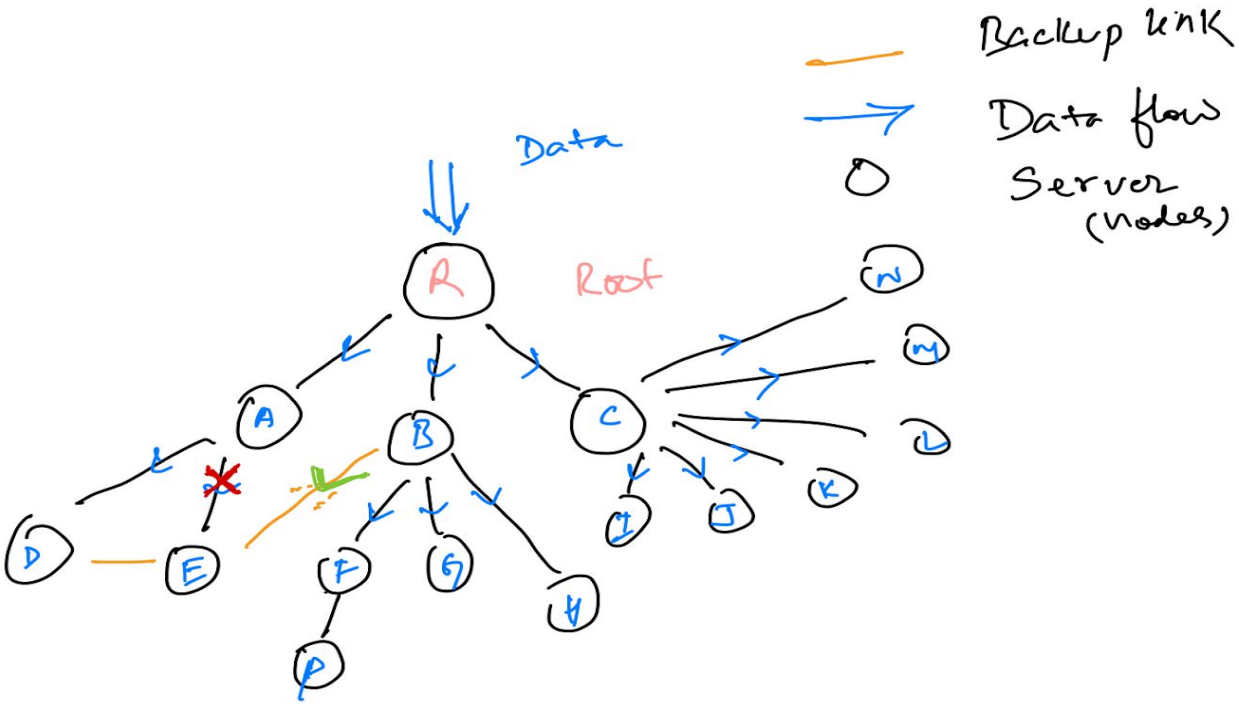
Fault-tolerant Distributed Spanning Tree

