Optimizing Edge-Based Query Processing for Real-Time Applications

Kalgi Gandhi and Minal Bhise
Distributed Database Group, DA-IICT, Gandhinagar, India
{*kalgi_gandhi, minal_bhise*}*@daiict.ac.in*

Keywords: Column Imprint, Edge Intelligence, Edge Query Processing, Resource Utilization, Workload-Aware.

Abstract: The rapid growth of edge devices in large-scale systems presents challenges due to limited processing power, memory, and bandwidth. Efficient resource utilization and data management during query processing are critical, especially for costly join operations. The Column Imprint-Hash Join (CI-HJ) accelerates hash joins using equi-height binning but lacks real-time efficiency and scans unnecessary cachelines. This paper introduces Workload Aware Column Imprint-Hash Join (WACI-HJ), a novel approach that leverages workload prediction to optimize hash joins for real-time edge query processing. WACI-HJ comprises of two phases: the WACI-HJ Generation Phase predicts query workloads and pre-processes data into bins using blocking and hashing techniques, reducing overhead before query arrival. The Query Processing and Resource Utilization Phase efficiently utilizes CPU, RAM, and I/O resources for runtime processing. Evaluations using Benchmark and Real-World datasets demonstrate significant improvements in the Percentage of Cachelines Read PCR, Query Execution Time QET, and Resource Utilization. PCR and QET show 18% and 5% improvement respectively. The proposed technique has been demonstrated to work well for scaled and skewed data. Although PCR is an indirect measure of energy consumption, direct Energy-Efficiency Experiments reveal gains of 1%, 23%, and 18% in CPU, RAM, and I/O utilization respectively. WACI-HJ provides an optimal and sustainable solution for edge database management.

1 INTRODUCTION

Edge Computing (EC) performs computing at the edge, i.e. the source of the network. In contrast to Cloud Computing, EC brings computation and data storage closer to the data sources (Awaysheh et al., 2023). The typical edge system is only a few hops away from the data layer. It enables real-time applications, minimizes data transfer, and facilitates rapid decision-making. Ideal for resource-constrained environments and the Internet of Things (IoT), EC analyzes data locally before transmitting structured data to the cloud.

In the context of the Internet of Things (IoT), if all the large amounts of unstructured or semi-structured data generated by the connected devices are transmitted to the cloud, latency and bandwidth will increase. EC is an intermediate step, as seen in Figure 1. It processes the unstructured or semi-structured data transferred by IoT and sends the processed structured data to the cloud for storage. This data-handling process is crucial for efficient query processing, as it ensures that only the processed data is sent to the cloud, reducing latency and bandwidth usage.

EC involves storing, processing, and analyzing data directly on edge devices or nodes rather than sending all data to the cloud. This approach reduces latency by enabling quick query responses and optimizing bandwidth by transmitting only relevant information. It handles and analyzes data closer to its source rather than sending it to distant servers. This minimizes latency, conserves bandwidth, enables real-time analytics, and reduces energy consumption (Bilal et al., 2018).

1.1 Motivation

EC performs computing close to the source of the network, and it is just a few hops away from the data layer. The number of edge-based applications and the associated data are growing rapidly. EC applications in various fields, including Smart Manufacturing, Intelligent Transportation Systems, Health Monitoring Systems, and Smart Homes. These devices are actively deployed and utilized in these domains to enhance efficiency, safety, and convenience.

As smart applications continue to develop, the number and variety of connected devices have surged, resulting in rapid data growth. From 9.76 billion IoT

Gandhi, K. and Bhise, M.
Optimizing Edge-Based Query Processing for Real-Time Applications.
DOI: 10.5220/0013289300003929

In *Proceedings of the 27th International Conference on Enterprise Information Systems (ICEIS 2025) - Volume 1*, pages 259-266
ISBN: 978-989-758-749-8; ISSN: 2184-4992

