Limplock: Understanding the Impact of Limpware on Scale-Out Cloud Systems

THANH DO, MINGZHE HAO, TANAKORN LEESATAPORNWONGSA, TIRATAT PATANA-ANAKE, HARYADI S. GUNAWI

SOCC 2013

Presented by: Uttam Thakore

Outline

What are “limpware” and “limplock”?

Illustration

Limpbench: limplock benchmarking tool

Evaluation of limplock on:

- **Hadoop**
- **HDFS**
- **ZooKeeper**
- **Cassandra**
- **HBase**

Fault-tolerance & performance failures

Large cloud systems are very complex

→ Number of HW failures continue to increase

Existing mechanisms detect **crash-stop failures** and some **performance failures**

- E.g., stragglers, unbalanced load

“Limplock”

Performance failure due to “limpware” – hardware/software with significantly degraded performance

Twilight zone between slow and failed hardware/software

- Undetected by existing mechanisms, so recovery does not happen

Causes of Limpware

Disk:

- Weak head
- Vibration
- Firmware bugs
- Bad sector remapping

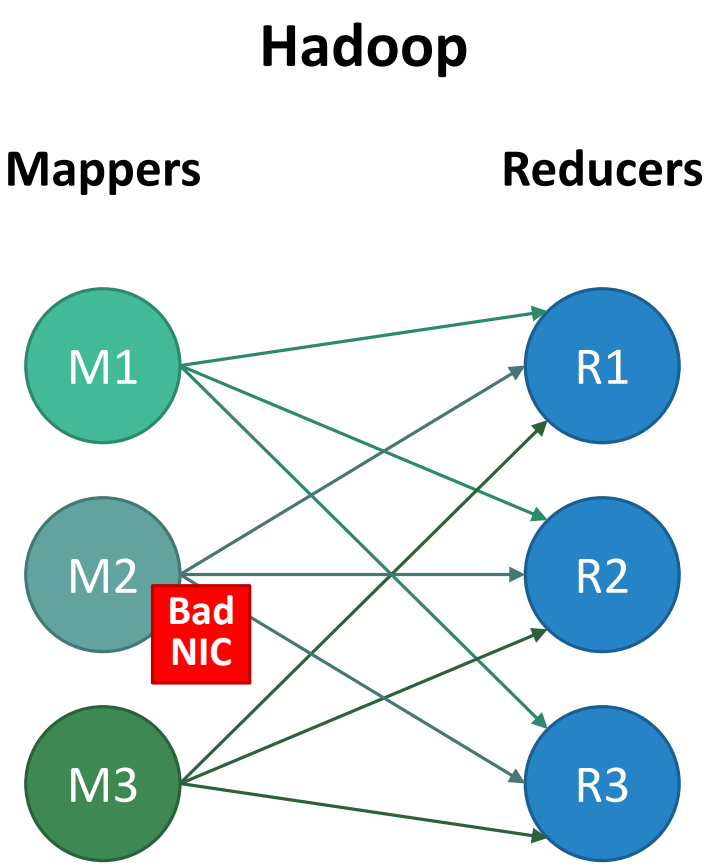
Network:

- Broken module or adapter
- Corrupt packets → Error correction
- Network driver bugs
- Power fluctuations

Types of Limplock

Operation limplock:

- Single point of failure (SPOF)
- Long timeout durations
- Memoryless retry

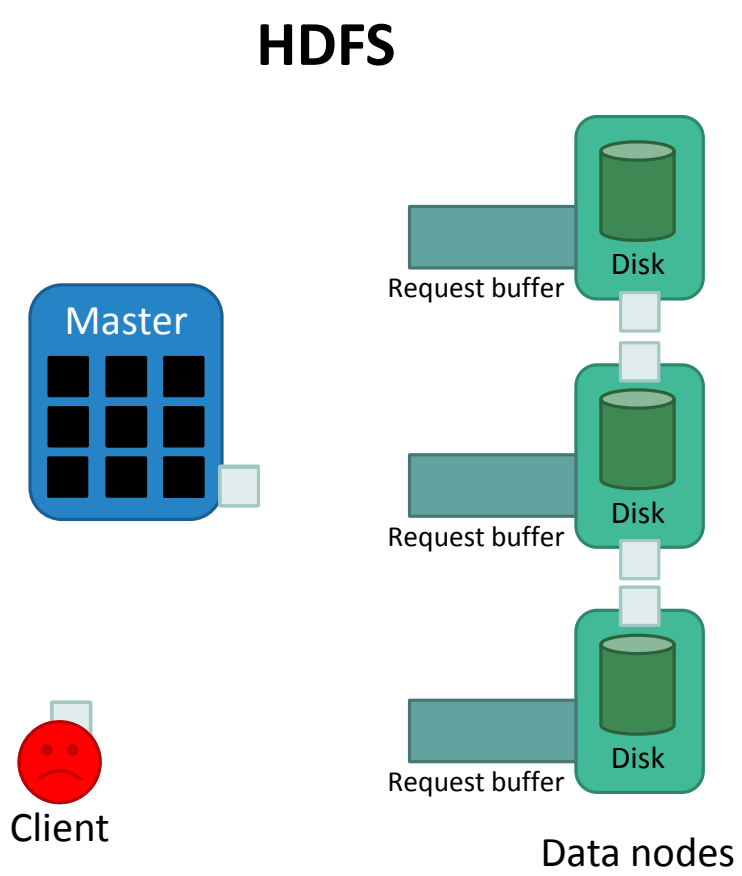


Types of Limplock

Operation limplock:
- Single point of failure (SPOF)
- Long timeout durations
- Memoryless retry

cascades into

Node limplock:
- Exhaustion of resource pool
- Unbounded thread or queue



Types of Limplock

Operation limplock:
- Single point of failure (SPOF)
- Long timeout durations
- Memoryless retry

Node limplock:
- Exhaustion of resource pool
- Unbounded thread or queue

Cluster limplock:
- All nodes in limplock
- Master node in limplock

Limpbench

Goals:

- Quantify limplock in cloud systems
- Uncover designs that lead to limplock

56 experiments, benchmarking 5 cloud systems

- Hits 22 protocols

Components:

- Evaluate data-intensive protocols
- Stress request load
- Fault- (and limp-) injection
- White-box analysis