By Tushar Gautam

Table of Contents

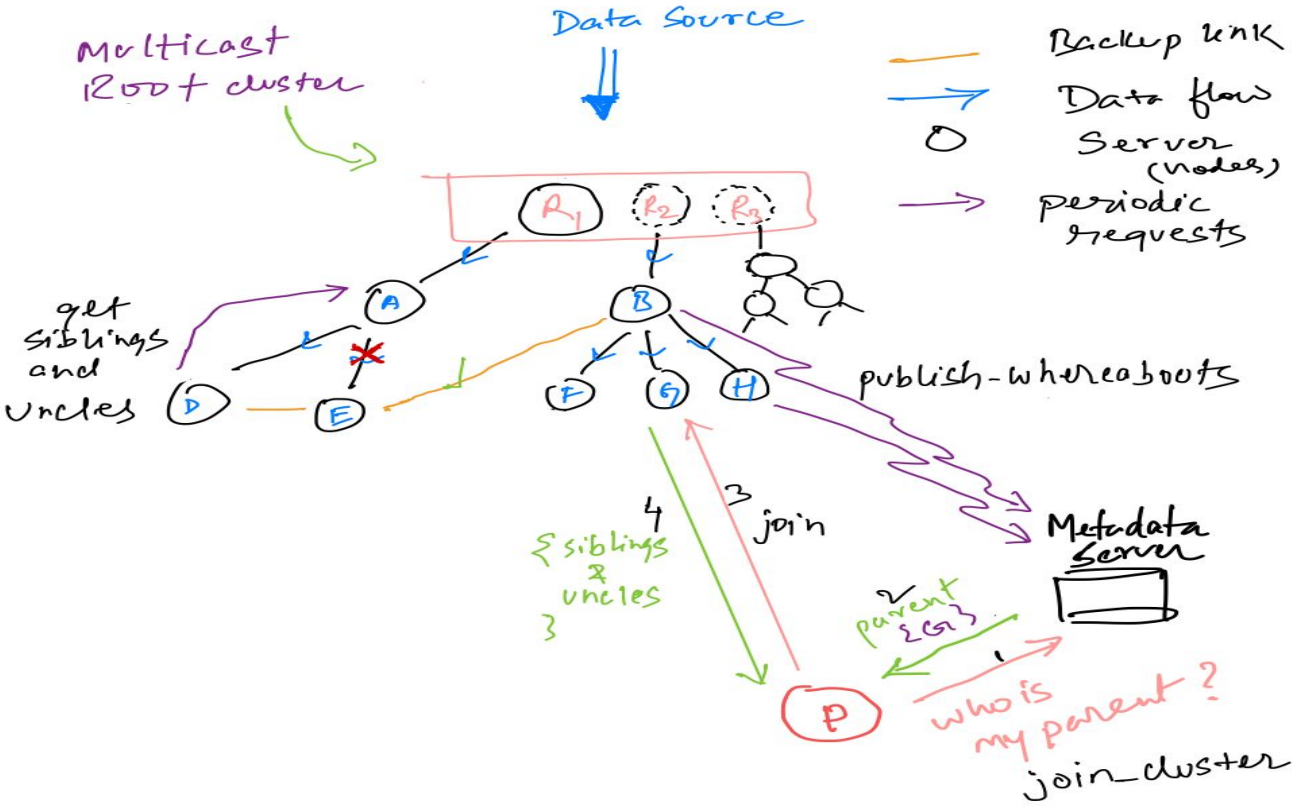
Topology


Backup link
Data flow
Server (nodes)
Data
R
Root
A
B
C
D
E
F
G
H
I
J
K
L
M
N
P

Topology

_id	node_id	parent_id	depth	children_count	capacity_left	endpoint_uri
638eeb55ce3de3c1ecc61774	4e74e42c-4654-4ce9-8ed4-4d5a0aa590f5		1	2	0	172.24.0.5:53800
638eeb67ce3de3c1ecc61775	9e92d2e4-d5a7-4728-8a69-2881b73db89f	4e74e42c-4654-4ce9-8ed4-4d5a0aa590f5	2	2	0	172.24.0.6:59444
638eeb67ce3de3c1ecc61776	e2ab2477-145b-4797-990e-232cc060ba12	4e74e42c-4654-4ce9-8ed4-4d5a0aa590f5	2	2	0	172.24.0.7:56830
638eeb7cce3de3c1ecc61777	ff2fe69e-8d03-4c0e-9067-ad6c0851bcc4	9e92d2e4-d5a7-4728-8a69-2881b73db89f	3	2	0	172.24.0.9:40798
638eeb7cce3de3c1ecc61778	f0e8e5ba-3ce2-480b-9f71-6f6e6f0f9b29	9e92d2e4-d5a7-4728-8a69-2881b73db89f	3	1	1	172.24.0.8:55908
638eeb7cce3de3c1ecc61779	8f1517b0-d717-4d34-918e-02a2adc02664	e2ab2477-145b-4797-990e-232cc060ba12	3	0	2	172.24.0.12:41562
638eeb7dce3de3c1ecc6177a	ef936160-50e4-4668-a33e-0fdb079d5e78	e2ab2477-145b-4797-990e-232cc060ba12	3	0	2	172.24.0.10:51290
	642fd195-2848-44bf-99f7-d848af50bdda	ff2fe69e-8d03-4c0e-9067-ad6c0851bcc4	4	0	2	172.24.0.14:41252

Architecture



Protocols

- Infrastructure layer: gRPC for intra-node cluster report exchange.
- Application layer: Media Streaming (RTMP), Config Deployment (QUIC) etc.

