Ricardo Grade

Cover

Hello! My name is Ricardo Grade, my group colleagues are Sara and Rafael, and we will present to you the Paper: Toward a Generic Fault Tolerance Technique for Partial Network Partitioning.

Introduction

This work focuses its study on a peculiar type of network failure, called Partial Network Partitions, which, unlike Complete Network Partitions, interrupts the communication between some, but not all, nodes in a cluster, so that they can still communicate over a longer path, possibly passing through several nodes, for example, as you can see in the figure, Group 1 has stopped communicating with Group 2, but can still communicate with it through Group 3 and vice versa.

We will present the results found by this analysis, we will understand the impact and characteristics of this type of failures, then we will explore the fault tolerance techniques implemented in these systems and identify their shortcomings, finally, we will present Nifty, a generic fault tolerance mechanism for Partial Network Partitions.

Utility

Network partitions cannot be avoided, they will always happen in large-scale clusters, so it is important to find a way to tolerate them to avoid catastrophic failures and at the same time continue to provide high availability.

Most of the mechanisms to prevent these failures contain many shortcomings, as you will see in our presentation, and they can also completely interrupt the availability of a cluster.

Analysis of Partial Network-Partitioning Failures

To understand the characteristics and impact of this type of failure, 51 reports found in 12 popular systems were analysed.

Among these systems:

- Kafka, ActiveMQ and RabbitMQ which are the most popular open-source messaging systems.
- MapReduce, HDFS and HBase which are the core of the Hadoop platform.
- Elasticsearch which is a popular search engine.
- MongoDB which is a popular database.

The reports analysed are relatively recent and of medium to high severity.

Limitations

- Although the selected systems were chosen because they have different properties, for example, in terms of levels of consistency or replication mechanisms, the results cannot be generalized to systems that were not covered in this study.
- The tickets analysed are a sample, it is estimated to have a 95% confidence level and an error margin of 13%.
- As stated, the tickets that make up this sample are all marked as high impact, since low priority tickets have been discarded, which can skew the results.
- In addition, the reports were analysed by different teams and discussed by everyone before reaching a verdict.

Finding 1
The most common failures identified in this analysis are catastrophic, which means that it leads to a system failure or violates the system's guarantees.

Data loss is the most common, followed by system unavailability.
Nevertheless, there is a significant percentage of reduced system availability, largely due to the continued leader-election thrashing.

Finding 2
Although partial partitioning failures are not uncommon and have a dangerous impact, most systems have ambiguous mechanisms, or do not even have them at all.

Finding 3
The mechanism most vulnerable to partial network partitions is the leader-election, which usually leads to the election of a leader on each side of the partition.

The 2nd is the configuration change, in RabbitMQ two nodes containing different membership views can lead to a complete system crash.
The 3rd most affected is the replication protocol, which due to the leader's stepping down is not immediate, a client could write to the leader and later read outdated data from other replicas.

Finding 4
Some mechanisms in the occurrence of a partition were preventing access to one side of it.
However, these studies show that only access to one side of a partition, or even lack of access can still lead to failures due to background operations.

Finding 5
These studies show that failures can arise from the occurrence of a very small number of events, which is extremely alarming, since a deployed system interacts with many clients.
As can be seen in the table, only one Partial Partition event can lead to failures.

Finding 6
One 3rd of the failures are manifested by the isolation of any node, regardless of the role it plays.
Regarding the failures that arise from the isolation of a node that plays a specific role, the most common is the leader.

Finding 7
Most failures are deterministic, which means that whenever the specific sequence of events occurs, the failure also occurs. And a 3rd of the failures has known time constraints, such as the period before concluding that a node has failed.

Finding 8
In this context, the design flaws are those whose fix significantly changes the implemented protocol.
According to this table, almost half of the failures are caused by flaws of this type, which means that system designers do not give this problem the deserved importance as it requires.

Finding 9
More than three quarters of the faults can be reproduced with a cluster of only 3 nodes, which when increased to 5 nodes can reproduce any studied fault. This shows the ease of reproducibility and, therefore, the danger of such failures.

