Speaker: Ricardo

Toward a Generic Fault Tolerance Technique for Partial Network Partitioning

Mohammed Alfatafta, Basil Alkhatib, Ahmed Alquraan, and Samer Al-Kiswany, University of Waterloo, Canada

G13

Sara Machado, 86923
Rafael Figueiredo, 90770
Ricardo Grade, 90774

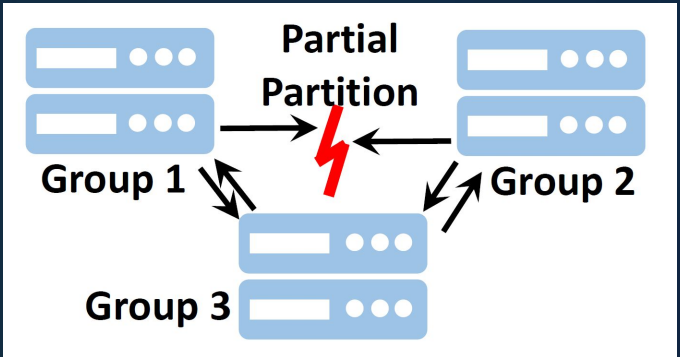
Speaker: Ricardo

Introduction

- This work focuses its study on Partial Network Partitions failures.

Presentation Phases:

1. Understand the impact and characteristics of this type of failures;
2. Explore some implementations of fault tolerance techniques;
3. Identify their shortcomings;
4. Explore Nifty, a generic fault tolerance mechanism for Partial Network Partitions.



Speaker: Ricardo

Utility

- Network partitions will always happen;
- Network partitions are common in large-scale clusters;
- It can cause catastrophic failures, such as completely interrupt the availability of a cluster.

Speaker: Ricardo

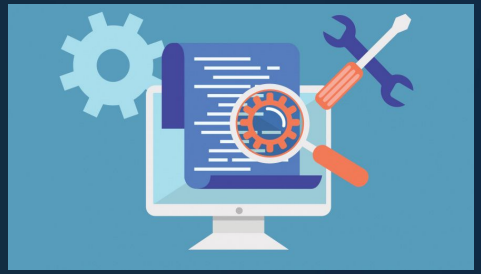
Analysis of Partial Network Partitioning Failures

- 51 reports found in 12 popular systems were analysed.

Among these systems:

- Messaging systems: Kafka, ActiveMQ and RabbitMQ;
- Core of the Hadoop platform: MapReduce, HDFS and HBase;
- Search engine: Elasticsearch;
- Popular database: MongoDB.

Reports prior to 2011 and marked as *"Minor"* have been discarded.



Speaker: Ricardo

Limitations

- Representativeness of the studied systems:
  - The results cannot be completely generalized.
- Limited number of tickets:
  - The tickets analysed are a sample.
- Priority bias:
  - Low priority tickets have been discarded, which can skew the results.
- Observer error:
  - Analyzed by different teams and discussed by everyone.

Speaker: Ricardo

Finding 1

A significant percentage (74.5%) of the studied failures have a catastrophic impact.

- The most common failures identified in this analysis are catastrophic.

Catastrophic Impact:

1. Data loss is the most common.
   - Example: In HBase, whenever a region server is isolated from the master while both can reach HDFS.
2. System unavailability is the second most common.
   - Example: Whenever Zookeeper manages to reach a master that the other cannot.

Non-Catastrophic Impact:

1. Reduced availability.
   - Example: Leader-election thrashing.

Impact	%	
Data loss	23.5%	Catastrophic (74.5%)
System unavailability	21.6%	
Stale read	15.7%	
Data corruption	5.9%	
Dirty read	3.9%	
Data unavailability	3.9%	
Reduced availability	23.5 %	
Other	2%	

Speaker: Ricardo

Finding 2

Most of the studied failures (84.3%) are silent the user is not informed about their occurrence.

- Partial partitioning failures happen relatively often and have a dangerous impact.
- There is no system that correctly handles these failures.
- The user does not know what happened or what he could do to resolve this.

Speaker: Ricardo

Finding 3

Leader election, configuration change, and replication protocol are the mechanisms most vulnerable to partial network partitioning.

The most vulnerable mechanisms to partial network partitions:

1. Leader election.
   - It can lead to the election of a leader on each side of the partition.
2. Configuration change.
   - It can lead to a complete system crash.
3. Replication protocol.
   - It can cause a client to read outdated data.