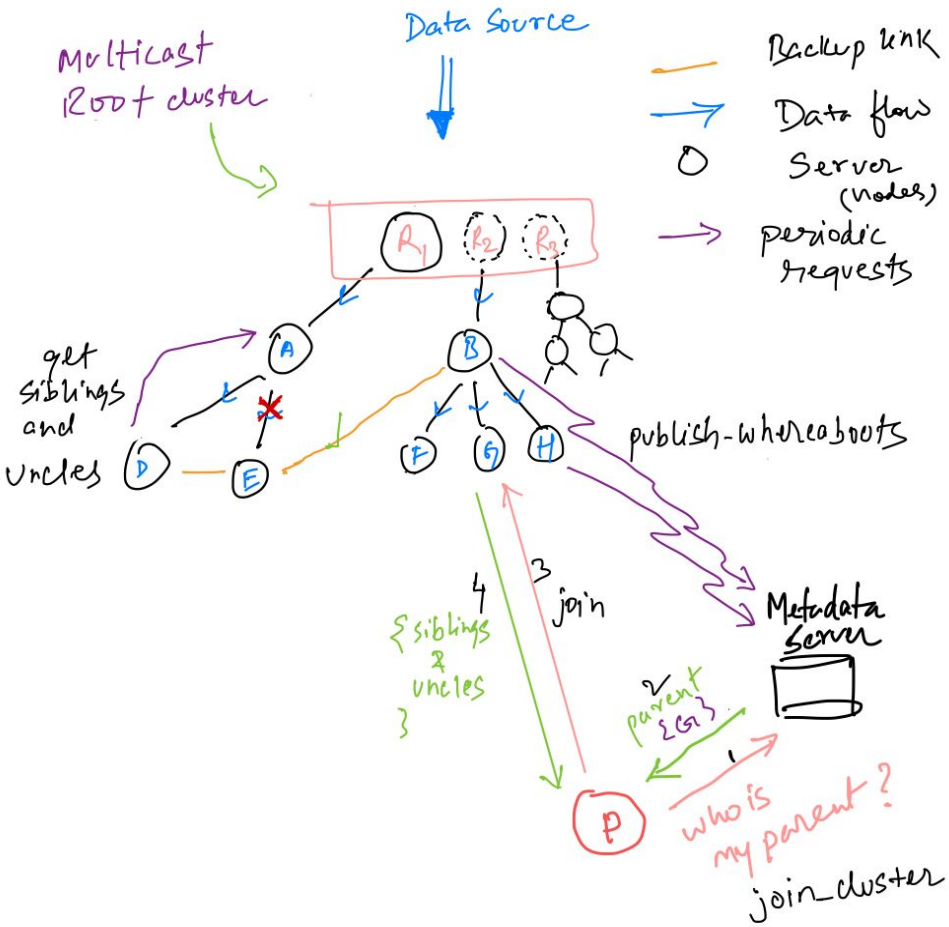
Protocols

- **Infrastructure layer: gRPC for intra-node cluster report exchange.**
- Application layer: Media Streaming (RTMP), Config Deployment (QUIC) etc.

