SPEED TEST REPORT

Introduction

This report presents the results of a performance benchmarking study conducted on six different algorithms designed to handle string search queries on large text files. The project aimed to develop an efficient server script capable of handling concurrent client requests for string searches within a specified file. The primary objective was to identify the fastest algorithm that could be implemented in the production environment.

Algorithm Overview

1. Algorithm 1: This algorithm, implemented in the server.py script, was found to be the fastest among the six algorithms tested. It utilizes a combination of techniques, including regular expression matching, and concurrent threading, to efficiently handle client requests.

2. Algorithms 2-6: These algorithms were not implemented in the production code but were evaluated for their performance characteristics. The source code for these algorithms can be found in the project_resources directory within the Introductory_project folder.

Methodology

The performance benchmarking was conducted by simulating client requests for string searches on text files of varying sizes: 10,000 rows, 250,000 rows, 500,000 rows, and 1 million rows. Each algorithm was tested under two conditions:

1. Reread on Query (True): In this condition, the algorithm reads the entire file content into memory for every client request, simulating a scenario where the file content changes frequently and Search String Not Found

2. Reread on Query (False): In this condition, the algorithm reads the file content into memory only once, caching the data for subsequent client requests. This scenario represents a more optimal use case when the file content remains static and Search String Not Found

Results and Analysis

The results of the performance benchmarking are presented in the following sections, highlighting the execution times for each algorithm under the tested conditions.

Algorithm 1 (Server)			When Reread On Query is TRUE And Search String Not Found				
File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)
10,000 Rows	0.002	250,000 Rows	0.056	500,000 Rows	0.11	1,000,000 Rows	0.24
Algorithm 1 (Server)			**When Reread On Query is Faslse And Search String Not Found**				
File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)
10,000 Rows	0.0000001	250000 Rows	0.00001	500,000 Rows	0.00002	1,000,000 Rows	0.00004

Algorithm 2			When Reread On Query is TRUE And Search String Not Found				

File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)
10,000 Rows	0.009	250,000 Rows	0.087	500,000 Rows	0.174	1,000,000 Rows	0.348

Algorithm 2			When Reread On Query is Faslse And Search String Not Found				
File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)
10,000 Rows	0.001	250000 Rows	0.008	500,000 Rows	0.016	1,000,000 Rows	0.032

Algorithm 3			When Reread On Query is TRUE And Search String Not Found				
File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)
10,000 Rows	0.005	250,000 Rows	0.118	500,000 Rows	0.236	1,000,000 Rows	0.472

Algorithm 3			When Reread On Query is Faslse And Search String Not Found				
File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)
10,000 Rows	0.002	250000 Rows	0.076	500,000 Rows	0.152	1,000,000 Rows	0.304

Algorithm 4			When Reread On Query is TRUE And Search String Not Found				
File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)
10,000 Rows	0.034	250,000 Rows	0.147	500,000 Rows	0.294	1,000,000 Rows	0.588

