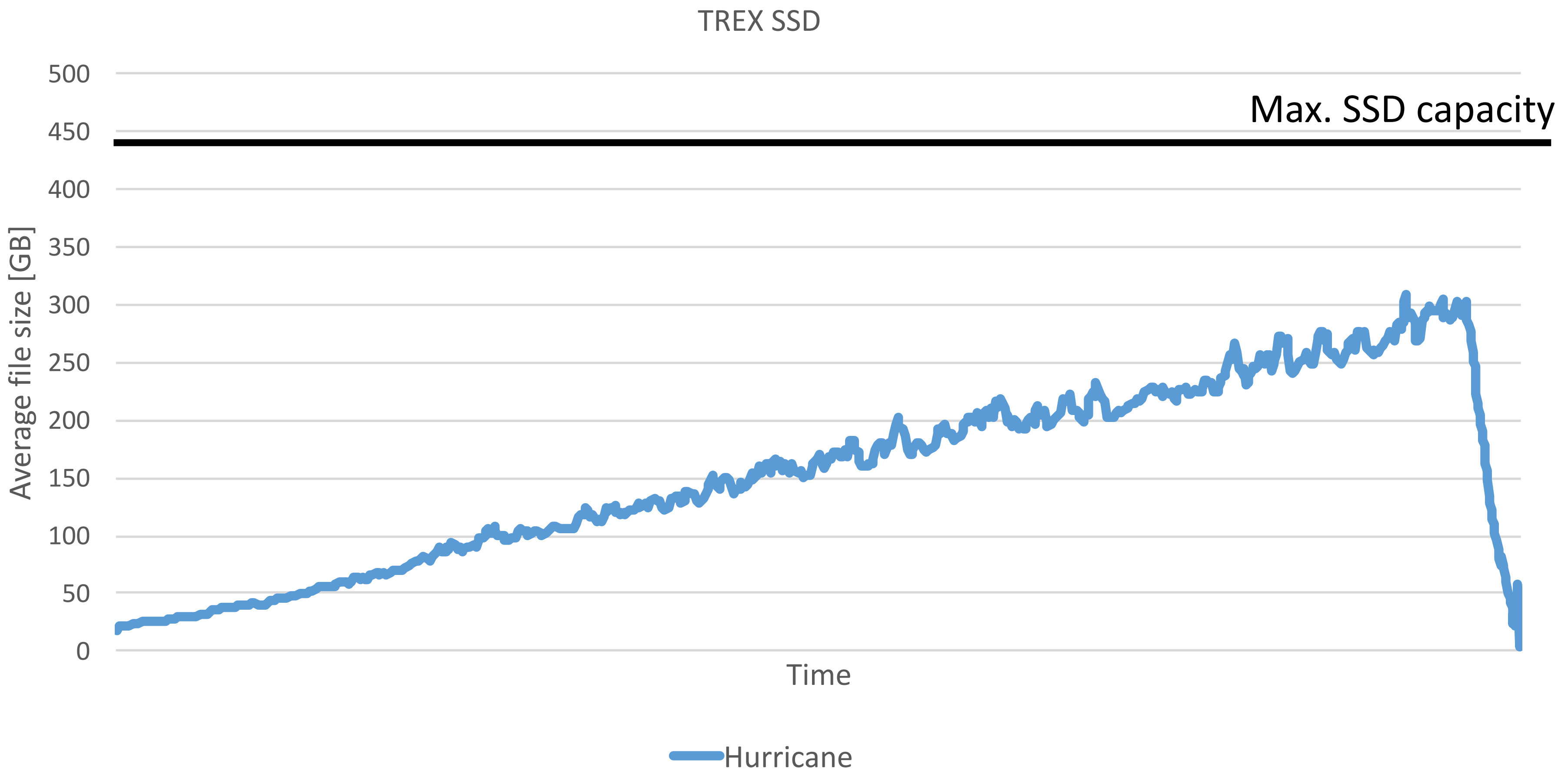
- Each node write/read a certain amount of data
- Use Hurricane to amortize mismatch between producers and consumers
- Show that it can accomodate temporary spikes seamlessly
- 16 machines on T-REX -> 16 servers & clients
- Measure average file size