Protocols

APIs

- **Data Flow**
  (Application Layer)
  - accept_data
  - forward_data



Protocols

APIs

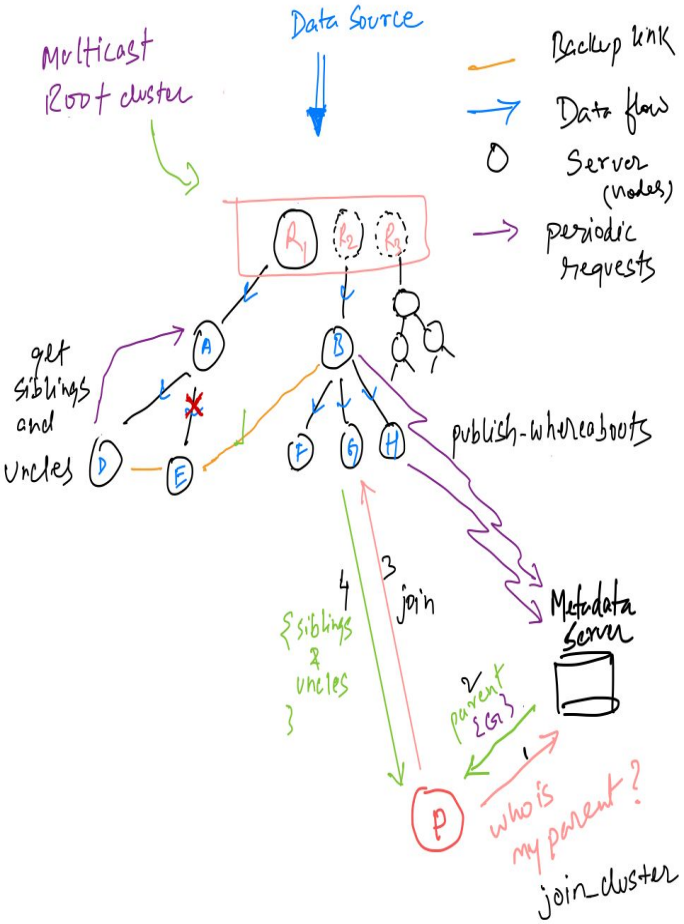
- **Metadata Service**
  (Metadata about self)

```
service MetadataService {
    rpc JoinCluster(JoinClusterRequest) returns (JoinClusterResponse);
    rpc PublishWhereabouts(WhereaboutsRequest) returns (WhereaboutsResponse);
}

message JoinClusterRequest {
    int32 capacity = 1;
    optional string node_id = 2; // existing nodes post node/parent failure recovery
}

message JoinClusterResponse {
    int32 status = 1;
    string node_id = 2;
    Node parent = 3;
    int32 depth = 4;
    optional string message = 5;
}
```

```
message Node {
    string node_id = 1;
    optional string endpoint_uri = 2;
}
```