Rafael Figueiredo

Dissecting Modern Fault Tolerance Techniques

To understand the current techniques used to solve partial partitions, it was studied six of the systems previously talked that managed to change their system to tolerate partial partitions, and also two new additional systems (VoltDB and LogCabin).
It was then identified four approaches to deal with partial partitions: Identifying the Surviving Clique, Checking Neighbours' views, Failure Verification and finally Neutralizing Partitioned Nodes.

Identifying the Surviving Clique

The main idea is that when a partial partition happens, the system identifies the maximum clique of nodes, which is the maximum subset of nodes that are completely connected, and all nodes that do not belong to that subset are shut down.
This is the approach that VoltDB implements, where every node in the system sends periodically heartbeats to all nodes, and if a node does not receive all heartbeats, it suspects that a partial partition occurred and starts the recovery procedure, where every node will broadcast the nodes that it can reach, which allows each node to construct a graph and find the maximum clique of nodes called The surviving clique, if a node finds that does not belong on the surviving clique, it shuts itself down.
In order to guarantee that no data was lost, at least one replica of every shard needs to continue to exist, (Being a shard a partition of data of the database). If it does not exist, the entire system shuts down.
In the figure on the right, we can see that a partial partition disrupted the communication between nodes, and only the clique with the most nodes stay alive.
There are two main problems with this approach:

- The 1st is that it is possible to shut down up to half of the cluster nodes, reducing performance and fault tolerance.
- The 2nd is that a complete system shutdown can occur.

Checking Neighbours' Views

The main idea is that when a node called S loses its connection to a node called D, S then asks all nodes in the cluster whether they can reach D, and if any node can, then the cluster has a partial partition, and S will either disconnect from all nodes that can reach D or pauses its operation.
Both RabbitMQ and Elasticsearch follow this approach.
RabbitMQ has three policies, the policy used depends on its configuration.

- **Escalate to a complete partition**, where S disconnects from any node that can reach D, which creates a complete partition.

- **Pause**, where to avoid inconsistency, once a node discovers the partial partition it pauses its activities, and it resumes when the partial partition heals.

- **Pause if anchor nodes are unreachable**, where exists a subset of nodes that act as anchor nodes, and a node only pauses when it cannot reach any anchor nodes, this could lead to complete partitions if the anchor nodes become partially partitioned and could also lead to pausing all nodes if all the anchor nodes are isolated.

Downfalls:
The 1st and 3rd policy can change a partial partition to a complete partition, which may lead to inconsistent copies of the data. The 2nd policy may lead to pausing the entire system, or a majority of nodes.
An example of this is the figure on the right, where if every node except the bridge detects that it cannot reach some nodes, all nodes pause except the bridge.
Elasticsearch uses this approach on Leader Election, which consists of if a node called S cannot reach the leader it contacts all nodes to check whether they can reach the leader, if at least one can, then S pauses.
Exists two major problems with this approach:

- The 1st is that it can affect cluster availability severely, as all nodes that cannot reach the leader pause. In the worst case, it can cause a complete cluster unavailability.
- The 2nd is because it is mechanism specific. To use this approach broadly, designers must design fault tolerance techniques for every mechanism in the system and understanding the interaction between them.

Failure Verification

Both MongoDB and LogCabin use this approach on its leader election, which consists of when a node called S receives a message that a node called D failed, the node S will verify if it can reach D before taking any fault tolerance steps.
Another problem that occurs is when a leader is on one side of a partial partition but can still reach the majority of nodes, the nodes on the other side of the partition will unnecessarily call for the leader election, to avoid this, when a node receives a call for election it 1st verifies that the current leader is unreachable, and only participate if it cannot reach the leader.

This approach has two major problems.

- The 1st is that it leads to the unavailability of a large number of nodes.
- The 2nd is that it is mechanism specific.

Neutralizing Partitioned Nodes

This approach tries to neutralize one side of the partition, HBase, MapReduce and Mesos use this approach with different implementations, but the approach follows the same idea, which is when the leader cannot reach a worker node, it neutralizes the node and assigns its work to other workers.

This approach has two major problems:

- The 1st is that it leaves the nodes on one side of the partition idle, which reduces system performance and availability.
- And 2nd, it is mechanism specific.