Limpbench

ID	Protocol	Limp-ware	Injected Node	Workload	Base Latency	OL	NL	CL
F1	Logging	Disk	Master	Create 8000 empty files	12	√	√	√
F2	Write	Disk	Data	Create 30 64-MB files	182	·	·	·
F3	Read	Disk	Data	Read 30 64-MB files	120	·	·	·
F4	Metadata Read/Logging	Disk	Master	Stats 1000 files + heavy updates	9	√	√	√
F5	Checkpoint	Disk	Secondary	Checkpoint 60K transactions	39	√	·	·
F6	Write	Net	Data	Create 30 64-MB files	208	√	·	·
F7	Read	Net	Data	Read 30 64-MB files	104	√	·	·
F8	Regeneration	Net	Data	Regenerate 90 blocks	432	√	√	√
F9	Regeneration	Net	Data-S/Data-D	Scale replication factor from 2 to 4	11	√	·	·
F10	Balancing	Net	Data-O/Data-U	Move 3.47 GB of data	4105	√	·	·
F11	Decommission	Net	Data-L/Data-R	Decommission a node having 90 blocks	174	√	√	√
H1	Speculative execution	Net	Mapper	WordCount: 512 MB dataset	80	√	·	·
H2	Speculative execution	Net	Reducer	WordCount: 512 MB dataset	80	·	·	·
H3	Speculative execution	Net	Job Tracker	WordCount: 512 MB dataset	80	·	·	·
H4	Speculative execution	Net	Task Node	1000-task Facebook workload	4320	√	√	√
Z1	Get	Net	Leader	Get 7000 1-KB znodes	4	·	·	·
Z2	Get	Net	Follower	Get 7000 1-KB znodes	5	·	·	·
Z3	Set	Net	Leader	Set 7000 1-KB znodes	23	√	√	√
Z4	Set	Net	Follower	Set 7000 1-KB znodes	26	·	·	·
Z5	Set	Net	Follower	Set 20KB data 6000 times to 100 znodes	64	√	√	√
C1	Put (quorum)	Net	Data	Put 240K KeyValues	66	·	·	·
C2	Get (quorum)	Net	Data	Get 45K KeyValues	73	·	·	·
C3	Get (one) + Put (all)	Net	Data	Get 45K KeyValues + heavy puts	36	·	·	·
B1	Put	Net	Region Server	Put 300K KeyValues	61	√	·	·
B2	Get	Net	Region Server	Get 300K KeyValues	151	√	·	·
B3	Scan	Net	Region Server	Scan 300K KeyValues	17	√	·	·
B4	Cache Get/Put	Net	Data-H	Get 100 KeyValues + heavy puts	4	√	√	·
B5	Compaction	Net	Region Server	Compact 4 100-MB sstables	122	√	√	·

Table taken from paper, [2]

Experimental evaluation of cloud systems

HDFS

Component with Limpware	Master disk	Datanode disk	Datanode NIC
Operation Limplock?	**Yes**	No	**Yes**
Node Limplock?	**Yes**	No	**Yes**
Cluster Limplock?	**Yes**	No	**Yes**

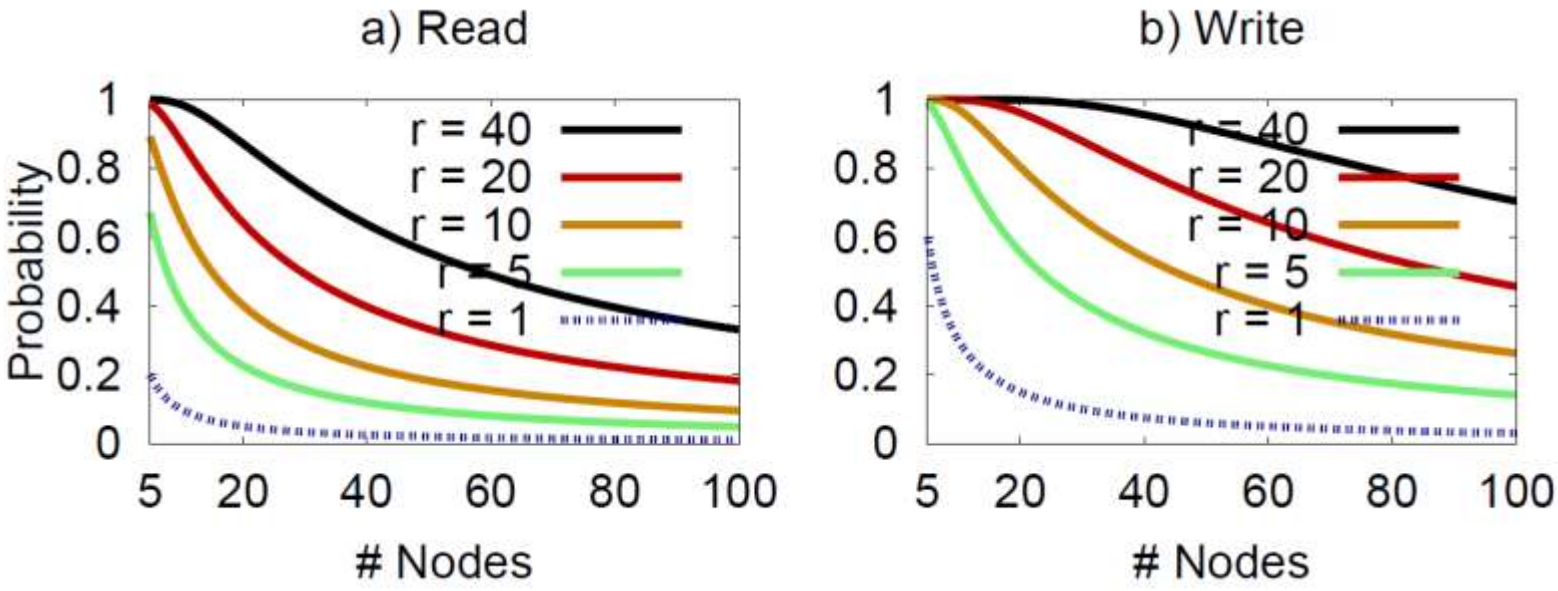
Datanode writes buffer in OS, so no limplock below write threshold

