Load Testing Documentation for Batch Insert API

Purpose

This document outlines the process for performing load testing on the Batch Insert API to evaluate its performance under varying levels of concurrency. The goal is to identify the API's capacity limits, potential bottlenecks, and error patterns to ensure robustness and reliability in production environments.

Tools and Setup

1. **Script: Use the provided Go script to execute concurrent requests to the Batch Insert API.**
2. **Environment:**
    - **Run the script on a server capable of handling high network throughput and sufficient system resources.**
    - **Ensure the API endpoint is accessible with valid authorization credentials.**
3. **Dependencies:**
    - **Install the Go programming language.**
    - **Ensure network access to the API endpoint.**
4. **Configurations:**
    - **Update constants in the script:**
        - **`url`: Target API endpoint.**
        - **`authKey`: API authorization token.**
        - **`filePath`: Path to the input file for batch insert.**
        - **`jsonPayload`: JSON payload to be sent with each request.**
        - **`initialConcurrency`, `maxRequests`, and `errorThreshold`: Define concurrency levels and error handling thresholds.**

Load Testing Procedure

Step 1: Initial Testing

1. Start with a concurrency level of 100 requests.
2. Run the script and observe:
    - Success and failure count.
    - Response times.
    - Logs for errors or anomalies.

Step 2: Analyze Performance with Different Input Files

1. Test with various input datasets:
    - **File A**: Large dataset with multiple properties (~4 lakh entries).
    - **File B**: Simplified dataset with fewer properties (~4 lakh entries).
2. Document observations:
    - Response times.
    - Errors

Step 3: Combined API Testing

1. Test the Batch Insert API alongside other public APIs to determine if concurrency issues affect other endpoints.
2. Gradually increase the total request count:
    - Start with 25 concurrent requests.
    - Observe behavior beyond this threshold.
    - Document errors like 524 (timeout) or stuck requests.

Key Observations

Batch Insert API Performance

Concurrency level	File type	Duration	Error/Issues
100	4 lakh with many properties	240 sec	Fatal error : concurrent map read/write
	4 lakh with few properties	2 sec	none
500	4 lakh with few properties	55 sec	none
600	4 lakh with few properties	66 sec	none
1000			Requests got stuck beyond 25 concurrent requests with 524 errors.

Combined API Testing

- Successful up to 25 concurrent requests.
- Beyond this, requests to all APIs were stuck, and 524 errors were encountered.

Future Load Testing Process

1. Conduct tests with a wider range of concurrency levels to pinpoint the API's capacity.
2. Use a load testing framework like Apache JMeter or Locust for advanced testing scenarios.
3. Implement monitoring tools to capture real-time metrics such as:
    - Latency.
    - Throughput.
    - Error rates.

Conclusion

This document provides a structured approach to load testing the Batch Insert API. Following these guidelines, you can evaluate API performance under varying conditions and make informed decisions to improve its reliability and scalability.

Go script

```
package main

import (
	"bytes"
	"fmt"
	"io"
	"mime/multipart"
	"net/http"
	"os"
	"sync"
	"time"
)

// Function to create a multipart form for the file and JSON
```

```
func createMultipartRequest(filePath, listID, url, authToken
string) (*http.Request, error) {
	file, err := os.Open(filePath)
	if err != nil {
		return nil, err
	}
	defer file.Close()

	body := &bytes.Buffer{}
	writer := multipart.NewWriter(body)

	// Add file
	part, err := writer.CreateFormFile("file", "4lakh1.csv")
	if err != nil {
		return nil, err
	}
	if _, err := io.Copy(part, file); err != nil {
		return nil, err
	}

	// Add JSON field
	jsonField := fmt.Sprintf(`{"list_id": "%s"}`, listID)
	if err := writer.WriteField("json", jsonField); err != nil {
		return nil, err
	}

	writer.Close()

	// Create the HTTP request
	req, err := http.NewRequest("POST", url, body)
	if err != nil {
		return nil, err
	}
	req.Header.Set("Authorization", authToken)
	req.Header.Set("Content-Type", writer.FormDataContentType())

	return req, nil
}

// Function to send a single request
func sendRequest(wg *sync.WaitGroup, client *http.Client,
filePath, listID, url, authToken string, results chan<- error) {
	defer wg.Done()

	req, err := createMultipartRequest(filePath, listID, url,
authToken)
	if err != nil {
		results <- err
```

```
            return
        }

        resp, err := client.Do(req)
        if err != nil {
            results <- err
            return
        }
        defer resp.Body.Close()

        if resp.StatusCode != http.StatusOK {
            results <- fmt.Errorf("unexpected status code: %d",
resp.StatusCode)
            return
        }

        results <- nil
}

func testConcurrency(concurrent int, filePath, listID, url,
authToken string) (successCount, errorCount int, totalTime
time.Duration) {
        client := &http.Client{}
        var wg sync.WaitGroup
        results := make(chan error, concurrent)

        startTime := time.Now()

        for i := 0; i < concurrent; i++ {
            wg.Add(1)
            go sendRequest(&wg, client, filePath, listID, url,
authToken, results)
        }

        wg.Wait()
        close(results)

        totalTime = time.Since(startTime)

        for err := range results {
            if err != nil {
                errorCount++
            } else {
                successCount++
            }
        }

        return
```

```
}

func main() {
	const (
		filePath  = "path/to/file"
		listID    = "list_id"
		url       = "https://filesync.vinmail.io/v1/file-sync"
		authToken = "Auth Token"
	)

	concurrent := 10 // Starting number of concurrent requests
	maxConcurrent := 1000 // Maximum concurrency to test
	step := 10            // Increment step

	fmt.Println("Starting concurrency test...")

	for concurrent <= maxConcurrent {
		fmt.Printf("\nTesting with %d concurrent requests...\n",
concurrent)
		successCount, errorCount, totalTime :=
testConcurrency(concurrent, filePath, listID, url, authToken)

		fmt.Printf("Concurrency: %d, Success: %d, Errors: %d,
Total Time: %s\n", concurrent, successCount, errorCount,
totalTime)

		if errorCount > 0 {
			fmt.Printf("Reached error threshold at %d
concurrent requests. Stopping test.\n", concurrent)
			break
		}

		concurrent += step
	}

	fmt.Println("Test completed.")
}
```