Final Thoughts

As you can see in this table all approaches severely affect system availability.
Also, some approaches cannot provide system-wide fault tolerance, and the approaches that are able to provide, can lead to a complete system shutdown or significant loss of system capacity.
This realization motivated the creation of Nifty a system-wide fault tolerance technique that overcomes all these problems.

Sara Machado

Nifty Design

To overcome the limitations of fault tolerance techniques and the catastrophic failures partial network partitioning can cause, the paper authors presented Nifty, the network partitioning fault tolerance layer.
This layer follows a peer-to-peer design where the nodes in a cluster collaborate to monitor the cluster availability. When a partial network partition is detected, Nifty reroutes the traffic between the partitions to go through the nodes that connects them, called bridge nodes.

Nifty can be divided in 4 main processes:

- **Connectivity monitoring:** Each node contains a distance vector with the distance in number of hops to the other nodes. Heartbeat messages are sent between all nodes in the cluster. If one node misses 3 heart beats from another node, it assumes that the connection between them is lost.

- **Recovery:** To allow for the reroute of the messages, each node piggybacks their heartbeat messages with their distance vector. This allows nodes to build and maintain a routing table, based on the information received on the heartbeat messages, used to update their distance vector whenever necessary.

- **Route deployment:** As we can see in the image there are two partial network partitions. Node 1 cannot reach nodes 3 and 4. And node 4 cannot reach node 1 and 2. For node 1 to send a packet to node 4, it sends it with node 4 IP and sets the outgoing mac to the node set on the routing table, the mac of node 2. Node 2 after receiving the packet with node 4 IP does the same process and reroutes the packet to node 3. Node 3 has a direct connection to node 4 so it sends the packet accordingly.

- **Node classification:** Nifty identifies which nodes are on the same side of the partition or are bridge nodes which allow for a system using Nifty to use this information to its advantage, like reducing the amount of data forwarded through bridge nodes.

Evaluation

The evaluation of the paper focus 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?

Regarding the 1st two questions we have the following results:

- Presented on the graphs we can see the results of tests performed on multiple of the previously mentioned systems:
    - In blue the system is running without any changes.
    - In black the system is running with Nifty without partial network partitions.
    - In red the system is running with Nifty in the presence of a partial network partition.
    - In grey and only present in 2 graphs, graph (b) and (f), a system running without Nifty and in the presence of a partial network partition.
    - The other graphs do not present such evaluation, since whenever a partial network partition was present these systems would shut down or completely pause, not being able to support this fault.

- It is possible to see that the overhead caused by the presence of Nifty, is not significant compared to the benefits it might present in cases where the system would stop or pause.

- The scalability of the system was tested with 100 nodes, however real-life systems may present a much larger number of nodes. Therefore, the presented system does not scale for those cluster sizes.

The last question of the evaluation is meant to shed a light on the utility of having a node classification API. This API can distinguish in which side of the partition the nodes are present and find the nodes that are bridges between partitions.
The way a system can use this API is system-specific, in the sense, that depending on how the system functions the API can have different uses. For example, a system that is leader based may benefit, in the presence of partial partition, to change the leader to the bridge node, reducing the number of messages needed to be exchanged between partitions. As it is shown on the presented graph.

Related Work

The paper is the 1st to characterise the problems partial network partitioning may cause and propose a valid solution. There were previous works that approached partitioning failures however, their focus was on complete partitions. Despite the similarity with complete network partitioning, where both are easy to manifest and may cause catastrophic failures, they are fundamentally different, so, the fault tolerance techniques applied to complete partitioning cannot be used to solve partial partitioning faults.

Paper Reflection

- This is 1st work to characterize these failures and explore fault tolerance techniques for Partial Partitioning failures.
- Clearly presents the problem.
- Considers different types of systems which can be affected by partial network partitioning.
- However, the presented solution is not scalable so it might not be applicable in most real case scenarios.

Conclusion

In conclusion, the paper provides an insight to a topic that has not been yet explored. Even though complete network partitioning is more common, partial network partitioning can cause similar effects and be more subtle, by affecting only a smaller portion of the cluster. Applying a transparent layer like Nifty provides a fault tolerance technique that prevents a system from stopping or pausing, improving a system's availability. Despite its flaws Nifty proves that this concept it is worth exploring in order to protect systems against partial network partitions.

Thank you!