Limping datanode NIC → limping reads and writes

Logging when master disk is limping → cluster limplock

Regeneration limplock → datanode and cluster limplock

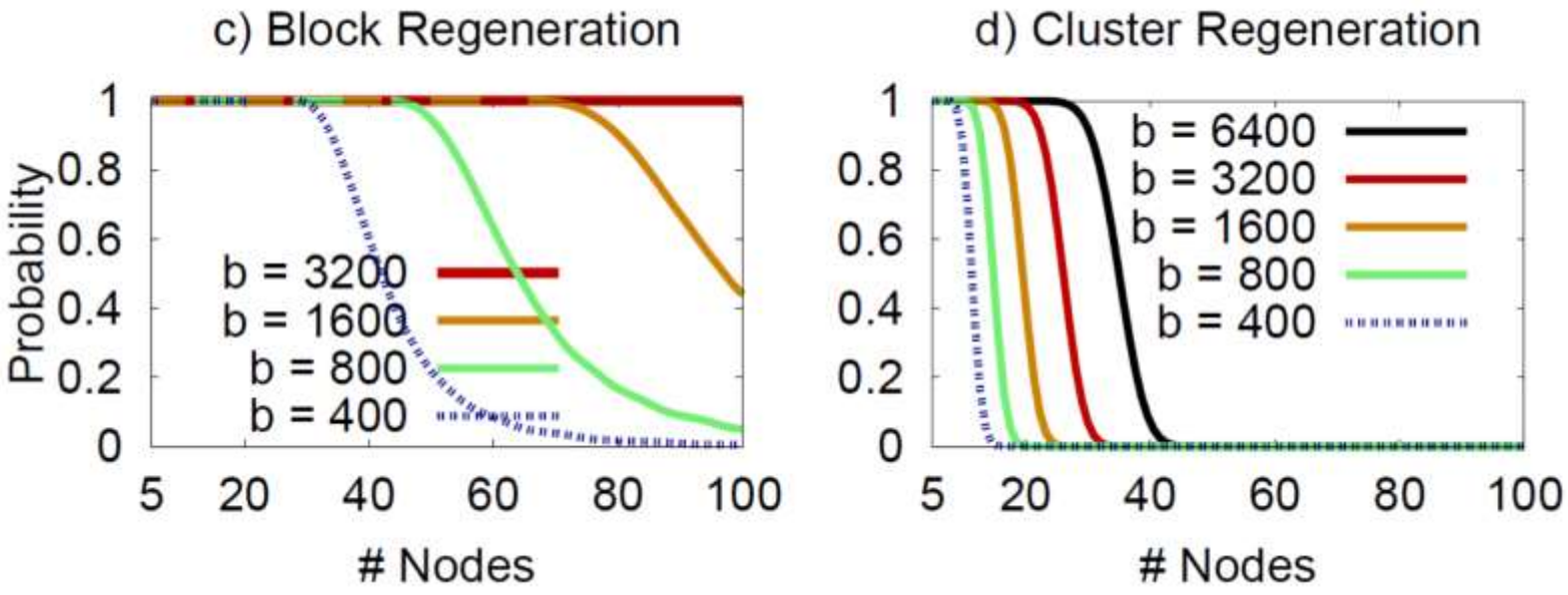
HDFS



Probability of experiencing at least one limplock
(r = number of user requests)

Figure taken from paper, [2]

HDFS



Probability of experiencing at least one regeneration limplock
(b = number of 64MB blocks to regenerate)

Figure taken from paper, [2]

HDFS – Limpbench results

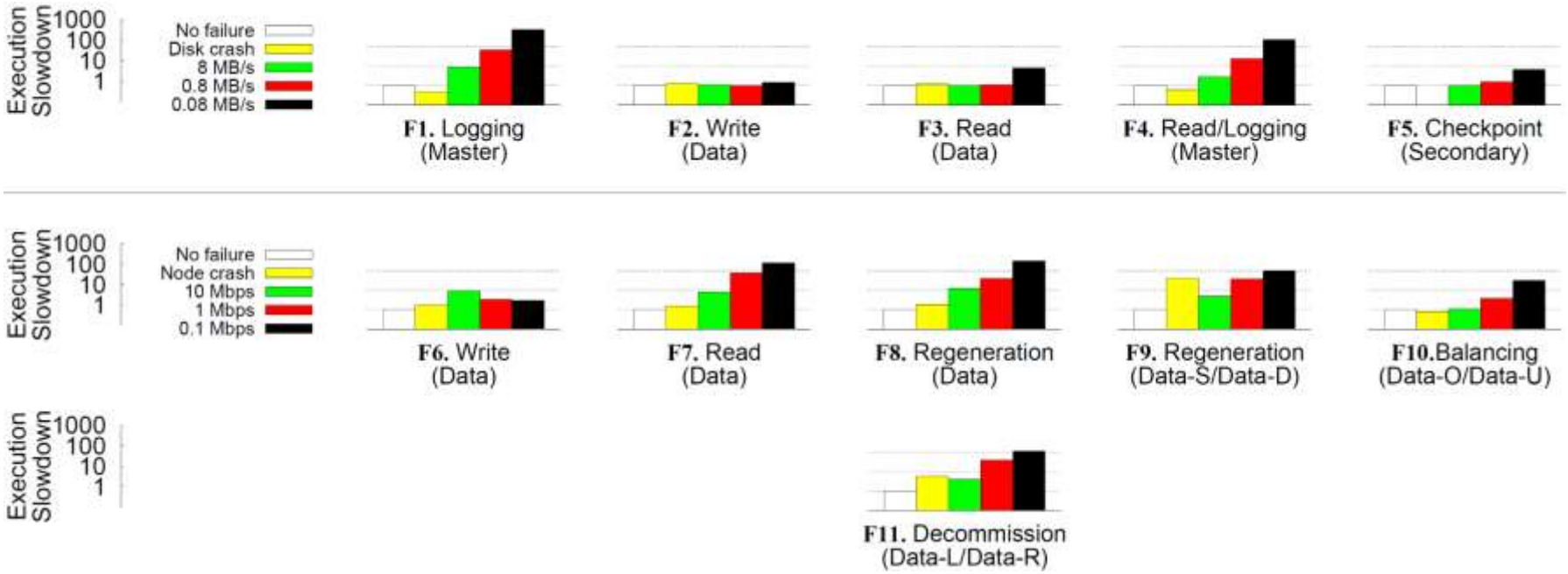


Figure taken from paper, [2]

Hadoop

Node with Limpware	Mapper	Reducer	Job Tracker	Task Node
Operation Limplock?	**Yes**	No	No	**Yes**
Node Limplock?	No	No	No	**Yes**
Cluster Limplock?	No	No	No	**Yes**

