A Hybrid Framework for Resource-Efficient Query Processing by Effective Utilization of Existing Resources

Mayank Patel[1][a] and Minal Bhise[2][b]
[1]*Department of Information & Communication Technology, Adani University, Shantigram, Gujarat, India*
[2]*Distributed Databases Group, Dhirubhai Ambani Institute of Information and Communication Technology, India*

Keywords: Big Data Partitioning, Optimize Resource Utilization, Raw Data Query Processing, Real-Time Dynamic Resource Allocation, Task Scheduling.

Abstract: Scientific experiments and contemporary applications generate substantial volumes of data daily, posing a challenge for traditional database management systems (DBMS) that expend considerable time and resources on data loading. In-situ engines offer a distinct advantage by enabling immediate querying on raw data. Researchers have observed that resources are often underutilized during data loading. In contrast, in-situ engines spend ample time and resources in reparsing required data multiple times. Allocating query specific resources is another challenging task that must be addressed to reduce overall workload execution time and resource utilization. This research paper introduces a novel approach called the Resource Availability & Workload-aware Hybrid Framework (RAW-HF), designed to enhance the efficiency of data querying by judiciously utilizing optimal resources in systems comprising an in-situ engine and DBMS. RAW-HF incorporates modules that facilitate the optimization of resources necessary for executing a given workload, striving to maximize the utilization of available resources. The effectiveness of RAW-HF is demonstrated using the scientific dataset Sloan Digital Sky Survey (SDSS) and Linked Observation data (LOD). Comparative analysis with the state-of-the-art workload-aware partial loading technique (WA) reveals that RAW-HF excels in allocating query-specific resources and implementing resource-aware task scheduling. Results from the study indicate that RAW-HF outperforms WA, reducing workload execution time by 26%. It also reduces CPU and IO resource utilization by 26% and 25% compared to WA at a cost of 33% additional RAM.

1 INTRODUCTION

The escalating pace of data generation from diverse sources, including modern applications, scientific experiments, and IoT applications, underscores the challenge of managing perishable resources effectively. The volume of Astronomy datasets like the Sloan Digital Sky Survey (SDSS) has increased by 233 times since DR-1 to DR-17(Ahumada et al., 2020). NASA's Earth Observing System (EOS) collects more than 3.3TB of data every day(Guo et al., 2016).

Research indicates that the bottleneck in data processing often lies in slow IO devices, particularly magnetic disks, leading to under-utilization of existing CPU resources by most DBMS(Dziedzic et al., 2017). Shockingly, data centers observe a mere 12% CPU utilization, elevating operational costs in cloud and distributed environments (Ailamaki, 2015; Maximilien et al., 2022). Parallelizing data loading proves ineffective for systems with disk-based storage devices (Dziedzic et al., 2017; Patel and Bhise, 2023b).

HTAP or hybrid systems utilize additional resources in processing the same dataset twice. Widely used DBMSs like Postgres, MySQL, Oracle, AutoSteer (Anneser et al., 2023) and other open-source systems do not monitor the utilization of available resources or consider them during query planning. The existing systems or techniques proposed to address resource optimization or maximization issues have been developed for specific DBMSs, which may not work for most DBMSs, hybrid systems, or cloud vendor (Pimpley et al., 2022; Kaviani et al., 2019).

This paper addresses the issues of under-utilized resources and slow IO devices by focusing on the availability of resources. By monitoring resource utilization in real-time, the proposed approach aims to mitigate the shortcomings of existing works (Li et al., 2022; Pimpley et al., 2022; Viswanathan et al., 2018; Patel and Bhise, 2023a). Specifically, it allocates

[a] https://orcid.org/0000-0002-7804-4017
[b] https://orcid.org/0000-0003-4364-3930

Patel, M. and Bhise, M.
A Hybrid Framework for Resource-Efficient Query Processing by Effective Utilization of Existing Resources.
DOI: 10.5220/0012691600003690
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 26th International Conference on Enterprise Information Systems (ICEIS 2024) - Volume 1*, pages 337-344
ISBN: 978-989-758-692-7; ISSN: 2184-4992
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

query-specific resources and optimizes task scheduling, presenting a novel solution to enhance efficiency in data querying. This shift in perspective, from the under-utilization of resources to the effective management of resources, marks a significant contribution in the domain of database management and system optimization.

1.1 Motivation

The majority of existing Database Management Systems (DBMSs) and contemporary cloud resource utilization strategies encounter challenges in effectively leveraging all available resources(Ailamaki, 2015; Maximilien et al., 2022). The optimization of hardware resource utilization has become a focal point, given the significant demand for cloud or hardware resources in querying large datasets to reduce operational costs. Notably, external Machine Learning (ML)-based solutions demand substantial time and resources for filtering, analyzing resource utilization data, and training ML models (Ailamaki, 2015; Pimpley et al., 2022). However, the challenge lies in the fact that most systems do not account for real-time availability of existing resources when scheduling tasks or dynamically allocating query-specific resources in a dynamic operational environment.

