State Machine Representation

States:

Idle:

* Initial state where the system is waiting for a new campaign creation or article processing request.

Campaign Creation:

* Transitions to this state when a new campaign is created using the POST /createCampaign API endpoint.
* Receives the topic information and initializes the campaign with default MDims.

Article Processing:

* Moves to this state upon receiving a request to process an article through the POST /processArticle API endpoint.
* Extracts MDims from the article content, aggregates them with existing values, and initiates the Paxos consensus mechanism.

MDim Retrieval:

* Enters this state when a request to retrieve MDims is received via the GET /readMDims API endpoint.
* Fetches the MDims for a given campaign and date from the JSON KV Store.
* Returns the MDims to the requester.

Consensus Handling:

* Represents the Paxos consensus mechanism responsible for synchronizing updates across nodes.
* Enters this state during campaign creation or article processing to ensure consistency across the distributed system.
* Transition back to Idle upon successful consensus.

Transitions:

* Campaign Creation to Article Processing:
  + Triggered by a new campaign creation, indicating the system is ready to process articles related to the campaign topic.
* Article Processing to MDim Retrieval:
  + Occurs after processing an article, enabling users to retrieve the updated MDims for a specific campaign and date.
* Article Processing to Consensus Handling:
  + Initiates Paxos consensus to synchronize updates across nodes after processing an article.
* Consensus Handling to Idle:
  + Signifies the completion of the Paxos consensus process, allowing the system to return to an idle state.

API Endpoint Implementation Details

1. POST /createCampaign

Implementation Details:

* Description:
  + This endpoint is responsible for creating a new campaign with a specified topic and default MDims.
* Implementation Steps:

1. Input Validation:

* Verify that the request body contains the required "topic" parameter.
* Validate the format and length of the topic.

1. Campaign Creation:

* Generate a unique identifier (campaignId) for the new campaign.
* Initialize the campaign in the JSON KV Store with default MDims.

1. Response:

* Return the generated "campaignId" in the response.

1. Logging:

* Log the creation of a new campaign, including the campaignId and topic.

2. POST /processArticle

Implementation Details:

* Description:
  + This endpoint processes an article, extracts MDims from its content, and updates the MDims for the specified campaign and date.
* Implementation Steps:

1. Input Validation:

* Ensure the request body contains "campaignId," "articleContent," and "publishedDate" parameters.
* Validate the format of the campaignId and publishedDate.

1. Metadata Extraction (Simplified for PoC):

* Extract MDims from the article content. For the PoC, use a simplified method, like generating random values between 0 and 1.

1. MDim Aggregation:

* Retrieve the current MDims and the count of processed values for the specified date from the JSON KV Store.
* Aggregate the new MDims using the provided formula.
* Update the MDims and increment the processed count.

1. Consensus Handling:

* Initiate the Paxos consensus mechanism to synchronize updates across nodes.

1. Response:

* Return a success flag and the newly calculated MDims in the response.

1. Logging:

* Log the processing of the article, including the campaignId, publishedDate, and the new MDims.

3. GET /readMDims (Concise Description)

Implementation Details:

* Description:
  + This endpoint retrieves the MDims for a given campaign and date.
* Implementation Steps:

1. Input Validation:

* Validate the format of the "campaignId" and "date" query parameters.

1. MDim Retrieval:

* Fetch the MDims for the specified campaign and date from the JSON KV Store.

1. Response:

* Return the retrieved MDims in the response.

JSON Key-Value Store Design and Implementation

Design Considerations:

1. Data Structure:

The JSON KV Store will be organized as a hierarchical structure, where each node corresponds to a campaign, and within each campaign, there are key-value pairs representing MDims for different dates.

json

Copy code

{

"campaigns": {

"campaignId1": {

"date1": {

"mdim1": 0.75,

"mdim2": 0.42,

// ... other MDims

},

"date2": {

// ...

},

// ... other dates

},

"campaignId2": {

// ... similar structure for campaignId2

},

// ... other campaigns

}

}

2. Node State:

Each node in the KV Store represents a campaign, maintaining MDim values for different dates.

3. Consistency and Atomicity:

To ensure consistency, updates to the KV Store are performed atomically. Paxos consensus can be used to synchronize these updates across nodes in the distributed system.

4. Log for Persistence:

Each node maintains an append-only log file to record operations. This log is periodically compressed and snapshot to update the JSON KV Store, providing a form of data persistence.

Implementation Steps:

1. Initialization:

On system startup, read the persisted JSON KV Store from the last snapshot and reconstruct the in-memory representation.