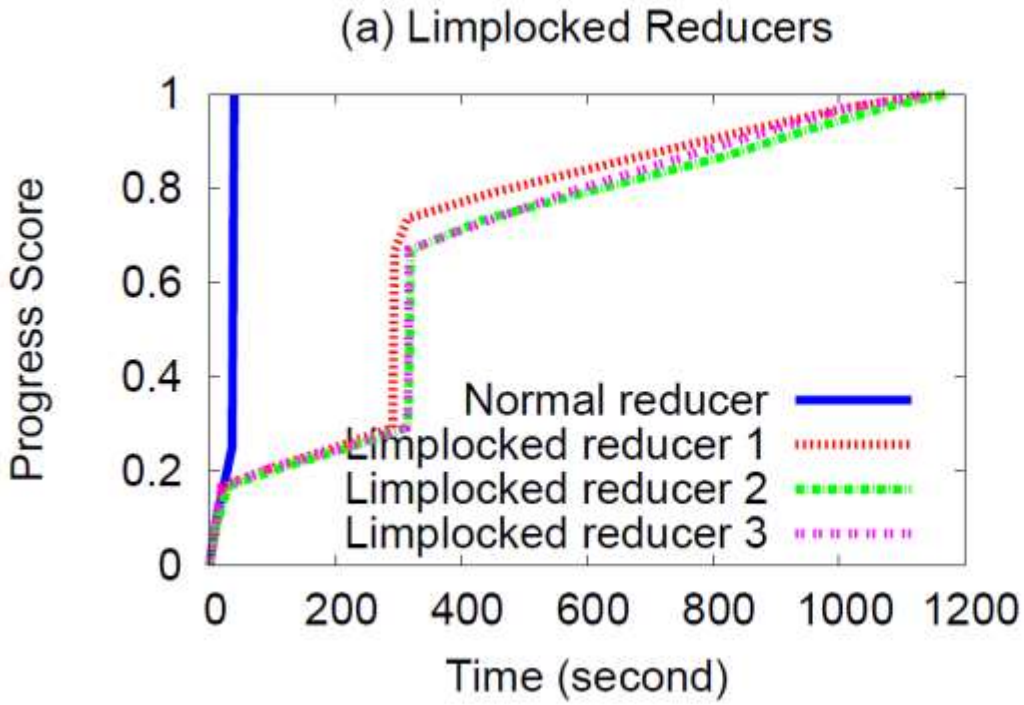
Mapper with slow NIC → all reducers slow down during shuffle

HDFS limplock → reducer and mapper limplock

- No speculative execution!

All tasks are limping → node and cluster limplock

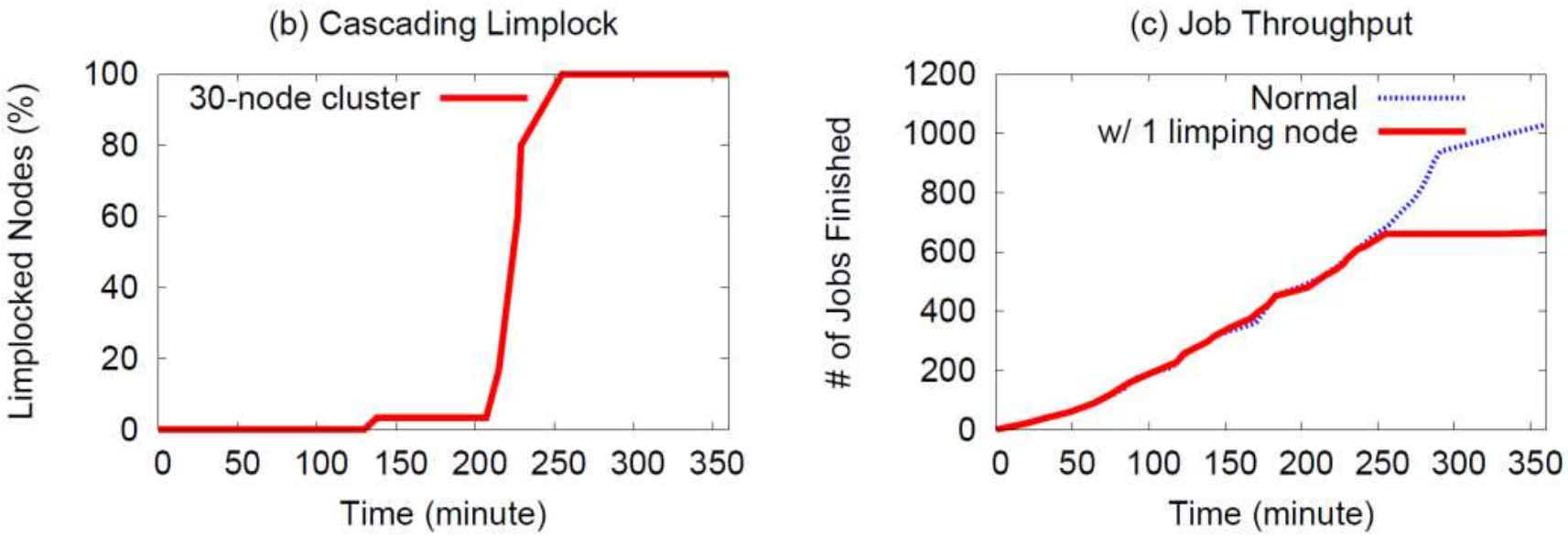
Hadoop



Experiment H1: Degraded mapper NIC

Figure taken from paper, [2]

Hadoop



Facebook workload on a 30-node cluster

Figure taken from paper, [2]