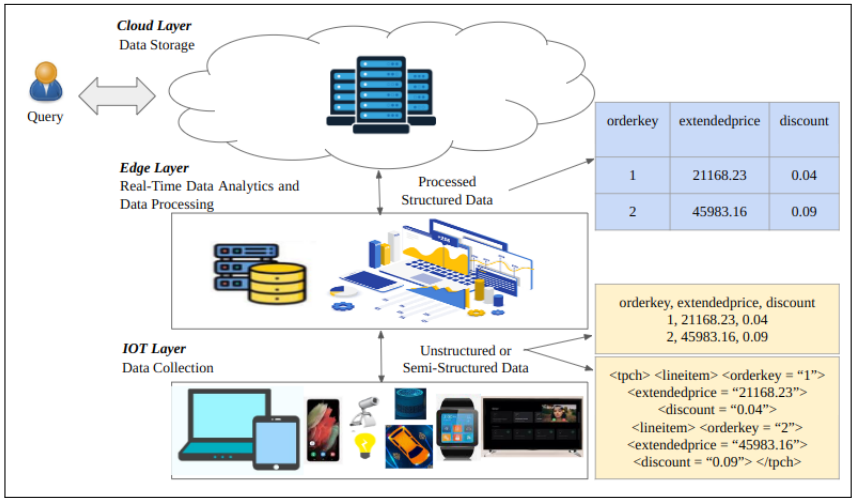

Figure 1: Query Processing at Edge.

devices in 2020, the count is projected to reach 34.6 billion by 2030 (iot, b), reflecting widespread adoption across various industries. Correspondingly IoT data, which was 44 zettabytes (ZB) in 2020, is expected to soar to 246 ZB by 2030 (iot, a). This explosive increase underscores the crucial role of effective data transmission and storage at the edge to effectively manage and utilize this expanding digital ecosystem.

1.2 Contributions

The main contributions of the work are as follows:

- To accelerate Query Processing in Edge Workload-Aware Column Imprint-Hash Join (WACI-HJ) is proposed.
- To enhance the real-time query processing performance, Workload Prediction is incorporated with WACI-HJ.
- WACI-HJ is implemented for benchmark Transaction Processing Performance Council (TPC) TPC-H (tpc, b) and TPC-D (tpc, a) datasets alongwith, a real-world dataset Metropolitan Atlanta Rapid Transit Authority (MARTA) (mar,) aims to validate the robustness and versatility of the algorithm across diverse domains.
- WACI-HJ performance is compared with the State-of-the-Art, Column Imprint-Hash Join (CI-HJ) (Li and Xu, 2021) for the identified evaluation parameters.
- WACI-HJ contributes to reducing latency, improving energy-efficiency, and optimizing data access, which can support sustainable database management for diverse EC applications.

The rest of the paper is organized as follows: Section 2 briefly summarizes the Literature Survey. The Proposed Technique is elaborated in section 3, its Implementation Details and Results are discussed in sections 4 and 5, and section 6 concludes the paper.

2 LITERATURE SURVEY

This section discusses the edge data management and query processing domains. Additionally, merging Artificial Intelligence (AI) with EC predicts queries and keeps track of the resources used in EC setups.

2.1 Data Management and Query Processing at Edge

EC transforms data processing by decentralizing computation and storage, reducing latency and enhancing responsiveness. Techniques such as Load Balancing (Li et al., 2020) ensure balanced task distribution across edge devices, optimizing performance and preventing overload. Data Compression (Gandhi et al., 2021) minimizes data size before transmission, maximizing bandwidth efficiency and accelerating data transfer. Predicate Caching (Schmidt et al., 2024) stores frequently accessed data locally, improving query response times, while Predictive Analytics (Ma et al., 2018) enables proactive decision-making based on historical data insights. Efficient resource management strategies like Data Partitioning (Gandhi and Bhise, 2019) and Prefetching (Chen et al., 2007) enhance processing efficiency and reduce access latency, while techniques such as Parallel Joins (Vitorovic et al., 2016) enhance scalability. Initiatives in