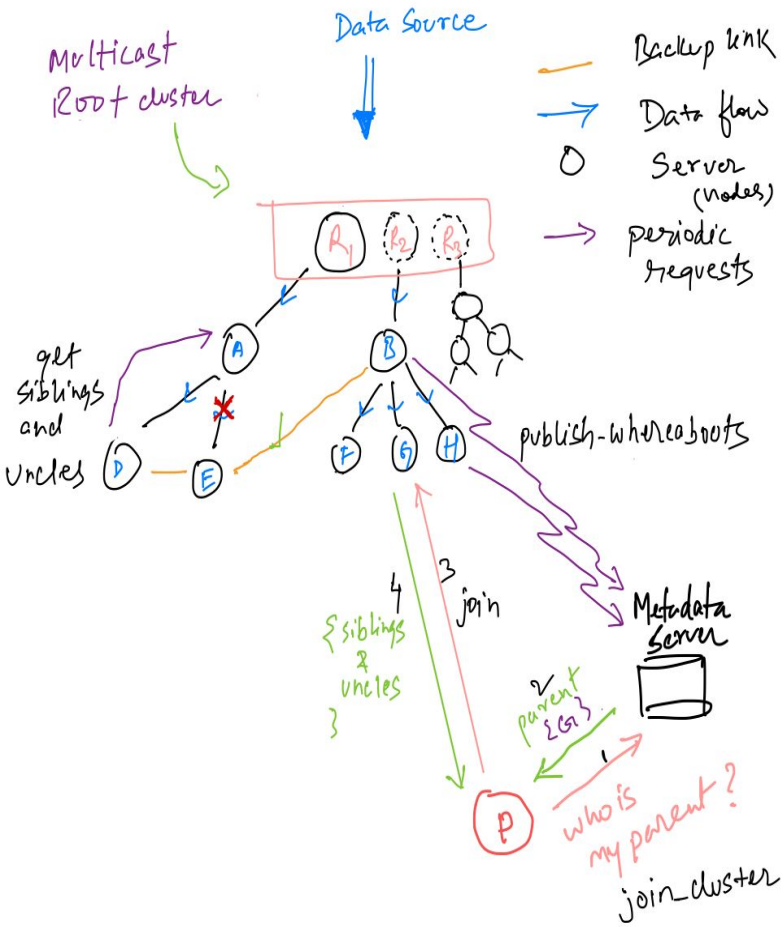
Protocols

APIs

- **Node State**
  (Metadata about self)

```
class NodeState {
    constructor(node_id, depth, parent_endpoint, parent_id) {
        this.node_id = node_id;
        this.depth = depth;
        this.parent_endpoint = parent_endpoint || "";
        this.parent_id = parent_id || "";
        this.grand_parent = "";
        this.grand_parent_endpoint = "";

        // The children and siblings to be populated from peer discovery
        // Cache stored as node_id -> Node(node_id, endpoint_uri)
        this.children = new NodeCache({ maxKeys: config.get(`${CONFIG_ROOT}.capacity`) });
        this.siblings = new NodeCache();
        this.uncles   = new NodeCache();
    }
```



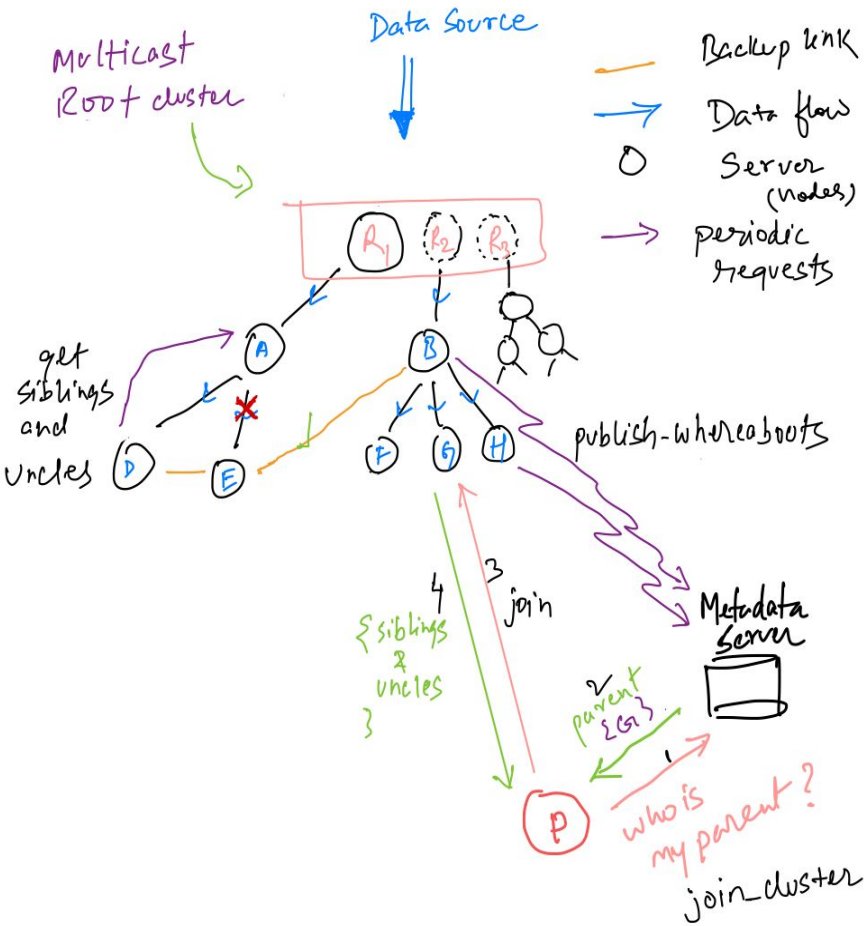
Protocols

APIs

- Metadata Service
  - publish_whereabouts
    (periodic health reporting to MDS)

```
message WhereaboutsRequest {
    string node_id = 1;          // node's unique uuid
    string parent_id = 2;        // parent node
    int32 depth = 3;             // tree depth
    int32 children_count = 4;    // current children count
}

message WhereaboutsResponse {
    int32 status = 1;            // HTTP return code 2xx, 4xx etc
    optional string message = 2;
}
```