Hadoop – Limpbench results

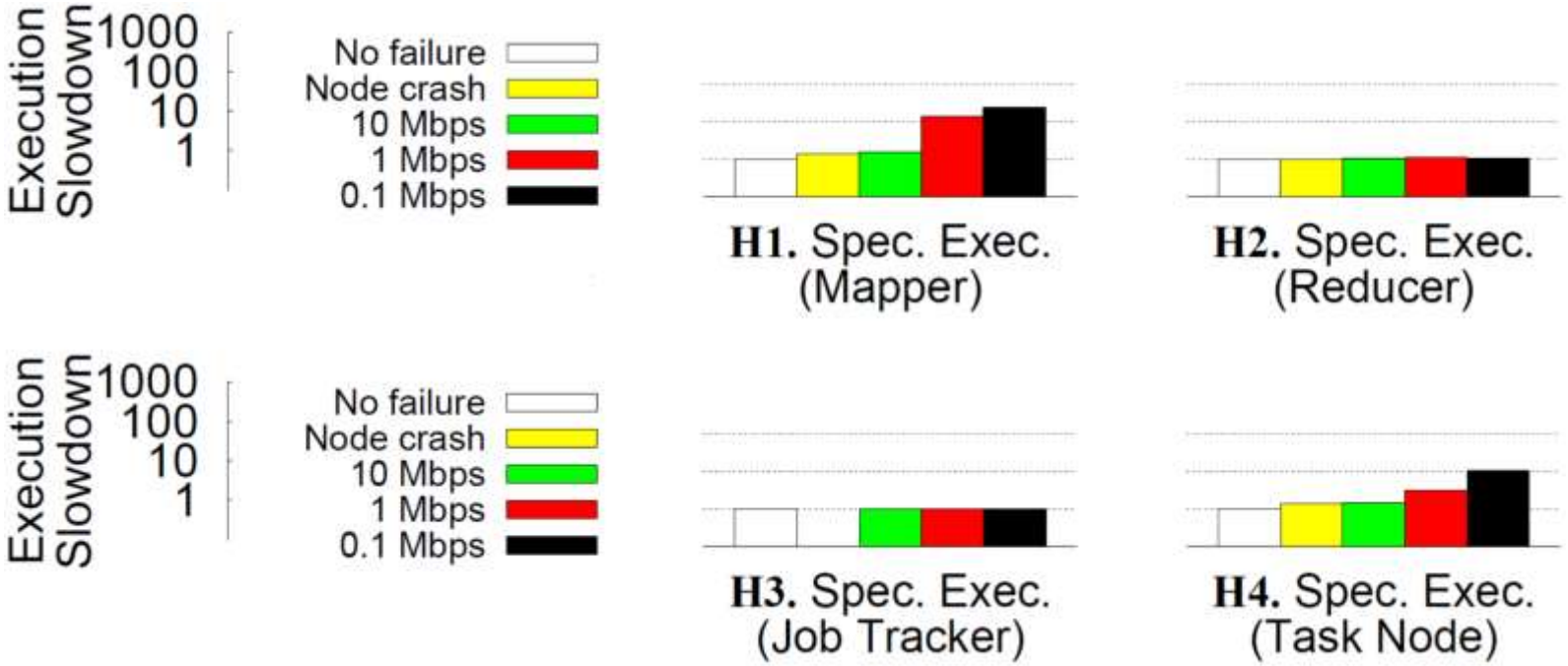


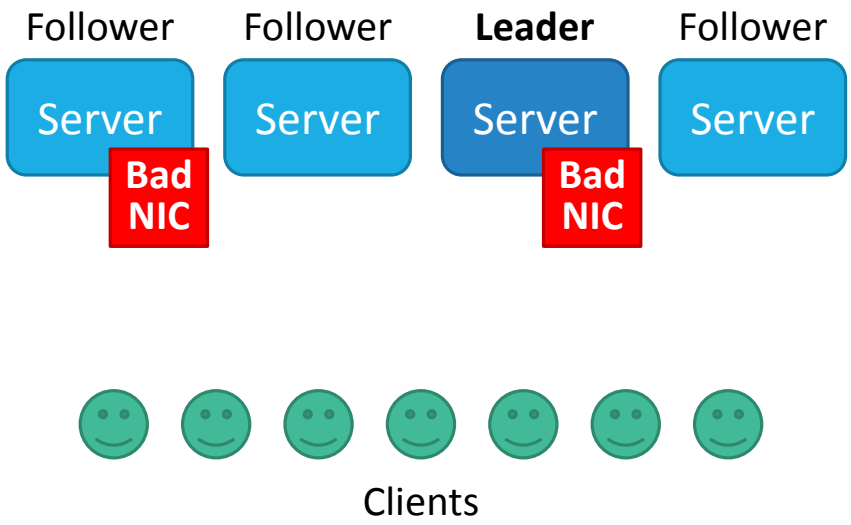
Figure taken from paper, [2]

ZooKeeper

Operations:

- **Get** – served by any node
- Create
- Set
- Delete
- Sync

Update operations – must go through leader, require quorum of followers



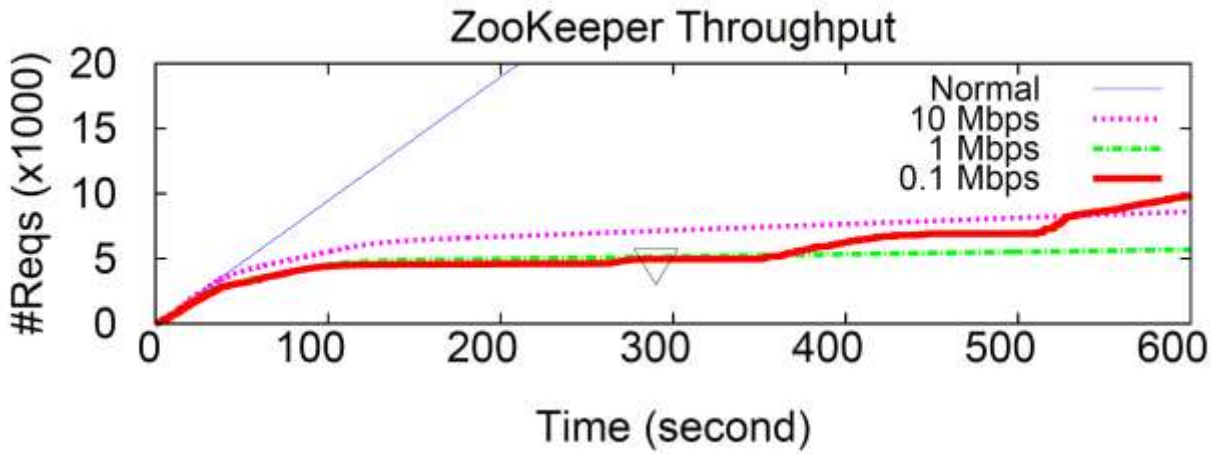
ZooKeeper

Component with Limpware	Leader NIC	Follower NIC
Operation Limplock?	**Yes**	**Yes**
Node Limplock?	**Yes**	**Yes**
Cluster Limplock?	**Yes**	**Yes**

