Toward a Generic Fault Tolerance Technique for Partial Network Partitioning

Mohammed Alfatafta, Basil Alkhatib, Ahmed Alquraan, Samer Al-Kiswany
Modern Networks are Complex

• Multiple data centers
• Large scale
• Variety of middle boxes
• Heterogenous hardware and software
• Softwarization

Catastrophic network failures are common [1, 2, 3, 4]

Partial partitions

Isolate a set of nodes from some, but not all, nodes in the cluster.
Partial partitions

Isolate a set of nodes from some, but not all, nodes in the cluster.

Impact: Confuses systems as nodes disagree whether a given node is up or down.
Outline

• What causes partial network partitioning?
• How do they impact systems?
• Are there any fault tolerance techniques?
• NIFTY: a generic fault tolerance technique
• Evaluation
Outline

• What causes partial network partitioning?
• How do they impact systems?
• Are there any fault tolerance techniques?
• NIFTY: a generic fault tolerance technique
• Evaluation
Causes of partial partitions

• Failure of additional links between racks [1,2]

• Network and Firewall misconfigurations [3]

• Network upgrades [4]

• Flaky links between switches [5]

Outline

• What causes partial network partitioning?
• How do they impact systems?
• Are there any fault tolerance techniques?
• NIFTY: a generic fault tolerance technique
• Evaluation
Methodology

• Study 51 high-impact partial partitioning failures from 12 systems.

• Study failure report, discussion, logs, code, and tests.

• Reproduce some of the failures.
Partial network partition – ActiveMQ
Partial network partition – ActiveMQ

Cluster
Master

Replica

Replica

Replica

APACHE
ZooKeeper

Heartbeating

Cannot see majority!

Cannot see master!
Partial network partition – ActiveMQ

I see no problem!!

Heartbeating

Complete cluster pause

Cannot see majority!
Partial network partition – ActiveMQ

- Complete cluster pause.
- Common vulnerability in Zookeeper deployments.

I see no problem!!

Cannot see majority!

Complete cluster pause

Heartbeating

Cannot see master!
Partial network partition – HBase

Master

Region server A

Region server B

User

Log A-1

Log A-2

Create a new log
Partial network partition – HBase

Master

Region server B

Region server A

User

Server A is down. Select another

Update metadata

Log A-1

Log A-2

Server A is down. Select another
Partial network partition – HBase
Partial network partition – HBase
Partial network partition – HBase

Master

Region server B

Region server A

Log A-1

Log A-2

Log B-2

Update metadata

User
Partial network partition – HBase

Master

Region server B

Region server A

User

Log A-1

Log A-2

Log B-2

Update metadata
Partial network partition – HBase

• Catastrophic: data loss.

• Easy to manifest: deterministic and requires a few events.
What is the impact of partial partitioning?

• **Catastrophic**: 75% (e.g., data loss or corruption).

• **Silent**: 84%.

• **Permanent**: 24% have lasting impact.
How easy are they to manifest?

- Partition only one node.

- No client access or a client access to one side: 60%

- Three or less events: 69%

- Deterministic

Surprisingly, easy to manifest failures cause catastrophic effects.
Other findings

• **Vulnerable mechanisms**: leader election, config. change, and replication

• **Testability**: reproducible on 5 nodes

• **Design flaws**: majority are due to design flaws
Outline

• What causes partial network partitioning?
• How do they impact systems?
• Are there any fault tolerance techniques?
• NIFTY: a generic fault tolerance technique
• Evaluation
Study of fault tolerance techniques

• Study the fault tolerance techniques of 8 popular systems.

• Study code patches of all studied failures.
Current Fault Tolerance Techniques

1. Graph-based connectivity monitoring (VoltDB)
2. Checking with neighbours (Elasticsearch, RabbitMQ)
3. Failure verification (MongoDB, Raft, Elasticsearch)
4. Neutralizing partitioned nodes (Mesos, MapReduce, HBase)
Current Fault Tolerance Techniques

1. Graph-based connectivity monitoring (VoltDB)

2. Checking with neighbours (Elasticsearch, RabbitMQ)

3. Failure verification (MongoDB, Raft, Elasticsearch)

4. Neutralizing partitioned nodes (Mesos, MapReduce, HBase)
Graph-based connectivity monitoring

Idea: Build and analyze a connectivity graph.

How it works:

• All-to-all heart beating
• On a partition: nodes exchange connectivity information
• Each node finds the largest fully-connected sub-graph
Graph-based connectivity monitoring

Idea: Build and analyze a connectivity graph.