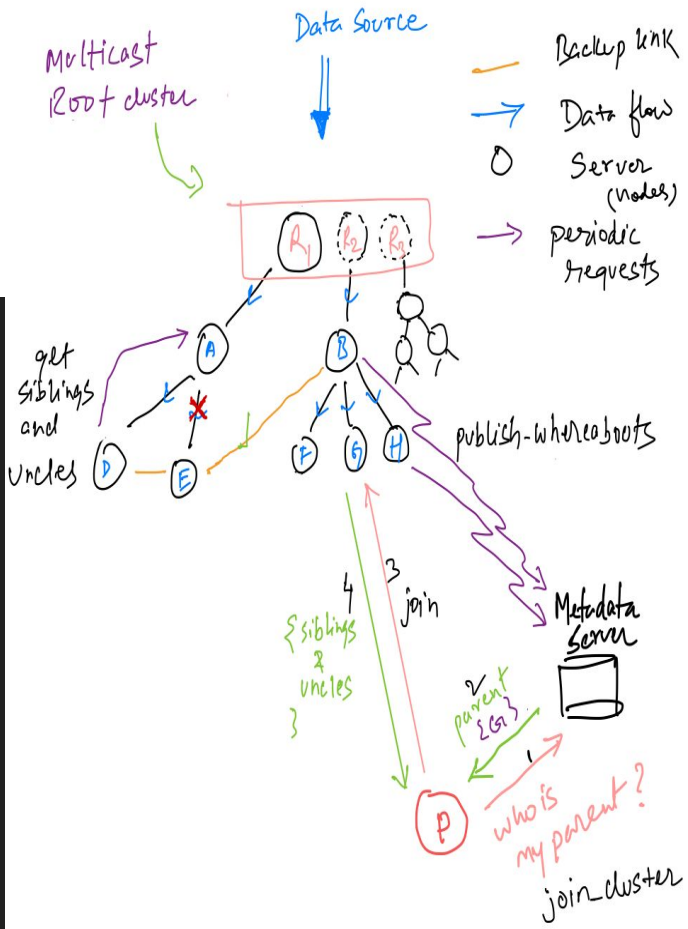


Protocols

APIs

- **Node Manager Service**
  get_siblings_and_uncles
  (periodic peer discovery)

```
service NodeService {
    rpc JoinParent(JoinParentRequest) returns (JoinParentResponse);
    rpc GetSiblingsAndUncles(SiblingsAndUnclesRequest) returns (SiblingsAndUncles);
}
message SiblingsAndUnclesRequest{
    string node_id = 1;
}
message SiblingsAndUncles {
    repeated Node siblings = 1;
    repeated Node uncles = 2;
    Node grand_parent = 3;
};
```