Gets are limplock-free

Updates are subject to leader or follower node limplock

ZooKeeper



ZooKeeper throughput under single follower NIC degradation

Figure taken from paper, [2]

ZooKeeper – Limpbench results

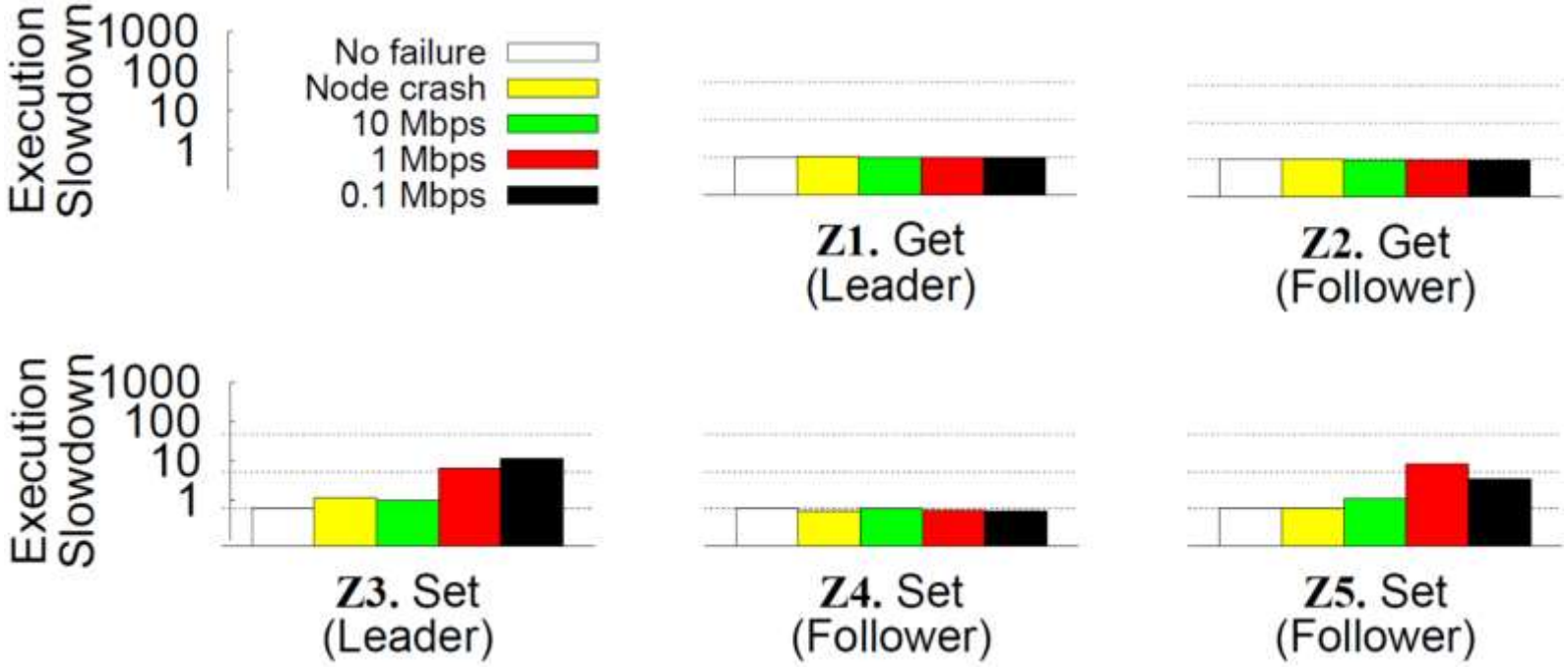


Figure taken from paper, [2]

Cassandra

Node with Limpware	Data Node
Operation Limplock?	**Yes**
Node Limplock?	No
Cluster Limplock?	No