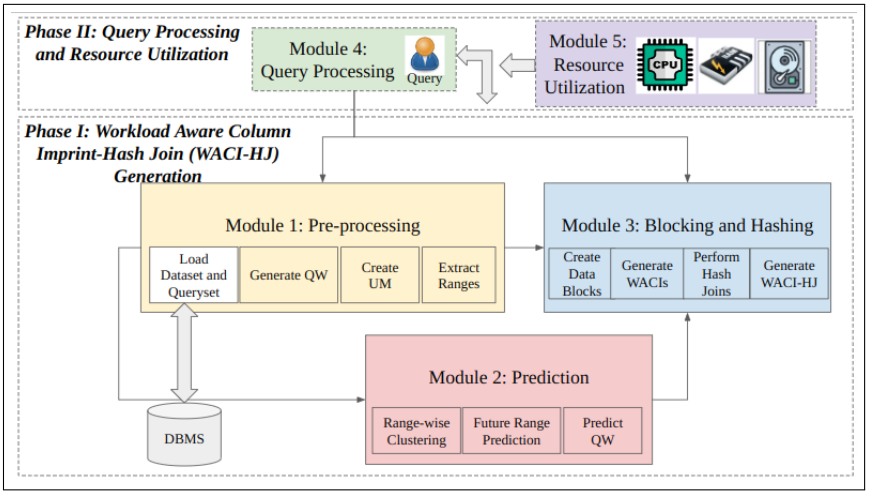

Figure 2: WACI-HJ System Architecture.

energy-efficient EC, such as Green Cloud Networking (GCN) (Sun and Ansari, 2017), optimize energy consumption. Data encoding enhances memory usage and reduce transmission volume (Symeonides et al., 2019) while leveraging parallelism to split queries into independent sub-queries helps optimize throughput and latency (Xu et al., 2018). The CI-HJ strategy (Li and Xu, 2021) further improves performance in large-scale edge systems by pre-computing CIs for join operations, accelerating hash processing and reducing the data volume that must be scanned and compared.

2.2 Edge Intelligence

Integrating AI with EC enhances processing and decision-making at the network edge. Techniques like Regression (Ma et al., 2018), Neural Networks (Ma et al., 2018), and ARIMA (Calheiros et al., 2014) enable real-time data analysis and predictions, reducing latency and bandwidth usage. However, current Edge Intelligence (EI) models face challenges in predicting complex queries, particularly in resource-constrained edge environments.

3 WORKLOAD AWARE COLUMN IMPRINTS-HASH JOIN (WACI-HJ)

Cloud-based database systems often face scalability and sustainability challenges due to bandwidth limitations and high latency (Meruje Ferreira et al., 2024). Edge computing mitigates these issues by bringing storage and computation closer to the user, enabling efficient real-time query processing. However, resource constraints in edge systems necessitate optimized query execution, especially for join operations. Hash joins are a key strategy for improving performance and scalability in such environments (Barber et al., 2014). CI-HJ (Li and Xu, 2021), a state-of-the-art approach for large-scale edge systems, leverages pre-computed CIs to accelerate hash joins by reducing the data volume involved, thus enhancing query performance and minimizing resource utilization.

WACI-HJ enhances hash joins in large-scale edge systems by leveraging workload-aware binning and predictive optimization. Unlike traditional equi-height binning, it uses workload patterns and value weights, such as cachelines or value frequencies, to construct histograms aligned with query requirements. The approach pre-computes CIs for each table and generates optim ized imprint vectors for probe columns, reducing cache misses during hash joins. Additionally, WACI-HJ integrates a forecaster to analyze historical data trends, predict upcoming workloads, and allocate resources efficiently, enabling real-time query processing and improved performance in edge systems.

Phase I includes Pre-processing, Prediction, and Blocking and Hashing modules, which compute bins ahead of query arrival. Module 1 loads the dataset, generates queries using Zipf's Law, and constructs a Usage Matrix to identify attribute usage patterns. Module 2 computes cluster usage frequency and applies the ARIMA technique to predict future usage. Module 3 assigns bins, sets borders, and divides data into blocks, storing them as WACIs.

In Phase II, Module 4 updates the workload and WACI-HJ Output, loading necessary cachelines into main memory for query processing. Module 5 moni-

tors CPU, RAM, and I/O utilization to ensure efficient resource management during query execution.

3.1 WACI-HJ Data Structures and Algorithm

As depicted in Algorithm 1, WACI-HJ for Module 1 involves the Usage Matrix Basket (UMB), which lists queries and their associated attributes. For Module 2, the Query Workload List (QWL) holds the total query frequency count. The Cluster-Range Table (CRT) comprises of ranges. The Cluster-Frequency Table (CFT) contains the query range and weekly arrival frequency. The workload and WACI-HJ Output are updated upon the query arrival. WACI-HJs, stored in secondary memory, bring the necessary cachelines to the main memory for query output. Further, CPU, RAM, and I/O utilization are monitored during query execution.

The time and space complexities of WACI-HJ are $O(2^x(n*t)*64)$ and O(n) respectively, where x is the log of the number of bins, n represents the total number of tuples in the probe column, i.e. no_of_cachelines*64, and t is the size of the data type.