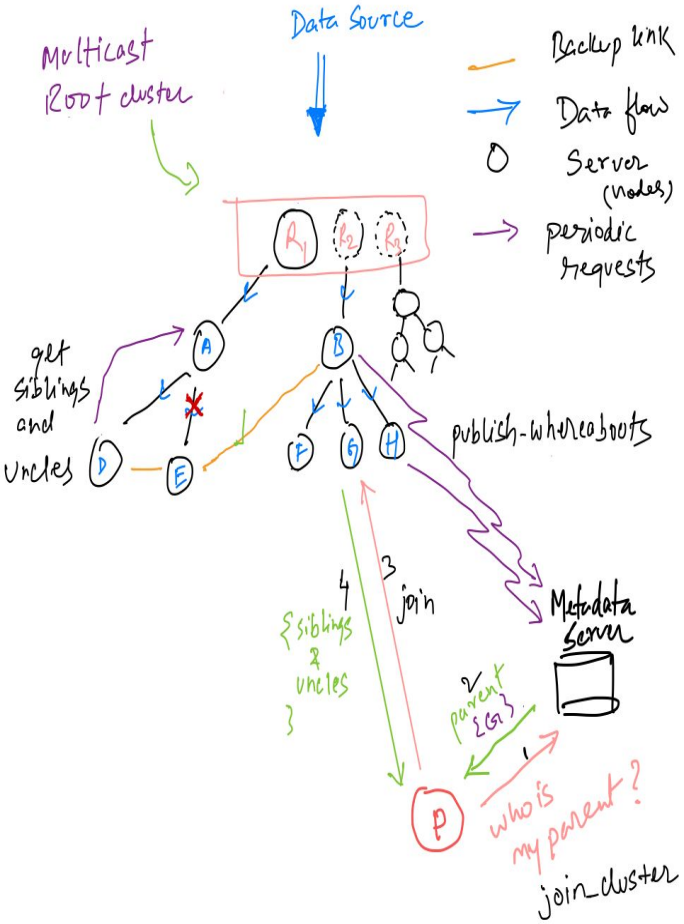
Protocols

APIs

- **Node Manager Service**
  join_parent
  (join assigned parent node)

```
message JoinParentRequest {
    Node child = 1;
};
message JoinParentResponse {
    int32 status = 1;
    string message = 2;
    SiblingsAndUncles siblings_and_uncles = 3;

};
```



Protocols

APIs

- **Node Manager Service**

  get_new_parent

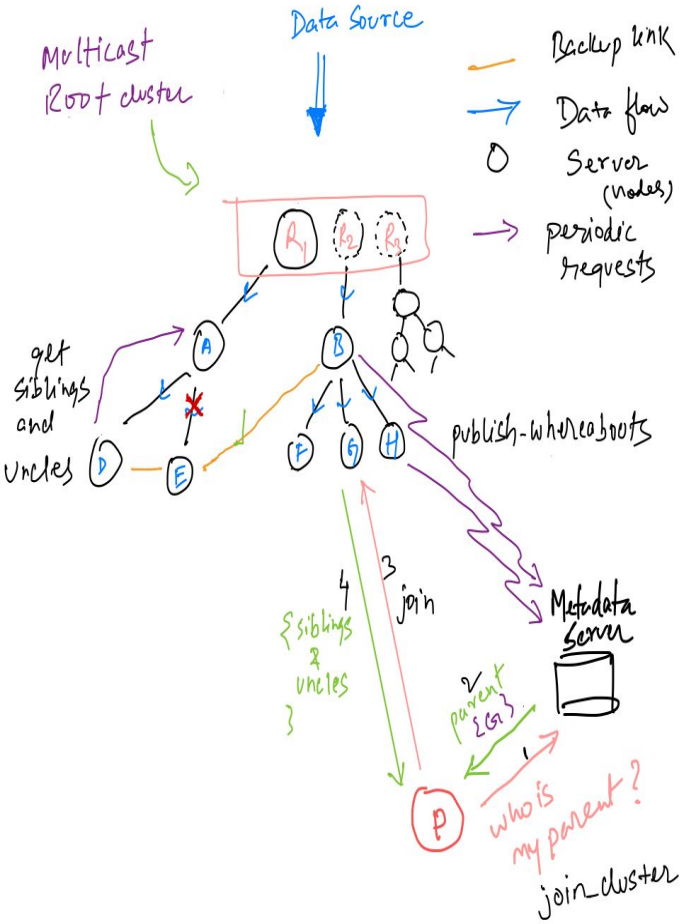
  - Compute new parent based on the NodeState during failure recovery.

  trigger_parent_switch

  - Event-driven switching to new parent as failure recovery.

    parent = get_new_parent (NodeState)

    JoinParent() // join rpc to new parent



Demo

Future Scope

Reference

- Protocol Buffers
- gRPC

Thank you!