Mechanism	%
Leader election	37.3%
Configuration change	19.6%
Replication protocol	17.6%
Request routing	11.8%
Scheduling	5.9%
Data migration	5.9%
Data consolidation	2%

Speaker: Ricardo

Finding 4

Most failures (60.8%) do not require client access or require only that clients access one side of the partition.

- Only access to one side of a partition, or even lack of access can still lead to failures due to background operations.
    - Example: In MongoDB when a partition isolates the balancer from the metadata service, while a rebalance operation is being performed can lead to the unavailability of the shard being relocated.

Speaker: Ricardo

Finding 5

The majority of failures (68.6%) require three or fewer events (other than the partial partition) to manifest.

- A small number of events can lead to failures.
  - Even only the Partial Partition event itself can.

Number of events	%
1 (Just a partial partition)	13.7%
2	9.8%
3	31.4%
4	13.7%
>4	31.4%

Speaker: Ricardo

Finding 6

All the studied failures can be triggered by a single-node partial partition, with 33.3% of them happen by partitioning any node.

- Regardless of which node is isolated, one third of failures are manifested.
- The most critical role is the leader, which is also a very common role to be played, since almost all nodes in a cluster are leaders of some shard.

Network Partition Characteristic		%
Partition any node		33.3%
Partition a specific node		66.6%
• Leader	45.1%	
• Nodes with a special role	9.8%	
• A central service	7.8%	
• New nodes	2%	

Speaker: Ricardo

Finding 7

All the studied failures, except one, are deterministic or have known time constraints.

- Most failures are deterministic.
- A third of the failures have known time constraints.
  - Example: The period before concluding that a node has failed

Timing constraint	%
No timing constraints	64.7%
Known timing constraints	33.3%
Nondeterministic	2%

Speaker: Ricardo

Finding 8

The resolution of 56.8% of the fixed failures required changing the design of a protocol or a mechanism.

- Almost half of the failures are caused by design flaws.
- System designers do not give this problem enough attention that it requires.

Flaw type	%	Average Time to Resolution
Design	41.2%	260 days
Implementation	31.4%	98 days
Unresolved	27.5%	-

Speaker: Ricardo

Finding 9

All failures can be reproduced with five nodes, and all but one can be reproduced using a fault injection tool.

- Almost all failures can be reproduced with a cluster of just three nodes.
- All observed failures can be reproduced by a five-node cluster.
- Easy to reproduce, which makes it more dangerous

Number of nodes	%
3 nodes	76.5%
4 nodes	21.6%
5 nodes	2%

Speaker: Rafael

Dissecting Modern Fault Tolerance Techniques

- **Eight systems** that implemented fault tolerance techniques for partial partitions were analysed:
  - Elasticsearch (**Changed**);
  - MongoDB (**Changed**);
  - RabbitMQ (**Changed**);
  - MapReduce (**Changed**);
  - HBase (**Changed**);
  - Mesos (**Changed**);
  - VoltDB (**Found**);
  - LogCabin (**Found**).
- **Four approaches identified**:

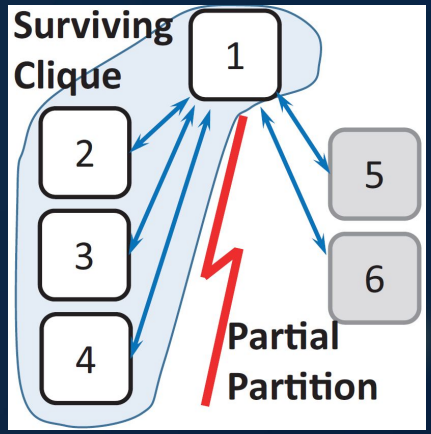
1. Identifying the Surviving Clique (VoltDB)
2. Checking Neighbors' views (RabbitMQ, Elasticsearch)
3. Failure Verification (MongoDB, LogCabin)
4. Neutralizing Partitioned Nodes (MapReduce, HBase, Mesos)

Speaker: Rafael

1 Dissecting Modern Fault Tolerance Techniques

Identifying the Surviving Clique [VoltDB]

- Main Idea: Partial Partition Occurs → Find maximum clique of nodes → Every other node shuts down



- VoltDB Approach:
  - Send heartbeats between all nodes periodically;
  - If a node did not received an heartbeat from all nodes: Recovery Procedure.
- Recovery Procedure:
  - 1º phase:
    - The node broadcasts the list of nodes it can reach;
    - When a node receives it also broadcasts the list of nodes it can reach.
  - 2ª phase
    - Each node combines the information received and finds the "Surviving Clique";
    - Nodes not in "Surviving Clique" shut down.
- Downfalls:
  - Reduces performance and fault tolerance;
  - Can cause a complete system shutdown.