How it works:

• All-to-all heart beating
• On a partition: nodes exchange connectivity information
• Each node finds the largest fully-connected sub-graph
• Nodes out of the sub-graph shut down
• If any data is lost, shut down the cluster
Graph-based Technique Shortcomings

• Unnecessarily shut down nodes.

• High chance of a complete cluster shutdown.

    Partitioning 20% of nodes often leads to complete cluster shutdown.
# Shortcomings

<table>
<thead>
<tr>
<th>Surviving Clique</th>
</tr>
</thead>
<tbody>
<tr>
<td>VoltDB</td>
</tr>
</tbody>
</table>
| Reduced Availability      | X  
| Complete Unavailability   | X  
| Complete Partition        |   
| Double Execution          |   
| Data Unavailability       |   
| Scope (System/Mechanism)  | S  

Shortcomings

All current fault tolerance techniques have severe shortcomings.

<table>
<thead>
<tr>
<th></th>
<th>Surviving Clique</th>
<th>Checking w/ Neighbors</th>
<th>Failure Verification</th>
<th>Neutralizing Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VoltDB</td>
<td>Elasticsearch/RabbitMQ</td>
<td>MongoDB/LogCabin</td>
<td>MapReduce/Hbase/Mesos</td>
</tr>
<tr>
<td>Reduced Availability</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Complete Unavailability</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete Partition</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double Execution</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Data Unavailability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Scope (System/Mechanism)</strong></td>
<td><strong>S</strong></td>
<td><strong>M/S</strong></td>
<td><strong>M</strong></td>
<td><strong>M</strong></td>
</tr>
</tbody>
</table>
Outline

• What causes partial network partitioning?
• How do they impact systems?
• Are there any fault tolerance techniques?
• NIFTY: a generic fault tolerance technique
• Evaluation
NIFTY

A Network partitioning fault tolerance layer (NIFTY)

Goals:
• System agnostic
• No changes to existing systems
• Negligible overhead

Insight: leverage existing monitoring techniques to detour traffic around partial partitions.
How NIFTY works

• Use heartbeats to detect partial partitions

• On a partial partition: detour packets through intermediate nodes

• Use distance vector routing

• Use OpenVSwitch to deploy routes on end nodes
How NIFTY works

Rerouting done through MAC address manipulation
How NIFTY works

Rerouting done through MAC address manipulation
How NIFTY works

Rerouting done through MAC address manipulation

• Simple

• Agnostic to system running atop of it

• Transparently masks partial partitions
Outline

• What causes partial network partitioning?
• How do they impact systems?
• Are there any fault tolerance techniques?
• NIFTY: a generic fault tolerance technique
• Evaluation
Evaluation

• What is Nifty’s overhead?

• How systems perform under a partial partition?

• How does nifty scale for large clusters?

• What is the utility of Nifty’s classification API?
Evaluation

• What is Nifty’s overhead?

• How systems perform under a partial partition?

• How does nifty scale for large clusters?

• What is the utility of Nifty’s classification API?
Evaluation setup

• Measure the impact of Nifty on 6 systems.

• 40 nodes in Cloudlab Utah cluster.
Evaluation: Overhead - RabbitMQ

![Diagram showing RabbitMQ and NIFTY brokers with latency graph]

- **Broker**: RabbitMQ
- **NIFTY**: Multiple instances of NIFTY connected to the RabbitMQ broker.

![Graph showing latency with RabbitMQ and RabbitMQ-Nifty]
Evaluation: Performance with a Partition

![Diagram showing network topology and performance metrics related to brokers and RabbitMQ]

Latency (ms) vs. Throughput (1000 ops/s) for RabbitMQ with and without a partition.
Evaluation: Performance with a Partition

Broker

Broker

Broker

Broker

RabbitMQ

RabbitMQ

RabbitMQ-Nifty

RabbitMQ-Nifty-P

Latency (ms)

0 1 2 3 4 5 6 7 8

0 20 40 60 80

Throughput (1000 ops/s)

RabbitMQ
Evaluation: Performance with a Partition
Evaluation: Performance with a Partition

NIFTY has a negligible overhead.

Effectively mask partial partitions.
Conclusion

• First comprehensive study of partial partitioning failures:
  • Failures are catastrophic
  • Failures are easy to manifest
• First study of current fault tolerance techniques:
  • All current techniques have severe shortcomings
• Built Nifty
  • Simple
  • Transparent
  • Low overhead
Thank you!

Source code available at: https://wasl.uwaterloo.ca/projects/nifty/