4 IMPLEMENTATION DETAILS

This section provides a comprehensive overview of the hardware and software setup, dataset and query-set specifications, and evaluation parameters used for the implementation. It outlines the technical environment and resources utilized to demonstrate the effectiveness of the WACI-HJ algorithm.

4.1 Hardware and Software Setup

The machine hardware configuration included a quad-core Intel i3-2100 CPU clocked at 3.10 GHz. The system has 32 GB of RAM, temporarily storing data that the CPU needs to access quickly. It has 500 GB of hard disk space for storing data and files. Additionally, the system includes hardware with L1, L2, and L3 caches sized at 64 Kilobyte (KB), 512 KB, and 3 MB respectively, enhancing processing speed by storing frequently accessed data close to the CPU.

The software required to implement WACI-HJ is the VS Code IDE platform (vsc,) for C language development, and MonetDB (mon,), an open-source column store database for implementing CIs. Google Colab (goo,) for Python, supporting machine learning computing are used for prediction tasks.

Algorithm 1: Workload Aware Column Imprint-Hash Join (WACI-HJ) Generation Algorithm.

```
Input: Tuples T_i, Queries Q_j
Output: WACI-HJ Output
//Module 1: Pre-processing
Initialize UMB
for query Q_j do
   A_j = Extract_Attributes_From_Query(Q_j)
   for attribute A_i in A_j do
      if Ai exists in UMB then
         Add_Query_Index(A_i, Index_of(Q_j), UMB)
      else
         Set_Query_Index(A_i, [Index_of(Q_j)], UMB)
      end if
   end for
end for
//Module 2: Prediction
Create CRT C_i and CR_i State CFT = number of weeks * CRT
Cluster_week_freq = sum_{j=1}^{no_of_queries} QF_j
CF_i = array(cluster_week_freq)
model = auto_arima(CF_i, suppress_warnings = True)
PredictedQWL = model × predict(n_periods = 1, return_conf_int = True)
//Module 3: Blocking and Hashing
imp_vec = no_of_cachelines × sizeof(ImprintVector)
for i = 0 to no_of_cachelines − 1 do
   imp_vec[i] = no_of_bins × sizeof(datatype)
end for
for j = 0 to no_of_cachelines − 1 do
   for val = 1 to cachelineSize do
      bin = getBin(val)
      imp_vec[j][bin] = 1
   end for
end for
for val = 0 to build_col_size − 1 do
   HashArray[hashFunction(col[val])] = val[i]
end for
for i = minimum_bin_required to maximum_bin_required do
   while list_temp[i] do
      q = list_temp[i] → data
      for w = q × cacheline_size to min(q × (cachelines_size + 1), size_probe_col) do
         if probe_col[w] ≤ bin_borders[i] or probe_col[w] ≥ bin_borders[i+1] then
            continue
         end if
         if HashArray[hashFunction (probe_col[w])] ≠ −1 then
            store BAT Values corresponding to both columns in WACI-HJ Output
         end if
      end for
      list_temp[i] = list_temp[i] → next
   end while
end for
```

4.2 Dataset and Queryset

The implementation of WACI-HJ is demonstrated using a benchmark dataset, a standardized dataset that is generally well-documented, public, and widely used in research. Alongwith, real-world datasets contain authentic data from sources like sensors.
TPC-H (tpc, b)- TPC-H a benchmark dataset in the supply chain domain, simulates a data warehousing environment with eight tables. The orderkey attribute is used for experimentation, joining l_orderkey with o_orderkey. The 1 GB dataset is scaled upto 10x for testing. Out of 22 OLAP queries, 9 involve joins with orderkey, and 11 custom range queries with joins were added, totalling 20 queries. TPC-D, a skewed version of TPC-H, is also referenced.
MARTA (mar,)- MARTA, a real-world dataset from the transportation domain, includes 13 tables. The route_id attribute is used for experimentation. The original 93.1 MB dataset is scaled to 10x, and 10 custom range queries with joins were created for evaluation.