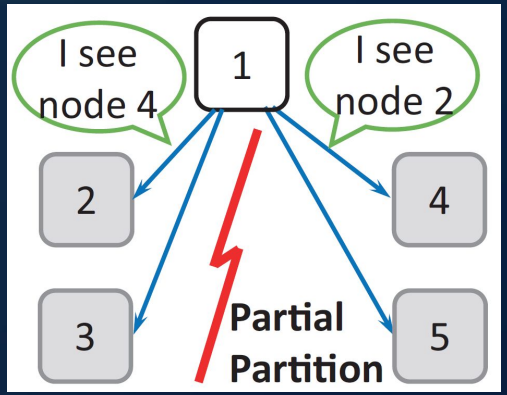
Speaker: Rafael

2 Dissecting Modern Fault Tolerance Techniques

Checking Neighbors' views [RabbitMQ, Elasticsearch]

- Main Idea:
  - Node S loses connection with Node D
  - Node S asks all nodes if they can reach node D
  - If any can, node S will either pause or create a complete partition

- RabbitMQ Approach:
  - Three Different Policies:
    1. Escalate to a complete partition;
    2. Pause;
    3. Pause if anchor nodes are unreachable.
  - Downfalls:
    - Policy 1 and 3 can lead to multiple inconsistent copies of the data;
    - Policy 2 may lead to pausing the entire system or a majority of nodes;



Node 1 is a bridge

Dissecting Modern Fault Tolerance Techniques

Checking Neighbors' views [RabbitMQ, Elasticsearch]

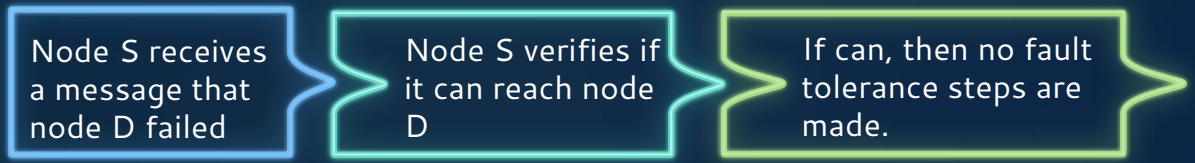
- Elasticsearch:
  - Only on its leader election protocol;
  - Approach:
    - If node S can’t reach the master it will check if any node can;
    - If any node can, the node S pauses;
  - Downfalls:
    - Can affect cluster availability drastically;
    - It is mechanism-specific.

Speaker: Rafael

3 Dissecting Modern Fault Tolerance Techniques

Failure Verification [MongoDB, LogCabin]

- Main Idea:



- Both MongoDB and LogCabin do this only on Leader Election protocol;
- To avoid leader elections when a majority of nodes can reach the master:
  - A node only participates in an election if it cannot reach the master.
- Downfalls:
  - Leads to unavailability of a large number of nodes;
  - It is mechanism-specific.

Speaker: Rafael

4 Dissecting Modern Fault Tolerance Techniques

Neutralizing Partitioned Nodes [MapReduce, HBase, Mesos]

- Main Idea: Neutralize one side of the partition
- Approach:
  - If the leader/manager cannot reach a worker node;
    - Neutralizes the node that it cannot reach;
    - Assigns its work to a new node.
- Downfalls:
  - Reduces performance and availability;
  - It is mechanism-specific.