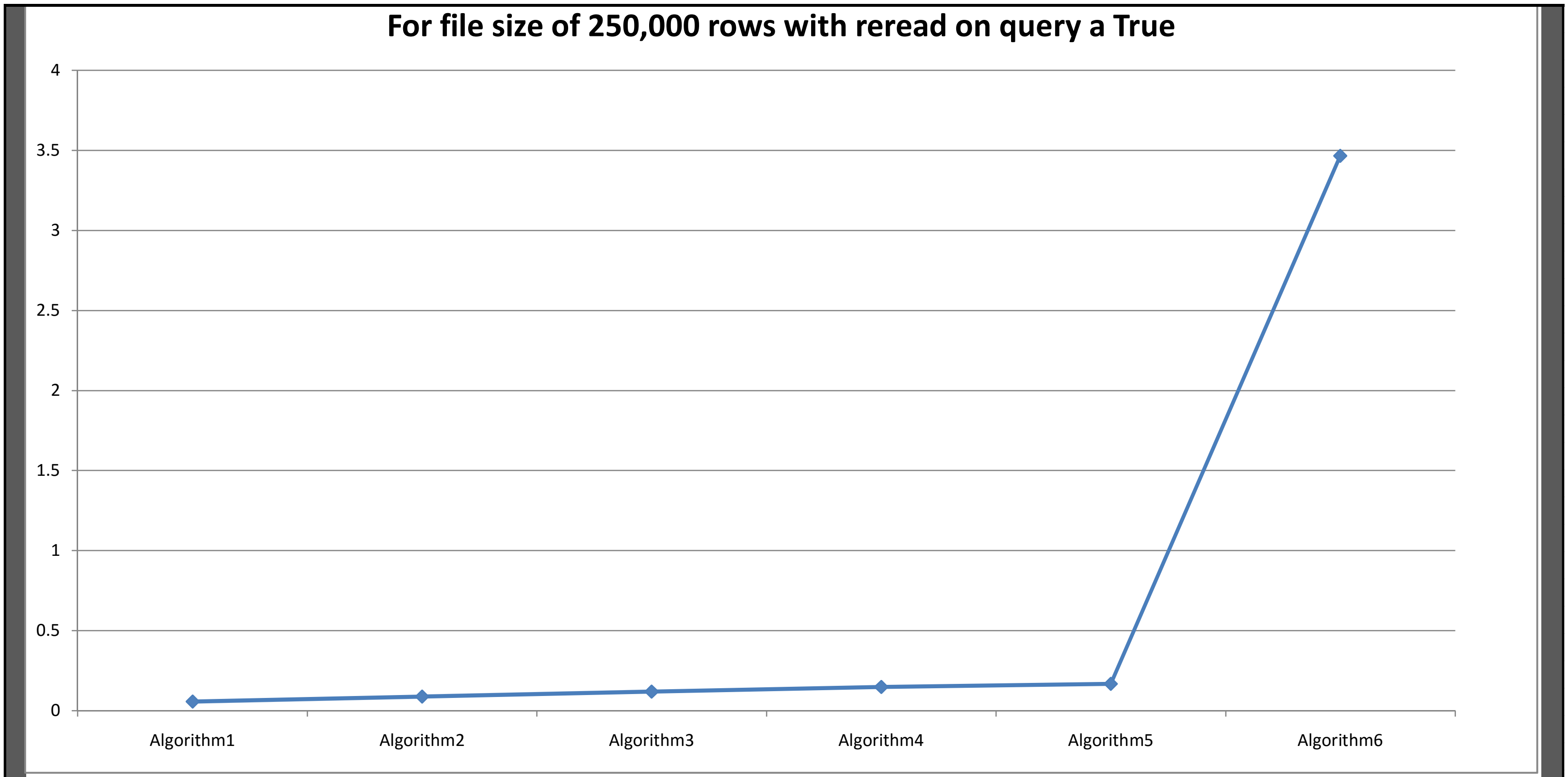
Algorithm 4			When Reread On Query is Faslse And Search String Not Found				
File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)
10,000 Rows	0.009	250000 Rows	0.139	500,000 Rows	0.278	1,000,000 Rows	0.556