4.3 Set of Experiments

The experiments are organized into distinct sets designed to thoroughly evaluate the algorithm performance, robustness, and adaptability across various scenarios.
Set 0: System Configuration- Initially, Set 0 focuses on configuring the system by determining the optimal bit size and number of clusters to enhance the algorithm effectiveness. Once the system is configured, the subsequent experiments are divided into four sets.
Set I: Basic Experiments- Set I involves basic tests conducted on the standard dataset size, with TPC-H and APM datasets initially set at 1 GB.
Set II: Data Scaling Experiments- In Set II, the experiments assess how the algorithm performs with varying data sizes, scaling the datasets from x upto 10x.
Set III: Varying Skewness Levels Experiments - Real-world data is often not evenly distributed. Hence, Set III examines the algorithms robustness by evaluating its performance on datasets with skewness levels ranging from 50% i.e. uniformly skewed to 99%, addressing real-world data distribution challenges.
Set IV: Energy-Efficiency Experiments- Set IV directly measures energy-efficiency by monitoring CPU and RAM utilization and I/O efficiency (Patel and Bhise, 2023). It aims to evaluate the systems energy consumption during query execution across different data types and workloads.

4.4 Evaluation Parameters

The evaluation of WACI-HJ contains Input, Output, and Resource Utilization parameters.
Input Parameters- Input Parameters include Bit Size (ranging from 8 to 128 bits), Data Size (scaled from x to 10x of a base 1 GB for TPC-H and 0.97 GB for APM), and Skewness Levels (ranging from 50% to 99%).
Output Parameters- Output Parameters measure system performance, PCR which indicates the proportion of blocks accessed by a query compared to the total number of blocks, QET measures the time taken for query execution in seconds, and AET denotes the total duration of algorithm execution, also in seconds.
Resource Utilization Parameters- Resource Utilization Parameters assess the use of computational resources, including CPU Utilization, RAM Utilization, and I/O Efficiency, to measure the system efficiency and performance under various conditions.

5 RESULTS AND DISCUSSIONS

This section evaluates indexing techniques, including the state-of-the-art CI-HJ (Li and Xu, 2021) and the proposed WACI-HJ with (Shah et al., 2024) and without workload prediction, based on input, output, and resource utilization parameters. The results are derived using benchmark datasets TPC-H (tpc, b), TPC-D (tpc, a), a skewed TPC-H and the MARTA dataset (mar,), both representing high growth edge data applications. Results are averaged across all queries in the queryset.

These evaluations involved four experiments, optimizing parameters like bit size and cluster count for efficient PCR and QET. Bit size, being system dependent, remains constant across datasets, while cluster count varies with data size larger datasets generate more clusters. Further sets use a 32-bit size, with 4 clusters for TPC-H (tpc, b) and TPC-D (tpc, a), and 2 clusters for MARTA (mar,).

5.1 Set I: Basic Experiments

In the Basic Experiments, PCR is measured based on query selectivity and frequency. Results shown in Figure 3, reveal a significant improvement in queries with lower selectivity due to more non-distinct values in the orderkey attribute, especially when compared to CI-HJ. WACI-HJ reduces scanned cachelines by 14% compared to CI-HJ, with an additional 4% improvement when workload prediction is included. This reduction is attributed to workload-aware optimization,

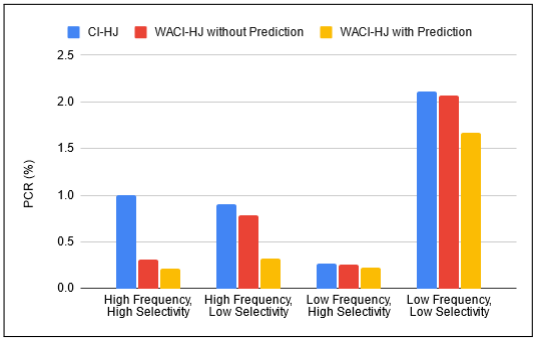

Figure 3: TPC-H Set I: PCR.

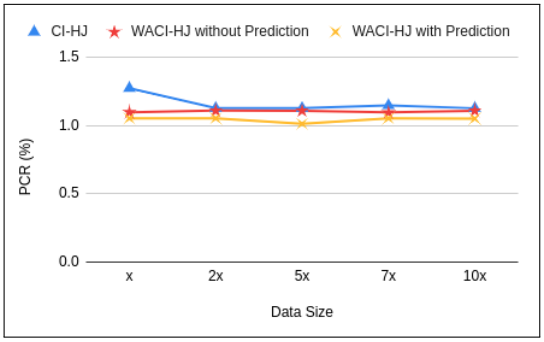

Figure 5: TPC-H Set II: PCR.

