Optimizing Yioop Search Engine for Analytical Workloads

A Project Report

Presented to

Dr. Chris Pollett
Department of Computer Science
San Jose State University

In Partial Fulfillment
of the Requirements for the Class

Fall 2024: CS297

By

Sujith Kakarlapudi

December,2024

ABSTRACT

Modern search engines and wikis, such as Yioop, generate massive volumes of analytics data, including user interactions [1], system usage trends, and engagement measures, all of which are crucial for improving performance and user experience. Yioop monitors events such as page visits, ad impressions, and search exit clicks in the ITEM_IMPRESSION table, with the computeStatistics function aggregating this data at regular intervals and saving it in the ITEM_IMPRESSION_SUMMARY table while eliminating outdated information. To improve Yioop's analytical performance, this project introduces a roll-up technique to the computeStatistics function for hierarchical data aggregation. Furthermore, synthetic datasets simulating Yioop's analytics are being developed to rigorously test these improvements. These enhancements attempt to grow Yioop's analytics framework for larger datasets and complex reporting demands by enhancing SQLite's processing capabilities and decreasing bottlenecks, and ensuring efficient management of growing data demands.

INTRODUCTION

Yioop, an open-source search engine and wiki, monitors a variety of metrics to better understand user behavior, such as ad impressions, page views, thread interactions, group memberships, and search exit clicks. These metrics are collected and analyzed to provide insight into user behavior and improve search engine performance. However, as data volume grows, Yioop's present SQLite-based [2] design becomes inefficient at meeting these expectations. This project addresses these issues to improve Yioop's capacity to process and store analytics data more effectively.

The major goal is to improve search engine performance by increasing data storage and aggregation in SQLite, particularly within the computeStatistics function. This function collects analytics data from the ITEM_IMPRESSION table at various time intervals (hour, day, month, and year) and saves it to the ITEM_IMPRESSION_SUMMARY table. Previously, data aggregation lacked a roll-up mechanism, resulting in inefficiencies when searching huge data sets.

To address this, a roll-up mechanism has been added to the computeStatistics function to enable efficient data aggregation over several time periods. Synthetic datasets simulating Yioop's analytics environment were created to thoroughly test these enhancements. These enhancements decrease redundancy, improve query performance, and lead to faster lookups and aggregations. The enhancements are intended to help Yioop scale with increasing data needs, improve performance and user experience, and establish the framework for future optimizations discussed in the next sections.

.

DELIVERABLE 1: RESEARCH ON ANALYTICS IN SEARCH ENGINES AND WIKIS

Search engines and wikis, especially Yioop, generate massive volumes of analytics data, which can provide crucial insights into user behavior, system efficiency, and content relevancy. To remain effective, Yioop must include efficient data management mechanisms as its scale grows. Insights from platforms such as Apache Solr, MediaWiki, Google, and Bing can help Yioop improve its analytics handling, assuring scalability and performance.

Platforms like Apache Solr highlight the benefits of lightweight logging for analytics tracking. Rather than depending solely on relational databases, these platforms monitor essential metrics such as query response times [3] and result counts, lowering storage overhead while retaining usable data. Yioop may use a similar recording method to monitor occurrences like search exit clicks and ad impressions. This would improve storage efficiency and reduce database load, especially during high usage. Logs could also help with performance tuning by capturing accurate timestamps and interaction details, which can be used for debugging and optimization.

MediaWiki manages huge traffic levels via incremental aggregation, which is another effective way. Rather than recording each user action, MediaWiki updates aggregated counters for data such as page views, which significantly reduces storage requirements. Yioop can use this method to manage metrics like group membership and thread interactions. Yioop might sustain effective data processing during increases in traffic by directly updating aggregate values, while also facilitating retrieval for analytics queries. This solution is ideally suited to Yioop's need to manage increasing workloads while providing a responsive user experience.

Closed platforms, such as Google and Bing, emphasize the need of session-level logging and data aggregation for optimizing caching and reducing query times. For example, these platforms record session parameters like dwell time and CTR [4] to improve relevance and performance. Yioop, being an open-source system, can take similar techniques by analyzing session data to find popular content and optimize cache strategies. These insights will enable Yioop to provide faster and more relevant search results while enhancing overall system performance.

Implementing these strategies would significantly improve Yioop's ability to process and analyze its expanding analytics dataset. Yioop's lightweight logging, incremental aggregation, and session-based optimization can help reduce performance bottlenecks and boost scalability. These enhancements will enable Yioop to manage larger datasets more efficiently, assuring a seamless and responsive experience for users while laying the groundwork for future advances in its analytical capabilities.