Weak consistency operations → No limplock

- However, “flapping” – 2x performance degradation

Full consistency operations → Limplock

Operation limplock does not cascade

Cassandra – Limpbench results

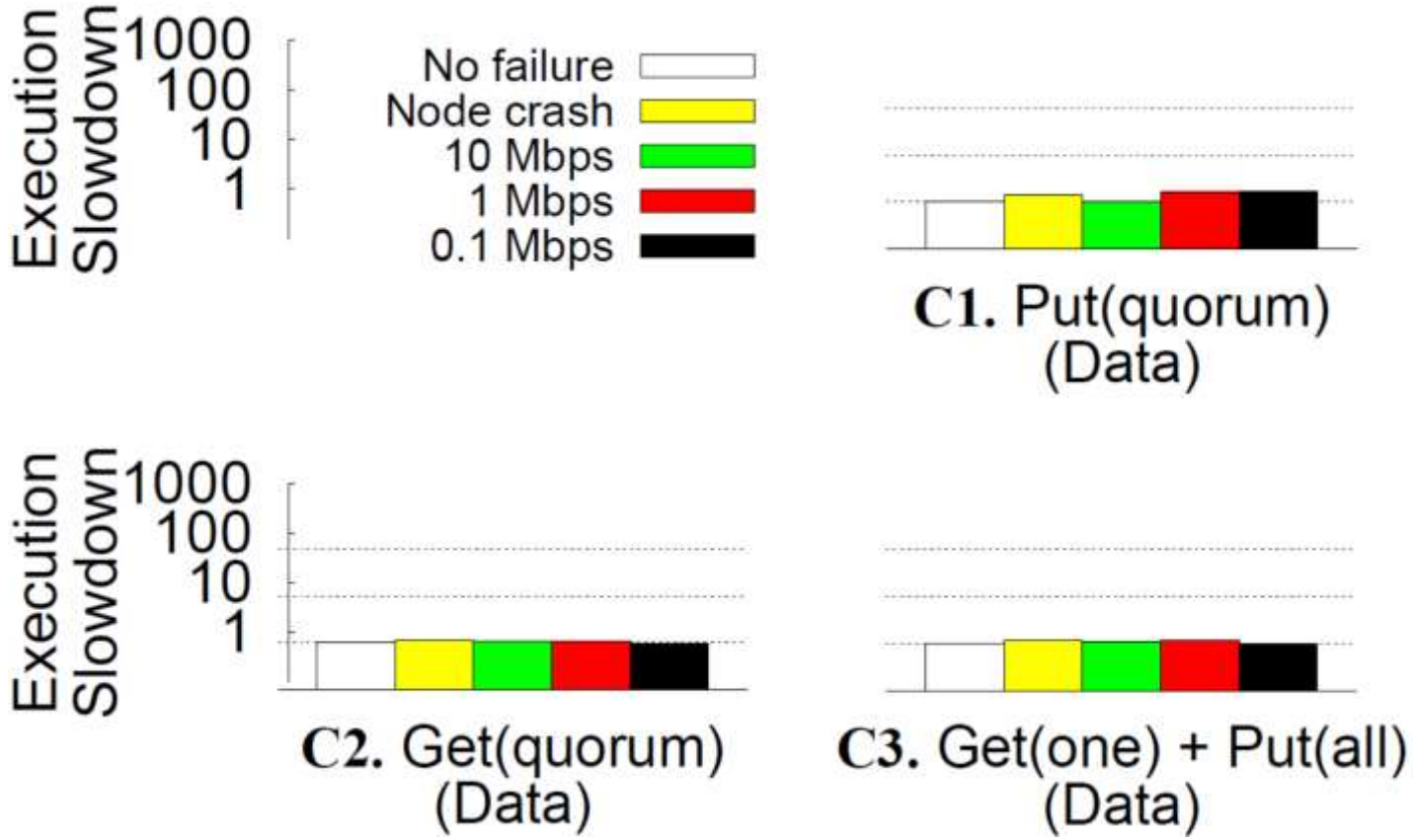


Figure taken from paper, [2]

HBase

Node with Limpware	Region server NIC	Master server NIC	HDFS read limplock	HDFS write limplock
Operation Limplock?	**Yes**	No	**Yes**	**Yes**
Node Limplock?	**Yes**	No	**Yes**	**Yes**
Cluster Limplock?	**Yes**	No	**Yes**	**Yes**

HDFS limplock → limplock on *all* HBase protocols

- Only reprieve is if data is in HBase caches

Resource exhaustion from HDFS write limplock → HBase region node limplock

Limplocked region server affecting metadata → cluster limplock

HBase – Limpbench results

Figure taken from paper, [2]

What to do?

Limplock avoidance:

- Converting limpware to crash-stop failures
- Quarantining limpware to prevent cascading
- Design in limplock tolerance
  - E.g., differentiated threads per operation type (Cassandra)

Limplock detection

- End-to-end limpware detection
- Traditional straggler detection methods

Recovery:

- Fail-in-place
- Recovery mechanisms with memory

Conclusion

"Limplock" is a real concern

Existing failure detection and recovery mechanisms do not handle limpware correctly

This paper serves to identify the failure type for further formal study

Discussion

Do you think these causes are complete?

- How would we prove this?
- Formal definition of limplock?
- What are the most primitive forms of limpware?

Lack of concrete recovery mechanisms

Limpbench is lacking, not comprehensive

Where limplock does not cascade, scale mitigates limplock

How does limplock compare to network bottlenecks?

How would you use the paper in your research? Future work?

References

[1] Thanh Do, “Limplock: Understanding the Impact of Limpware on Scale-out Cloud Systems,” presented at the 4th annual Symposium on Cloud Computing, 2013.

[2] T. Do, M. Hao, T. Leesatapornwongsa, T. Patana-anake, and H. S. Gunawi, “Limplock: understanding the impact of limpware on scale-out cloud systems,” in *Proceedings of the 4th annual Symposium on Cloud Computing*, 2013, pp. 1–14.

[3] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, Berkeley, CA, USA, 2004, pp. 10–10.

[4] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop Distributed File System,” in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–10.

[5] “ProjectDescription - Apache ZooKeeper - Apache Software Foundation.” [Online]. Available: https://cwiki.apache.org/confluence/display/ZOOKEEPER/ProjectDescription. [Accessed: 10-Mar-2015].

[6] “The Apache Cassandra Project.” [Online]. Available: http://cassandra.apache.org/. [Accessed: 10-Mar-2015].

[7] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable: A distributed storage system for structured data,” *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

Backup Slides

Let me tell you a story…

"… **1GB NIC card** on a machine that suddenly starts transmitting at **1 Kbps**,

this slow machine caused a chain reaction upstream in such a way that the performance of entire workload for a **100 node cluster was crawling at a snail's pace, effectively making the system unavailable for all practical purposes**."

– Borthakur of Facebook

Quote taken from SoCC presentation slides, [1]