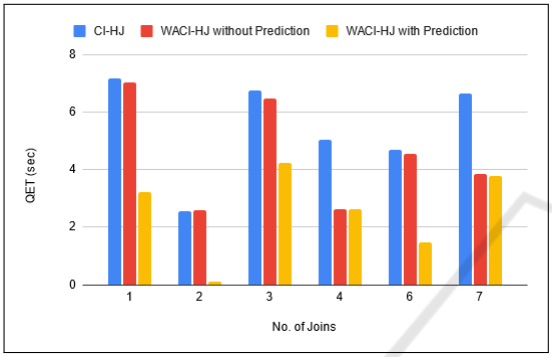

Figure 4: TPC-H Set I: QET.

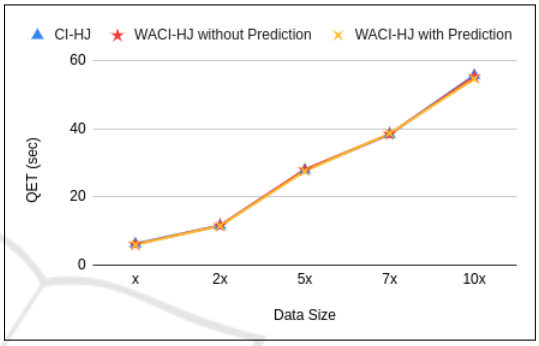

Figure 6: TPC-H Set II: QET.

leading to lower RAM and I/O usage and decreased system latency.

Figure 4 shows the QET results for the Basic Experiments, with queries categorized by the number of joins. WACI-HJ delivers better performance for queries with more joins, demonstrating its efficiency in hash join acceleration. Compared to CI-HJ, WACI-HJ achieves a 3% improvement in QET, with an additional 2% enhancement when workload prediction is integrated. This results in faster query execution and reduced CPU usage.

5.2 Set II: Data Scaling Experiments

Data Scaling Experiments execute the proposed technique with data sizes ranging from x, 2x, 5x, 7x, and 10x.

Figure 5 shows the results of the PCR for Data Scaling Experiments. The overall gain was 7% to 18% when scaled from x to 10x. The trail indicates that the PCR for WACI-HJ is constant wrt. data size.

Figure 6 shows the results of the scaled QET, with WACI-HJ achieving an overall improvement of 2% to 5% in Data Scaling Experiments, demonstrating consistent QET enhancement for large-scale query scenarios.

WACI-HJ demonstrates a 2% higher AET compared to CI-HJ, but achieves a significant 5% improvement in QET. When data scales from x to 10x, AET improves by up to 19% due to optimized workload bins, making WACI-HJ more suitable for larger datasets.

5.3 Set III: Varying Skewness Levels Experiments

Varying Skewness Levels Experiments are performed using the TPC-D (tpc, a) dataset. WACI-HJ is evaluated at different levels of data skewness.

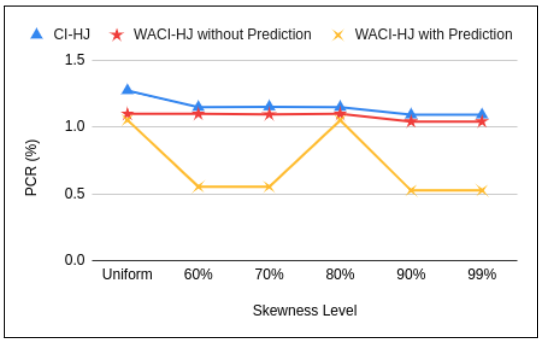

Figure 7: TPC-D Set III: PCR.

WACI-HJ PCR is constant for various skewness levels, as shown in Figure 7. An 5% to 50% improve-

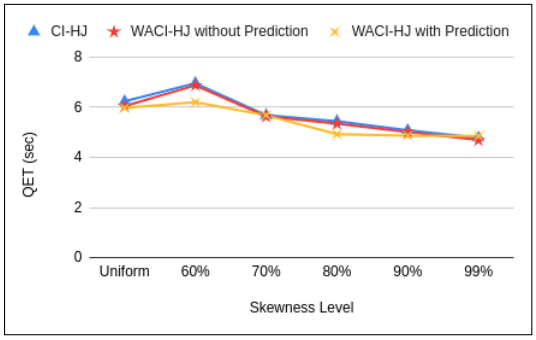

Figure 8: TPC-D Set III: QET.

ment is recorded for WACI-HJ over CI-HJ as the data gets skewed. A peak is observed for 80% skewed data, and we are further investigating it.