2. MDim Retrieval:

When processing an API request to retrieve MDims (GET /readMDims), fetch the MDims for a given campaign and date from the in-memory representation.

3. MDim Update:

During the processing of an article (POST /processArticle), update the MDims in the in-memory representation according to the aggregation formula.

4. Consensus Handling:

Before updating the in-memory KV Store, initiate the Paxos consensus mechanism to ensure that the update is consistent across all nodes.

5. Logging:

Log each operation (MDim update, campaign creation) in the append-only log file. This log serves as a historical record of operations.

6. Periodic Snapshot:

Periodically, create a snapshot of the in-memory KV Store to persist the current state. Compress the log file to save space and start a new log file.

7. Recovery:

In case of node failure or restart, use the last snapshot and replay the log to reconstruct the most recent state of the KV Store.

Advantages:

Scalability: The hierarchical structure allows for efficient scaling as the number of campaigns and processed articles increases.

Consistency: Paxos consensus ensures that updates to the KV Store are consistent across all nodes, maintaining a unified view.

Persistence: The use of a log file and periodic snapshots provide a level of data persistence, allowing the system to recover its state in case of failures.

Challenges:

Concurrency Control: Handling concurrent updates from multiple nodes requires careful consideration of synchronization mechanisms to maintain consistency.

Snapshot Frequency: The frequency of snapshots must balance the need for persistence with the potential overhead of frequent snapshot creation.

Recovery Complexity: Reconstructing the system state from snapshots and logs adds complexity to the recovery process.

Failures

1. API Request Validation Failure:

* Description: The API request sent to any endpoint does not meet the expected format or includes missing/invalid parameters.
* Handling: Return an appropriate HTTP status code (e.g., 400 Bad Request) with a descriptive error message indicating the validation failure. Include details about which parameter is missing or invalid.

1. Campaign Creation Failure:

* Description: An issue arises during the creation of a new campaign, such as a failure to generate a unique campaignId or an error in initializing the campaign in the JSON KV Store.
* Handling: Return an HTTP status code (e.g., 500 Internal Server Error) along with an error message indicating the specific problem encountered. Log the error details for further investigation.

1. Article Processing Failure:

* Description: Issues may occur during the processing of an article, such as metadata extraction errors, problems with the Paxos consensus mechanism, or errors in updating the JSON KV Store.
* Handling: Return an HTTP status code (e.g., 500 Internal Server Error) with an informative error message. Log details of the error, and consider implementing a retry mechanism for transient failures.

1. MDim Retrieval Failure:

* Description: Failing to retrieve MDims for a campaign and date, possibly due to missing data or an issue with the JSON KV Store.
* Handling: Return an HTTP status code (e.g., 404 Not Found or 500 Internal Server Error) along with an appropriate error message. Log the error details, and if the issue persists, consider implementing a retry mechanism.

1. Consensus Failure:

* Description: The Paxos consensus mechanism fails to reach an agreement among nodes, leading to inconsistencies in the distributed system.
* Handling: In case of Paxos failure, return an HTTP status code (e.g., 500 Internal Server Error) and a descriptive error message. Implement mechanisms to detect and recover from Paxos failures, potentially triggering a system-wide reconciliation.

1. Snapshot and Log Management Failure:

* Description: Errors occur during the periodic snapshot creation, log compression, or any other log management process.
* Handling: Return an HTTP status code (e.g., 500 Internal Server Error) with an informative error message. Ensure detailed logging of these operations for debugging. Implement retry mechanisms and consider alerting administrators for critical failures.

1. Node Failures:

* Description: One or more nodes in the distributed system become unavailable or fail.
* Handling: Implement strategies for node recovery, such as restarting the failed nodes or redistributing the workload to healthy nodes. Use techniques like redundancy and load balancing to enhance system resilience.

1. Security Breach:

* Description: Unauthorized access or malicious activities compromise the security of the system.
* Handling: Implement robust authentication and authorization mechanisms. Regularly audit and monitor system logs for suspicious activities. In the event of a security breach, enforce access controls, and consider notifying relevant stakeholders.

1. Performance Bottlenecks:

* Description: The system encounters bottlenecks, leading to degradation in performance, such as high latency or reduced throughput.
* Handling: Monitor system performance regularly. Identify and address bottlenecks through optimizations or scaling. Implement real-time monitoring and alerting to proactively address performance issues.

1. Data Corruption:

* Description: Corruption occurs in the JSON KV Store, compromising the integrity of campaign data or MDims.
* Handling: Implement data validation checks and ensure atomicity in data updates. In case of corruption, attempt to restore data from the latest snapshot and log files. Establish a data integrity verification mechanism.