Speaker: Rafael

Dissecting Modern Fault Tolerance Techniques

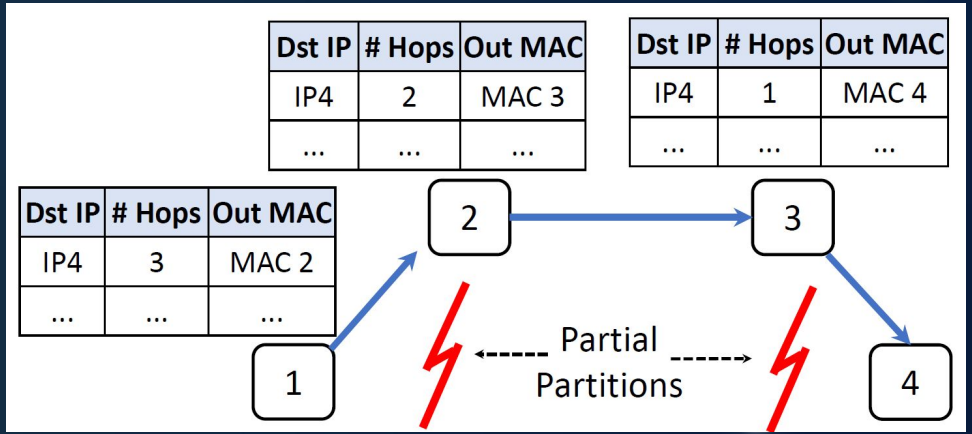
Final Thoughts

	Surviving Clique	Checking w/ Neighbors		Failure Verification	Neutralizing Nodes		Nifty
	VoltDB	Elasticsearch	RabbitMQ	MongoDB/LogCabin	MapReduce/HBase	Mesos	
Reduced Availability	X	X	X	X	X	X	
Complete Unavailability	X	X	X				
Complete Partition			X				
Double Execution						X	
Data Unavailability			X				
Scope	System-wide	Mechanism-specific	System-wide	Mechanism-specific	Mechanism-specific	Mechanism-specific	System-wide

Speaker: Sara

Nifty Design

- Follows a peer-to-peer design where nodes collaborate with each other;
- Has 4 main processes:
  - Connectivity monitoring
  - Recovery
  - Route deployment
  - Node classification



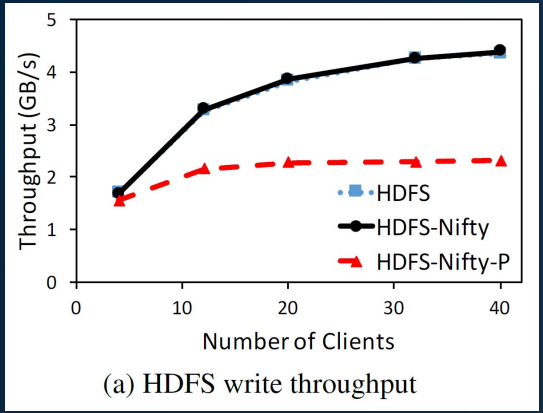
Speaker: Sara

Evaluation

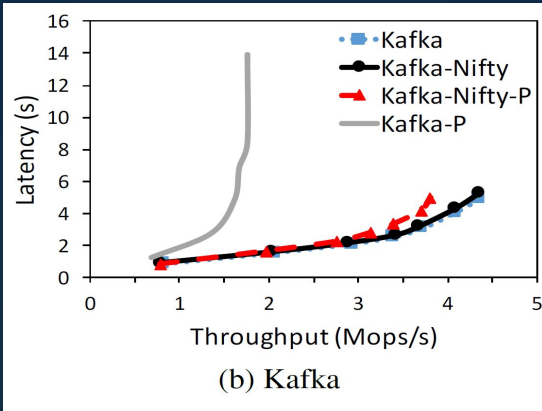
- Focused on 3 questions:
  - How much overhead does Nifty impose when there are no network partitions?
  - What is a system's performance with Nifty under a network partition?
  - What is the utility of Nifty's classification API?