As shown in Figure 8, WACI-HJ becomes 2% to 10% faster with a further increase in skewness level. It shows that WACI-HJ is robust for different skewness levels.

5.4 Set IV: Energy-Efficiency Experiments

Although PCR indirectly reflects energy consumption, the energy-efficiency of WACI-HJ is directly measured using Resource Utilization tools like htop and iotop to track CPU, RAM, and I/O consumption. This section compares the resource utilization of WACI-HJ with the state-of-the-art technique, using the TPC-H dataset for evaluation.

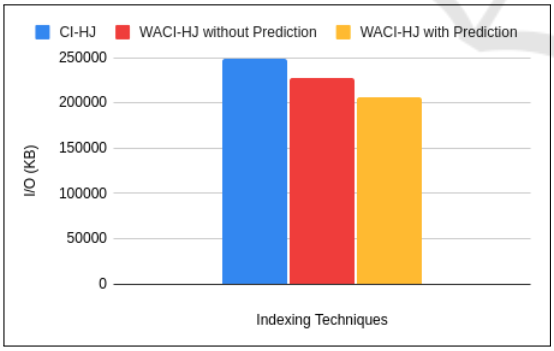

Figure 9: TPC-H Set IV: IO Efficiency.

The CPU utilization of WACI-HJ shows 2% gain is observed in CPU usage compared to CI-HJ, reducing the latency. A 27% gain is observed in RAM usage compared to CI-HJ due to the optimal number of cachelines read. Due to the effective number of cachelines read, I/O operations were reduced by 18%, making the system more faster and energy-efficient, as seen in Figure 9.

5.5 Comparison with State-of-the-Art

CI-HJ (Li and Xu, 2021) improves query performance by using CIs to avoid scanning irrelevant data blocks, but it may incur significant processing overhead with large datasets due to unnecessary data scans and lacks real-time workload adaptation.

In contrast, WACI-HJ adopts a workload-aware approach, creating bins based on workload information to minimize cache misses and predicting future workloads for effective real-time query processing.

5.5.1 Quantitative Comparison

In experiments on the TPC-H (tpc, b) dataset, WACI-HJ outperforms CI-HJ, with an 18% improvement in PCR and a 5 second reduction in QET, demonstrating faster query execution. Despite a 2% increase in AET due to workload prediction overhead, WACI-HJ proves effective in optimizing query performance.

WACI-HJ also shows significant energy-efficiency, achieving a 23% improvement in RAM utilization and an 18% enhancement in I/O operations, reducing memory and data access overhead. CPU utilization sees a minor 1% gain, highlighting WACI-HJ effectiveness in resource optimization for edge environments.

5.5.2 Qualitative Comparison

CI-HJ (Li and Xu, 2021) handles scaled and skewed data but scans unnecessary cachelines, lacks real-time processing, predictive features, and energy-efficiency, performing better on synthetic datasets than benchmarks like TPC-H.

WACI-HJ being workload-aware, handles scaled and skewed data avoids unnecessary scans, supports real-time processing, includes predictive capabilities, and is energy-efficient. It excels with skewed data and demonstrates robust performance on benchmarks like TPC-H and real-world datasets like MARTA, making it more versatile and effective.

6 CONCLUSIONS

WACI-HJ optimizes real-time edge query processing by accelerating hash joins using the workload-aware approach, where the upcoming workloads are predicted well in advance. Experimental results show significant reductions in PCR and QET, with 18% and 5% improvements for the benchmark dataset, respectively. The algorithm scales well, maintaining stable PCR across data sizes upto 10x and varying data skewness. In resource-constrained environ-

ments, WACI-HJ achieves gains of 2% in CPU, 27% in RAM, and 18% in I/O utilization. Testing with a real-world dataset shows a 54% improvement in PCR and 10% in QET. Energy-efficiency experiments confirm gains of 1% in CPU, 38% in RAM, and 49% in I/O. By optimizing data access and incorporating predictive capabilities, WACI-HJ reduces query latency and conserves energy, making it ideal for resource-constrained edge applications.

REFERENCES

Google Colab. Available: https://colab.research.google.com/ Accessed: June, 2024.