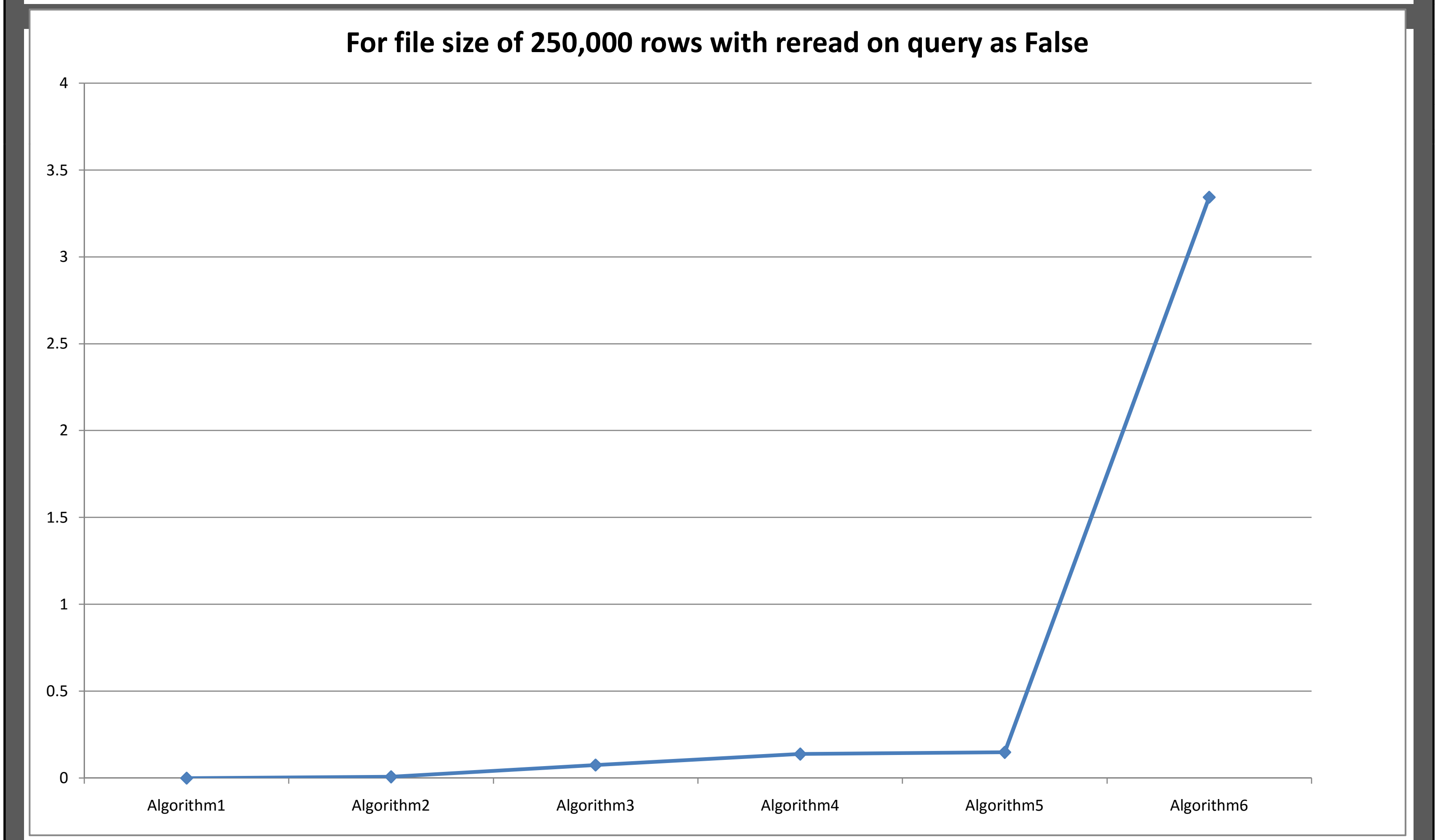
Algorithm 5			When Reread On Query is TRUE And Search String Not Found				
File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)
10,000 Rows	0.034	250,000 Rows	0.167	500,000 Rows	0.334	1,000,000 Rows	0.668

Algorithm 5			When Reread On Query is Faslse And Search String Not Found				
File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)
10,000 Rows	0.007	250000 Rows	0.149	500,000 Rows	0.298	1,000,000 Rows	0.596

Algorithm 6			When Reread On Query is TRUE And Search String Not Found				
File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)
10,000 Rows	3.223	250,000 Rows	3.466	500,000 Rows	3.677	1,000,000 Rows	4.398

Algorithm 6			When Reread On Query is Faslse And Search String Not Found				
File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)	File Size	Speed (s)
10,000 Rows	2.988	250000 Rows	3.344	500,000 Rows	3.634	1,000,000 Rows	4.101


For file size of 250,000 rows with reread on query a True


For file size of 250,000 rows with reread on query as False

Limitations and Considerations

Algorithm 1 demonstrated slightly better performance in the benchmarking tests, it is important to note that the maximum throughput capacity (queries per second) has not been assertained the maximum tested is 8 concurrent client connection which the server was able to handle with ease Additionally, Algorithm 1 is designed to run indefinitely without interruption, which may pose challenges in certain deployment environments or scenarios where periodic restarts or maintenance are required.

Conclusion

Based on the performance benchmarking results, Algorithm 1, implemented in the server.py script, emerged as the fastest and most efficient solution for handling string search queries on large text files. Its combination of file caching, regular expression matching, and concurrent threading techniques proved to be highly effective, especially in scenarios where the file content is not static (reread on query set to True). Depending on the specific requirements and constraints of the project, one of the other algorithms (2-6) may be better suited for certain use cases or scenarios.

Future Work

To further enhance the performance and functionality of the server script, the following areas could be explored:

1. Load Balancing: Implement a load balancing mechanism to distribute client requests across multiple server instances, increasing the overall throughput capacity and ensuring high availability.
2. Parallel Processing: Investigate techniques for parallel processing of client requests, potentially leveraging multiple CPU cores or GPUs for improved performance.

3. Disk-based Indexing: Explore the possibility of using disk-based indexing techniques, such as inverted indexes or suffix arrays, to improve search performance on very large text files that may not fit entirely in memory.

THE END