Scalable Consistency in Scatter

Presented by:
Yaonan Huang
What is Scatter?

• “This paper describes the design, implementation and evaluation of Scatter, a scalable and consistent distributed key-value storage system”
  
  - from paper’s Abstract

• In a nutshell, it is a distributed hash-table (DHT). Just like Kademlia that tortured us in our first project.
Q: Why do we need Scatter?
So far, we have seen two types of distributed system.

**DHT**
- Examples: Kademlia, and Chord

**Coordination Services**
- Examples: Chubby, an ZooKeeper
Both have their own advantages

• DHT achieve **scalability** by building on a highly flexible structure, and processing look up recursively.

• **So that update may happen only locally**

• Coordination Services ensure **consistency** by following rigorous distributed algorithm that provides strong consistency.

• **For example. ZooKeeper:** atomic broadcast protocol,

• **Chubby uses Paxos distributed consensus algorithm**
However, none of them can achieve both scalability and consistency
Inconsistency in DHT

(a) Key Assignment Violation

(b) Routing Violation
Limit on scalability of coordination services

• Every update to a replicated data requires communication with quorum of all nodes participating in service

• Therefore, As the number of nodes increase, scalability is severely limited
Solution? Scatter!
Design of Scatter, some intuition

- Scatter achieve **scalability** by following a similar macro-structure as DHT.
- While the building blocks of the network is self-managing **sets** of nodes, (called “groups”) instead of individual nodes.
- **Consistency** is achieved by applying replicated state machine mechanism internally.
Inconsistency and Unscalability?

• Inconsistency is not a problem, group of nodes does not fail as a whole.

• Limit of scalability is not a problem because size of each group is bounded.
How about “groups” themselves? Should they be static?
No, they shouldn’t

• Because static set of groups will be inherently limited in many ways:
• 1. burst of failure may cause a whole group to lost its functional quorum
• 2. static groups does not have the ability to adapt to dynamic conditions
Multi-group operations

• Therefore, we need to support multi-group operations to ensure adaptability of groups.

• 1. split: partition a group into two groups

• 2. merge: replace two neighboring groups by one new group

• 3. migrate: move individual nodes from one group to another group

• 4. repartition: change the key-space partitioning between two neighboring group
How do we implement these operations?
Define Consistency

• Because the building block of Scatter network is group instead of individual node, we need to adjust the definition or requirement of consistency of Scatter on a overlay-level.

• Basic: **Groups** (instead of individual node) that are adjacent in overlay agree on partitioning of the key-value between them. *Corresponding to Key Assignment violation*

• Additional: group can always reach its adjacent groups. *In other words, insurance of availability.*
Enforce Consistency by Nested Consensus

• Some naming and notations:
  • 1. The group initiating a transaction: **coordinator** group
  • 2. The other groups involved: **participant** groups
• 1. Coordinator replicates the decision to initiate the transaction
• 2. Coordinator broadcasts a transaction prepare message to participant
• 3. Participant decides whether or not to commit the proposed transaction and replicates its vote
• 4. Participant broadcasts a commit or abort message to coordinator
• 5. Coordinator decide transaction and replicates the response from participant
• 6. Coordinator broadcast outcome of transaction to participant
• 7. Participant replicate transaction outcome
• 8. Coordinator and participant executes the steps of proposed transaction
Example: splitting to two groups

![Diagram showing the process of splitting a group into two groups (G1, G2, G3) over time (t0, t1, t2, t3). The process involves preparing (PREPARE) and committing (COMMITTED) transactions within each group.](image-url)
Now we have scalability, consistency. How does it function? *In other word, how does it store key-value information?*
After we partition key-space by groups. We further divide them into individual nodes within a group.
Policy: An important property of Scatter’s design is that the separation of policy from mechanism.
Resilience.

- Scatter must be resilient to node churn as nodes join and depart the system unexpectedly. And a scatter group can still guarantee availability with up to half of the nodes fails.
- To improve resiliency, a group will merge with adjacent group if portion of live nodes fail below a threshold.
- However, because small groups are more efficient, there is a trade-off between availability and efficiency.
- Intuition: Dancing along the edge is fun, but dangerous
Latency

• Client Latency can be reduced by placing new node to group where latency with neighboring groups are low. This is implemented by policy below.

• Latency-optimized join policy: the joining node will randomly selects k groups and pass a no-op operation, which allows the node to estimate the performance of operating within each group.

• Intuitively, it is like placing a spy to observe the behavior of k groups, and join the optimal one.
Conclusion

• Scatter achieves scalability by applying a DHT network on high level, while ensures consistency by enforcing coordination service within each group.
Thank you