DELIVERABLE 2: IMPLEMENTATION OF ROLLUP

The enhanced computeStatistics function is a significant advancement toward optimizing Yioop for handling analytical workloads effectively. This function now includes a simulated rollup mechanism, allowing for hierarchical data aggregation over multiple time periods such as hourly, daily, monthly, and yearly. Yioop addresses SQLite's problem of not having native rollup functionality, enhancing scalability and enabling sophisticated data management.

To replicate the rollup capability found in other database systems such as MySQL, we used a sequence of SQL procedures to aggregate data at different levels. The strategy involved arranging the process as a union of queries, with each query individually aggregating data for a given time period. For example, one query concentrates on hourly data, another on daily data, and so on, with the results integrated into a single format.

The simulated rollup procedure uses UNION ALL to merge these granularities into a single result set. By grouping data by determined time-based attributes (such as year, month, day, and hour), the function guarantees that the aggregated data appropriately represents the various time periods. Yioop's hierarchical structure enables it to efficiently handle complicated analytical queries while remaining flexible enough to scale with rising data quantities.

Implementation Details:

To accommodate the rollup technique, many changes were made to the computeStatistics function and database schema.

1. Dynamic Time-Based Aggregation: The function calculates timestamps at each level of granularity (hourly, daily, monthly, and yearly). These timestamps ensure that data is properly sorted and aggregated for each period.
2. Enhanced Schema for Intuitive Analytics: The ITEM_IMPRESSION_SUMMARY table

now has four additional columns: UPDATE_YEAR, UPDATE_MONTH, UPDATE_DAY, and UPDATE_HOUR. These columns allow for more organized and efficient querying of aggregated data.

3. SQL Union for Simulated Rollup: The function uses UNION ALL to combine many SQL queries, each responsible for a specific level of granularity. This technique provides the hierarchical aggregation needed for analytical insights while overcoming SQLite's intrinsic restrictions

This approach resembles the behavior of a native rollup capability, ensuring that Yioop can process and summarize analytics data over a variety of time periods without losing efficiency.The rollup approach significantly improves Yioop's analytical architecture, making data aggregation more efficient for scaling applications. The computeStatistics function prepares the environment for testing and validation with synthetic datasets that mimic real-world Yioop analytics. These datasets will evaluate the rollup mechanism's effectiveness under a variety of scenarios and provide insights for future development.

DELIVERABLE 3: PREPARING SYNTHETIC DATASET

The objective of this deliverable was to create a synthetic dataset that simulated Yioop's analytics (for example, wiki impressions) in order to test the computeStatistics function's performance and correctness. The dataset simulates real-world user interactions with wiki pages and other content, collecting impressions at random timestamps to assess the function's ability to produce statistics in a variety of scenarios.

Dataset creation and key considerations:

A total of 500 users were created, each with their own group. Each user has a unique profile and is allocated to at least one group. This resembles the user-group relationships found in a social or collaborative platform.

Users were divided into 100 groups and allocated at random. Each user is a member of at least one group, and many users belong to several groups, reflecting typical user activity on platforms such as Yioop.

Wiki Page Creation:

Wiki pages were assigned to each group, representing the content with which members interact. These pages are essential for imitating real-world user behavior since users commonly engage with material within their groups.

Impression Data Generation:

The synthetic dataset is built using impression recordings. These entries describe exchanges in which a user examines a wiki page within their group. The ITEM_IMPRESSION table stores each impression, which includes the USER_ID, PAGE_ID, and interaction date.

Random Timestamps: To imitate realistic user activity, impressions were created with random timestamps from the previous year. This variety in timestamps guarantees that the dataset covers

various periods of user activity, allowing the computeStatistics function to be tested under more dynamic and unpredictable settings. Impressions are dispersed across the year, reflecting real-world user behavior that varies over time.

```
$current_timestamp = time ();
    $one_year_ago_timestamp = $current_timestamp - (365 * 24 * 60 * 60);
    foreach ($unique_user_id_list as $user_id) {
        foreach ($page_id_list as $page_id) {
            $random_timestamp = rand ($one_year_ago_timestamp, $current_timestamp - 60);
            $this->addWithDb($user_id, $page_id, C\WIKI_IMPRESSION, $db,
$random_timestamp);
            L\crawlLog("Adding to DB with USER_ID: $user_id, PAGE_ID: $page_id,
TIMESTAMP: $random_timestamp");
        }
    }
```

Dataset Summary:

Users: 500 people with individual IDs and personal groups.

Groups: 100 groups, each having several users to ensure accurate membership distribution.

Impressions: Approximately 10,000 impressions, randomly scattered across the past year.

Purpose: To offer a realistic dataset for testing the computeStatistics function, verifying that it can handle diverse, time-varying user interactions and perform analytics quickly in real-world scenarios.

DELIVERABLE 4: A/B TESTING

In this deliverable, an A/B test was conducted to compare the execution times of the computeStatistics function using two approaches.