Limpbench

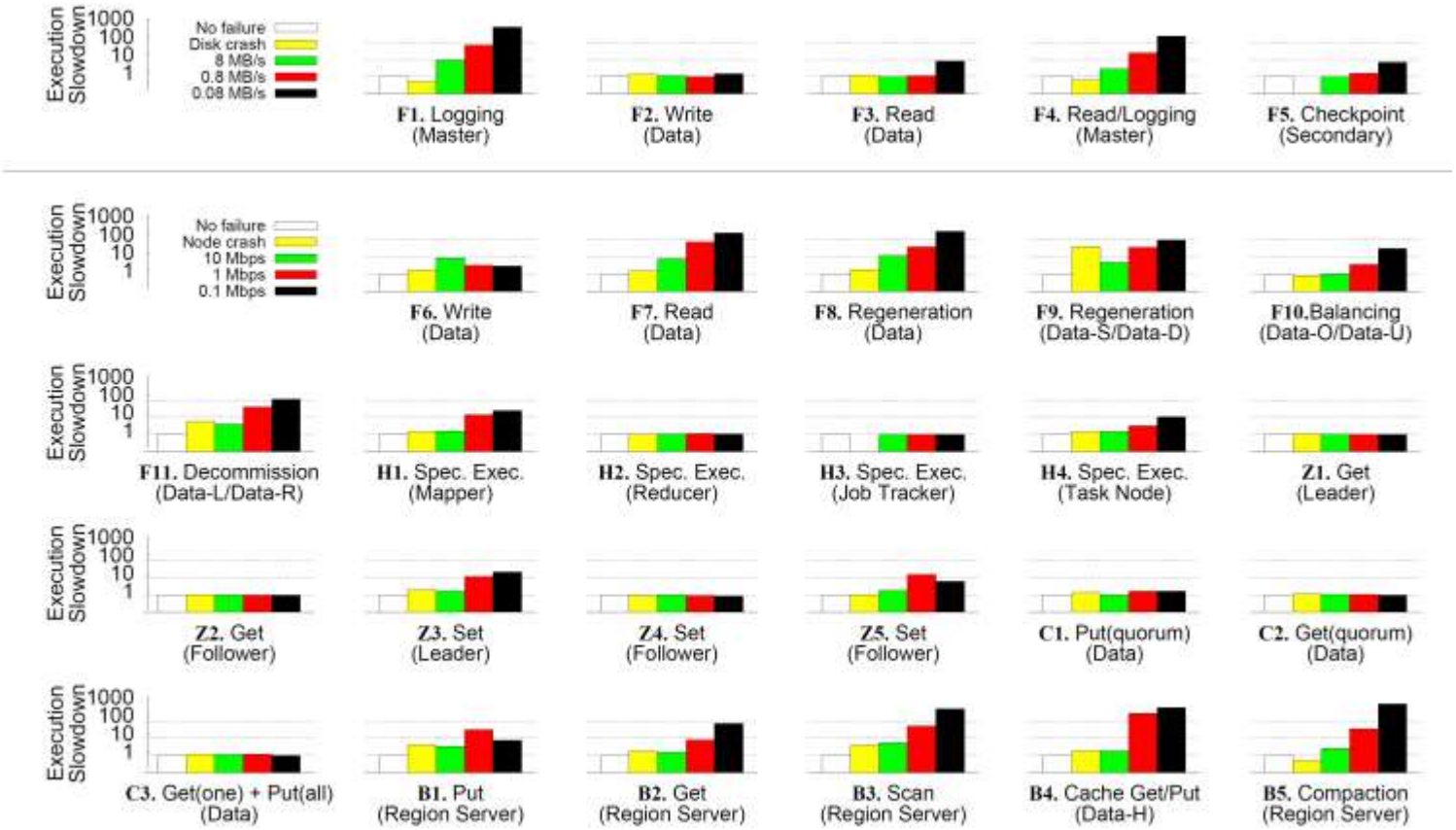
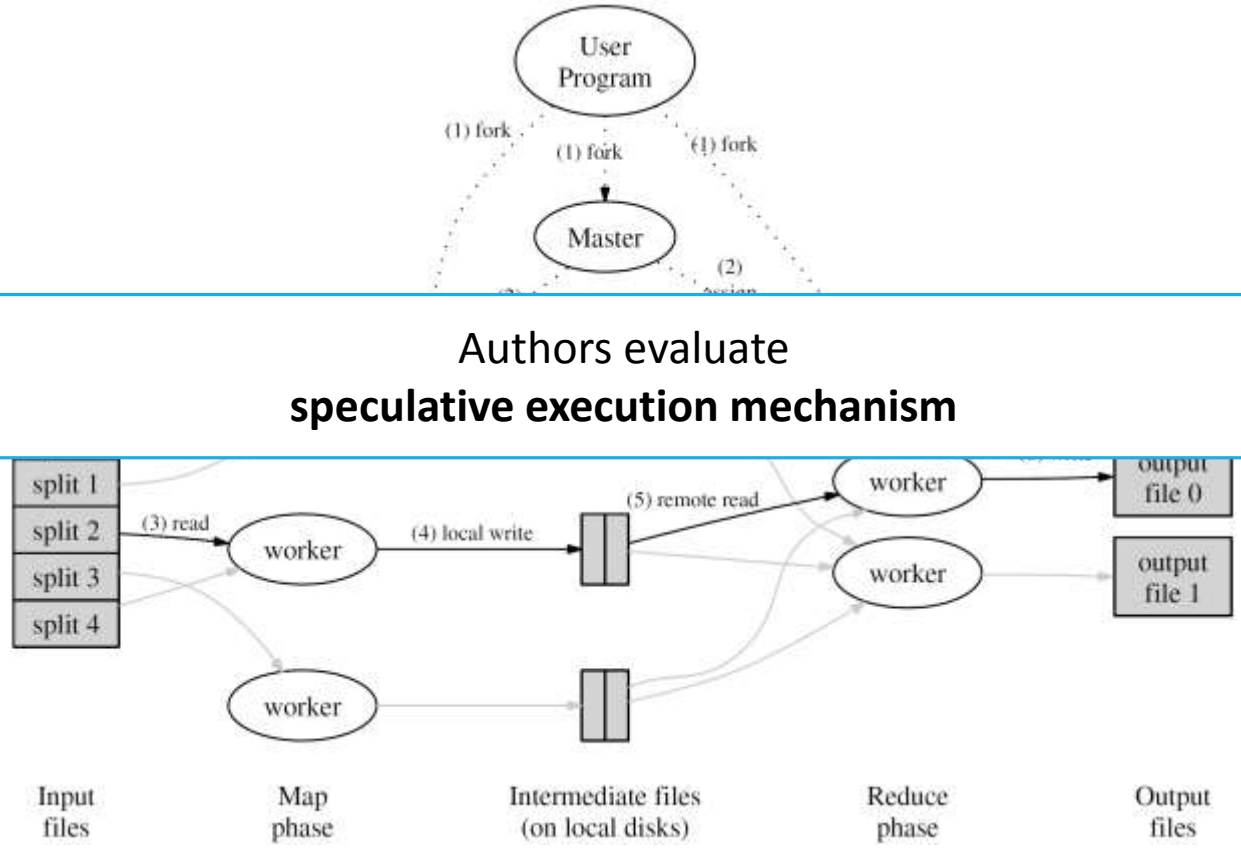


Figure taken from paper, [2]

Hadoop



Authors evaluate
speculative execution mechanism

Figure taken from MapReduce paper, [3]

HDFS

Master node fields requests

Data nodes service requests and store data locally

Data stored in 64-MB blocks

Triple replication

Regeneration runs in background upon failure of datanode

Authors evaluate effect of **degraded disk and NIC** on **master** and **data nodes**

ZooKeeper

Single leader node, with multiple followers

Operations:

- Create – must go through leader
- Get – served by any node
- Set – must go through leader
- Delete – must go through leader
- Sync – must go through leader

Authors evaluate effect of **degraded NIC** on **leaders** and **followers**

Cassandra

Distributed key-value store

Node involvement in operations depends on consistentency level:

- ONE
- QUORUM
- ALL

Replication factor = 3

Authors evaluate effect of **degraded NIC** on **put** and **get** protocols

HBase

Distributed key-value store running on top of HDFS

Row ranges are managed by **region servers**

Region assignment to nodes is handled by **master servers**

Authors evaluate effect of **degraded NIC** on region servers