IoT Data Growth. Available: https://www.gartner.com/en/documents/3996804 Accessed: June, 2024.

IoT Devices Growth. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/#:~:text=The%20number%20of%20Internet%20of,billion%20IoT%20devices%20in%202030. Accessed: June, 2024.

MARTA Dataset. Available: https://data.world/brentbrewington/marta-hackathon Accessed: June, 2024.

MonetDB. Available: https://www.monetdb.org/ Accessed: June, 2024.

TPC-D Dataset. Available: https://github.com/gunaprsd/SkewedDataGenerator Accessed: June, 2024.

TPC-H Dataset. Available: http://www.tpc.org/tpch/ Accessed: June, 2024.

VScode. Available: https://code.visualstudio.com/ Accessed: June, 2024.

Awaysheh, F. M., Tommasini, R., and Awad, A. (2023). Big data analytics from the rich cloud to the frugal edge. In *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*, pages 319–329. IEEE.

Barber, R., Lohman, G., Pandis, I., Raman, V., Sidle, R., Attaluri, G., Chainani, N., Lightstone, S., and Sharpe, D. (2014). Memory-efficient hash joins. *Proceedings of the VLDB Endowment*, 8(4):353–364.

Bilal, K., Khalid, O., Erbad, A., and Khan, S. U. (2018). Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. *Computer Networks*, 130:94–120.

Calheiros, R. N., Masoumi, E., Ranjan, R., and Buyya, R. (2014). Workload prediction using arima model and its impact on cloud applications' qos. *IEEE transactions on cloud computing*, 3(4):449–458.

Chen, S., Ailamaki, A., Gibbons, P. B., and Mowry, T. C. (2007). Improving hash join performance through prefetching. *ACM Transactions on Database Systems (TODS)*, 32(3):17–es.

Gandhi, K. and Bhise, M. (2019). Affinity-based fragmentation for sensor data. In *2019 IEEE 16th India Council International Conference (INDICON)*, pages 1–4. IEEE.

Gandhi, K., Pandat, A., and Bhise, M. (2021). Experiments on static data summarization techniques. In *2021 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE)*, pages 17–20. IEEE.

Li, G., Yao, Y., Wu, J., Liu, X., Sheng, X., and Lin, Q. (2020). A new load balancing strategy by task allocation in edge computing based on intermediary nodes. *EURASIP Journal on Wireless Communications and Networking*, 2020:1–10.

Li, Y. and Xu, W. (2021). Utilizing the column imprints to accelerate no-partitioning hash joins in large-scale edge systems. *Transactions on Emerging Telecommunications Technologies*, 32(6):1–17.

Ma, L., Van Aken, D., Hefny, A., Mezerhane, G., Pavlo, A., and Gordon, G. J. (2018). Query-based workload forecasting for self-driving database management systems. In *Proceedings of the 2018 International Conference on Management of Data*, pages 631–645.

Meruje Ferreira, L. M., Coelho, F., and Pereira, J. (2024). Databases in edge and fog environments: A survey. *ACM Computing Surveys*, 56(11):1–40.

Patel, M. and Bhise, M. (2023). Muar: Maximizing utilization of available resources for query processing. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)*, pages 176–183. IEEE.

Schmidt, T., Kipf, A., Horn, D., Saxena, G., and Kraska, T. (2024). Predicate caching: Query-driven secondary indexing for cloud data warehouses. In *Proceedings of the 2024 ACM SIGMOD International Conference on Management of Data*, pages 347–359.

Shah, K., Gandhi, A., Gandhi, K., and Bhise, M. (2024). Workload prediction for edge computing. In *Proceedings of the 25th International Conference on Distributed Computing and Networking*, pages 286–291.

Sun, X. and Ansari, N. (2017). Green cloudlet network: A distributed green mobile cloud network. *IEEE network*, 31(1):64–70.

Symeonides, M., Trihinas, D., Georgiou, Z., Pallis, G., and Dikaiakos, M. (2019). Query-driven descriptive analytics for iot and edge computing. In *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pages 1–11. IEEE.

Vitorovic, A., Elseidy, M., and Koch, C. (2016). Load balancing and skew resilience for parallel joins. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 313–324. Ieee.

Xu, R., Palanisamy, B., and Joshi, J. (2018). Queryguard: Privacy-preserving latency-aware query optimization for edge computing. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 1097–1106. IEEE.