Original Approach: The original computes statistics function directly performs aggregation without any rollup on the data.

Approach with Rollup Simulation: This method replicates a rollup process in which data is aggregated in intermediate steps prior to final computation.

Execution Time Results:

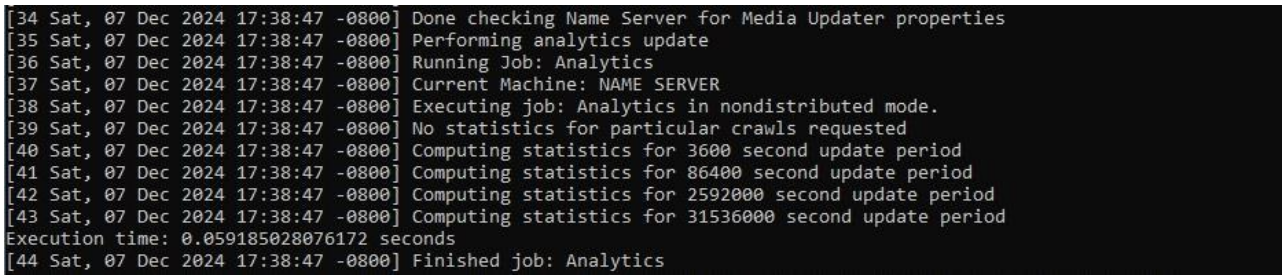
The test's purpose was to determine how the rollup technique affected the computeStatistics function's execution time. The results, measured in seconds, are summarized as follows:

Original Approach (Without ROLLUP):

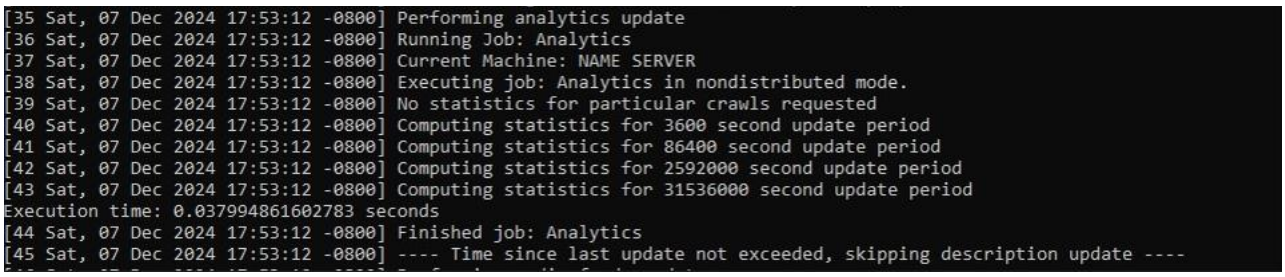
Execution times: 0.03, 0.04, and 0.05 seconds (approx.)

The original method produces relatively consistent execution times, with little change between runs.

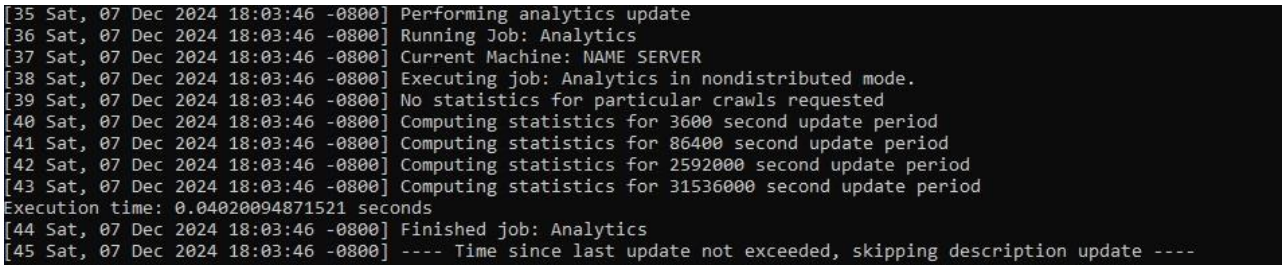
Test Result 1:



Test Result 2:



Test Result 3:

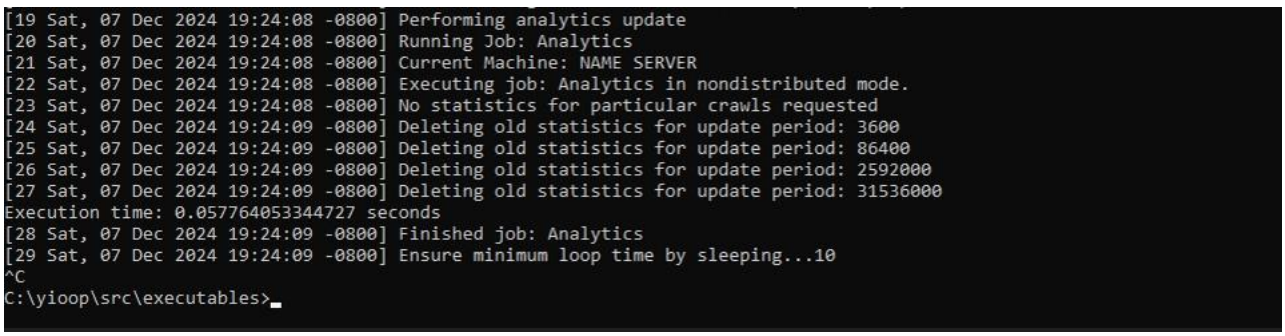


Updated ComputeStatistics (With ROLLUP Simulation):

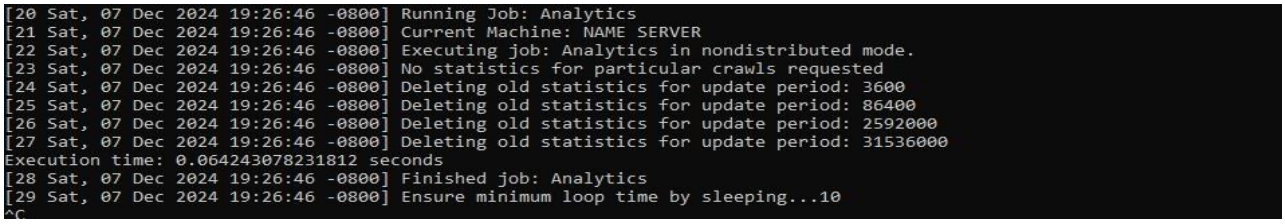
Execution times: 0.05, 0.06, and 0.07 seconds (approx.)

The approach integrating rollup techniques has slightly longer execution times than the original method.

Test Result 1:



Test Result 2:



Test Result 3:

```
[20 Sat, 07 Dec 2024 19:30:54 -0800] Running Job: Analytics
[21 Sat, 07 Dec 2024 19:30:54 -0800] Current Machine: NAME SERVER
[22 Sat, 07 Dec 2024 19:30:54 -0800] Executing job: Analytics in nondistributed mode.
[23 Sat, 07 Dec 2024 19:30:54 -0800] No statistics for particular crawls requested
[24 Sat, 07 Dec 2024 19:30:54 -0800] Deleting old statistics for update period: 3600
[25 Sat, 07 Dec 2024 19:30:54 -0800] Deleting old statistics for update period: 86400
[26 Sat, 07 Dec 2024 19:30:54 -0800] Deleting old statistics for update period: 2592000
[27 Sat, 07 Dec 2024 19:30:54 -0800] Deleting old statistics for update period: 31536000
Execution time: 0.078784942626953 seconds
[28 Sat, 07 Dec 2024 19:30:54 -0800] Finished job: Analytics
```

In conclusion, while the rollup simulation resulted in somewhat longer execution times, the difference was modest, showing that the rollup strategy adds some overhead. However, the rollup technique may offer further advantages in other areas, such as increased data accuracy or enhancements for future computations.

FUTURE WORK AND CONCLUSION

This project focused on on improving Yioop's analytics data summarization using the rollup process, which resulted in better data aggregation. However, it resulted in a minor increase in execution time over the previous approach. Future work in CS298 will investigate whether a database or log-based system is more efficient for storing and evaluating analytics data, considering retrieval and processing requirements. This will let Yioop determine the optimum strategy for long-term storage and high-performance data retrieval as its dataset expands. In addition, efforts will be made to optimize storage and query execution in order to assure scalability while maintaining efficiency.

In conclusion, while the rollup strategy improved data aggregation, it resulted in longer execution times. These discoveries lay the groundwork for additional optimization, such as investigating novel storage strategies and improving Yioop's ability to handle larger analytical workloads effectively.

REFERENCES

[1] Stonebraker, M. and Çetintemel, U. 2005. "One Size Fits All": An Idea Whose Time Has Come and Gone. *Proc. of the Int. Conf. on Data Engineering* (2005), 2-11.

[2] Heitzmann, A., van Beek, M., and Zhao, Z. 2019. SQLite: Past, Present, and Future. *Int. J. of Computer Science and Software Engineering* 8, 5 (2019), 197-209.

[3] Graefe, G. 1993. Query evaluation techniques for large databases. *ACM Computing Surveys* 25, 2 (1993), 73-170.

[4] Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M., and Vassilakis, T. 2010. Dremel: Interactive analysis of web-scale datasets. *Proc. of the VLDB Endowment* 3, 1-2 (2010), 330-339.

.