1 TB per node - ~2.5x SSD capacity



8 TB per node- ~20x SSD capacity

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE



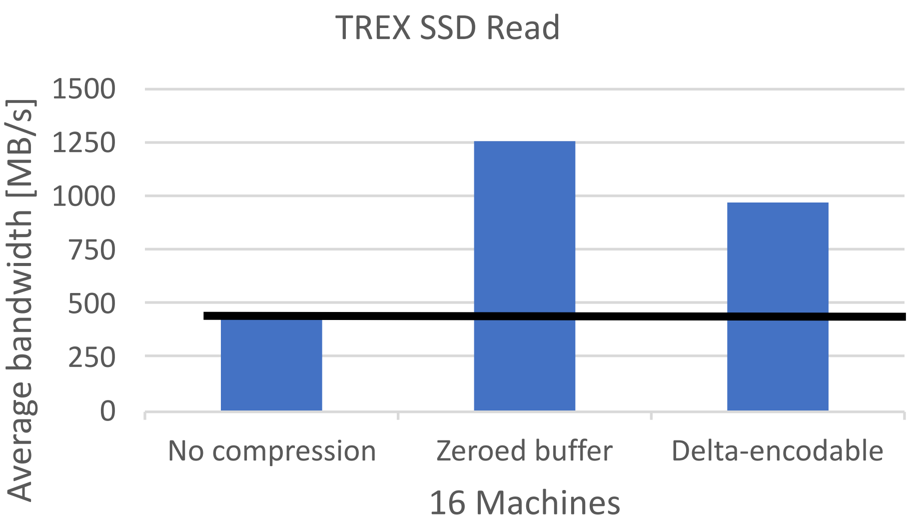
Case study - Summary

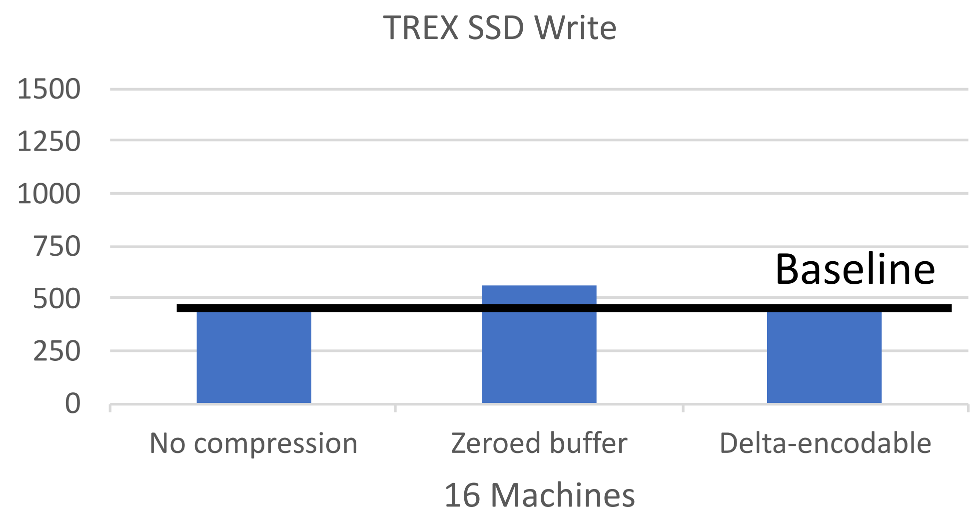
- We can write much more than the cluster can handle
- Still full I/O bandwidth !
- Effectively amortize write-read imbalance
- No degradation of I/O bandwidth
- Hurricane can buy you time to react to a write deluge

Case study - Compression

- Each node writes/reads 16 GB of data
- Compress (LZ4) data at disk rate
- 16 machines on T-REX -> 16 servers & clients
- Compare three cases :
  - No compression
  - Compress zeroes data
  - Compress data amenable to delta-encoding
- Measure average bandwidth

16GB of input





Type of data	Input	Output	Read speed	Write speed
No compression	16 GB	16 GB	443 MB/s	455 MB/s
Zeroed buffer	16 GB	65 MB	1260 MB/s	565 MB/s
Delta-encodable	16 GB	7.2GB	964 MB/s	455 MB/s

If data amenable to compression, both speed and storage gains !

Future work

Future work

- Fault tolerance
- Implement Chaos on Hurricane
- Integrate Hurricane into Hadoop or Spark
- Further experiments

Conclusion

Conclusion

- Hurricane is scalable decentralized storage system
- HDFS-like RPC interface (flexible)
- Outperforms HDFS
- Maximal I/O bandwidth

THANK YOU

QUESTIONS ?

References

1. Amitabha Roy, Laurent Bindschaedler, Jasmina Malicevic, and Willy Zwaenepoel: Chaos: Scale-out Graph Processing from Secondary Storage. SOSP 2015.
2. Amitabha Roy, Ivo Mihailovic, and Willy Zwaenepoel: X-Stream: Edge-centric Graph Processing using Streaming Partitions. SOSP 2013.
3. Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler: The Hadoop Distributed File System. MSST 2010.
4. Mark Slee, Aditya Agarwal and Marc Kwiatkowski: Thrift: Scalable cross-language services implementation. Facebook white paper 2007.
5. Edmund B. Nightingale, Jeremy Elson, Jinliang Fan, Owen Hofmann, Jon Howell and Yutaka Suzue: Flat Datacenter Storage. OSDI 12.
6. Michael Mitzenmacher : The Power of Two Choices in Randomized Load Balancing. IEE Transactions on Parallel and Distributed Systems 2001.