Speaker: Sara

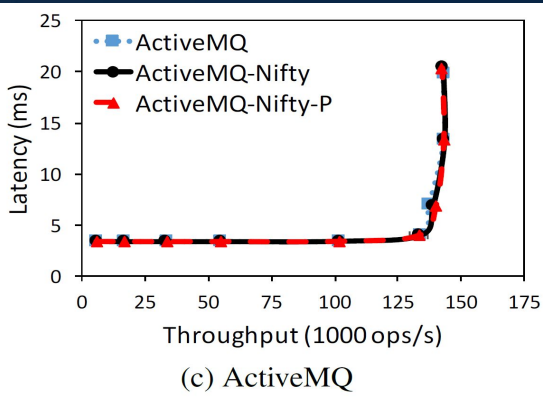
Evaluation



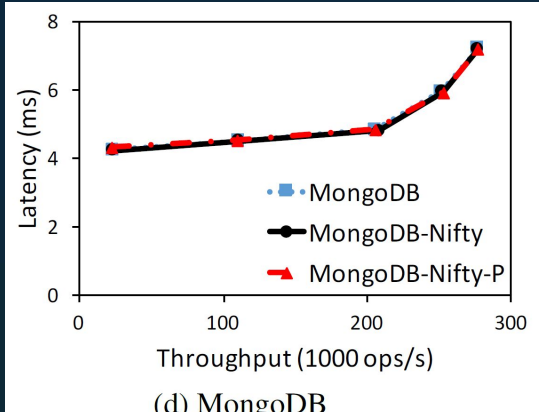
(a) HDFS write throughput



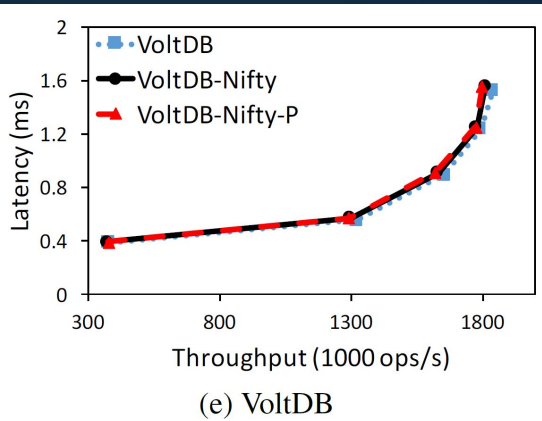
(b) Kafka



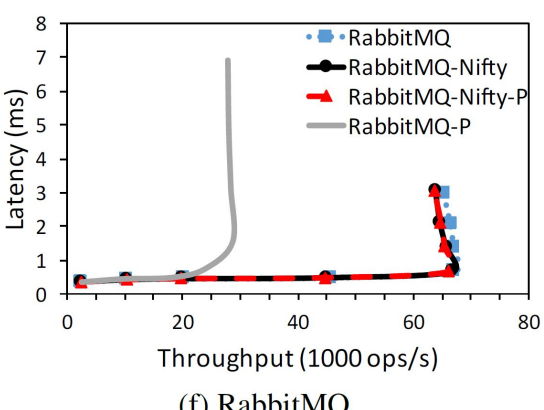
(c) ActiveMQ



(d) MongoDB



(e) VoltDB



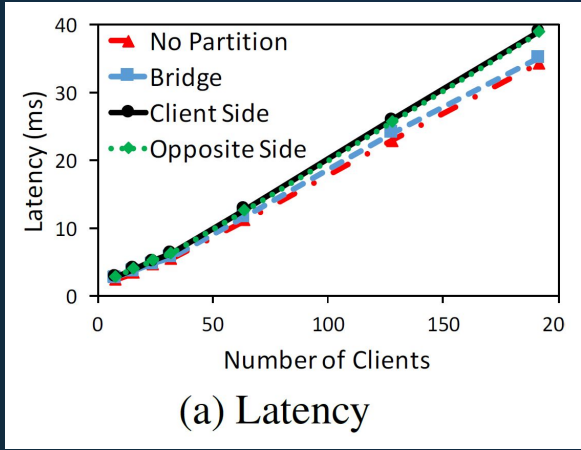
(f) RabbitMQ

Speaker: Sara

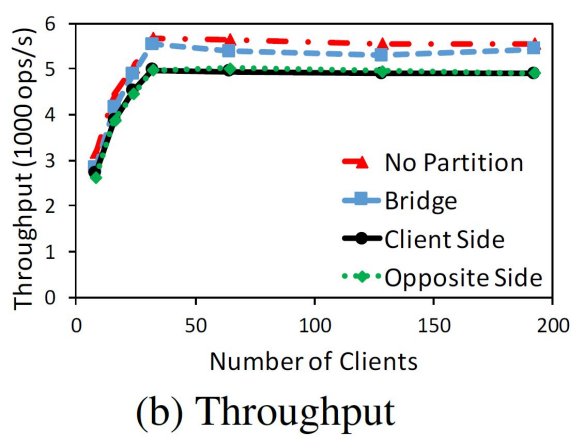
Evaluation

Nifty's classification API

- Allows to distinguish:
  - Bridge nodes;
  - Which side of the partition a node belongs to.
- Its use is system specific and can be used to improve performance when a partial network partition is present.



(a) Latency



(b) Throughput

Speaker: Sara

Related Work

- First work to characterize partial network partitions;
- Previous papers approach was more focused on complete network partitioning;
- Complete and partial partitioning are similar however, have key differences;
- Previous fault tolerance techniques are not able to prevent partial partitioning faults.

Speaker: Sara

Paper Reflection

Strengths

- The first work to characterize partial partitioning failures;
- It's clear;
- Considers different types of systems.

Weaknesses

- The presented solution is not scalable.

Speaker: Sara

Conclusion

- Paper provides insight to a unexplored topic;
- The existing solutions are not fool proof;
- Nifty causes a small overhead that allows a system to not lose availability;
- It is a topic and a solution worth exploring.