1.2 Paper Contributions

- This study consolidates modules like Raw Data Query Processing (RQP), Resource Monitoring (RM), Optimizing Required Resources (ORR), and Maximizing Utilization of Existing Resources (MUER) to construct RAW-HF framework.
- ORR module integrated lightweight query complexity, workload, & storage utilization aware partitioning algorithms designed for hybrid systems. MUAR integrated resource utilization aware task scheduling and resource allocation algorithms.
- RAW-HF results have been compared with state-of-the-art in-situ engine(Alagiannis et al., 2012), row store DBMS PostgreSQL(Pos, 2022), and workload-aware Partial Loading technique(Zhao et al., 2015) using real-world scientific datasets.

2 RELATED WORK

Traditional database management systems necessitate loading the entire dataset before executing a query, while in-situ engines expedite results by bypassing data loading and processing only essential raw data upon query arrival. However, in-situ engines exhibit slower query execution time (QET) compared to DBMSs due to reparsing of raw data. Notably, NoDB (Alagiannis et al., 2012) & Slalom (Olma et al., 2020) proposed caching and indexing processed raw data in main memory to reduce QET. Despite these advancements, caching techniques encounter occasional reparsing issues and high utilization for large datasets.

Invisible loading proposed loading processed data into column store DBMS generated as a side effect of executing queries on raw datasets to eliminate reparsing (Abouzied et al., 2013). SCANRAW, one of the first hybrid systems, suggested monitoring CPU and IO to speculatively load additional data into DBMS to utilize idle resources, in parallel to querying raw data using an in-situ engine (Cheng and Rusu, 2015). Regardless, both invisible loading and SCANRAW are prone to processing and loading the entire dataset when workload queries access database attributes even once.

To address this, cost-aware techniques have been introduced, calculating the cost of accessing raw data files and loading only frequently accessed partitions into DBMS (Patel et al., 2022; Zhao et al., 2015). The Query Complexity Aware (QCA) technique reduces loaded data by loading attributes required by complex queries, maintaining attributes accessed by zero-join simple queries in raw format (Patel and Bhise, 2022). These techniques aim to reduce data loading time, researchers have also partitioned datasets into smaller, workload-aware partitions to reduce query execution time (Padiya and Bhise, 2017; Tang et al., 2020).

As cloud usage rises, attention has shifted towards optimizing hardware resource utilization to reduce costs associated with each query. QROP (Query and Resource Optimization) proposed considering the resources required by each query during query planning to minimize costs (Viswanathan et al., 2018). PCC (Performance Characteristic Curve), derived from past resource utilization and query execution time data, is employed by QROP to allocate optimal resources to each query individually, reducing costs through a trade-off in query response time (Pimpley et al., 2022). An elastic resource management technique traded off OLTP throughput to enhance OLAP query time by allocating more resources to OLAP queries (Raza et al., 2020).

Despite advancements, techniques, and systems featuring optimizations for resources used by data loading and query execution tasks, task scheduling, and query-specific resource allocation based on real-time hardware resource availability remain scarce.

3 RAW-HF: RESOURCE AVAILABILITY & WORKLOAD AWARE HYBRID FRAMEWORK

This section discusses the proposed RAW-HF framework integrated with lightweight partitioning, task scheduling, and query-specific resource allocation algorithms. The following subsections discuss the architecture and modules of RAW-HF.

3.1 RAW-HF Architecture

The RAW-HF consists of four modules. Each module serves a task dedicated to it. Figure 1 shows how all the framework modules connect and communicate.

- **Raw Data Query Processing (RQP) module:** RQP module processes the given dataset and workload automatically for individual and hybrid systems. It can handle raw data storage, data loading, and query processing tasks for individual in-situ engine, DBMS, HTAP, or hybrid systems. The combination of an in-situ engine (NoDB) and DBMS (PostgreSQL) is used to build a hybrid system because in-situ engines eliminate DLT while DBMS performs complex queries faster.
- **Resource Monitoring (RM) module:** Monitors resources utilized by each workload task and filters essential information for each query obtained from RM tools like *top* and *iotop* for faster online or offline analysis. This real-time resource utilization data is used by MUER module.This module also records past query execution information for ORR and MUER modules to make partitioning and resource allocation decisions.
- **Optimizing Required Resources (ORR) module:** Reduces unnecessary utilization of resources. It optimizes data loading and query execution time by combining query complexity, workload, & storage utilization aware lightweight partitioning for a given dataset & workload.
- **Maximizing Utilization of Existing Resources (MUER) module:** MUAR considers the real-time availability of resources to utilize resources efficiently and reduces total workload execution time by allocating additional resources. It schedules data loading tasks in parallel to querying raw data. It identifies complex queries (CQs) and reduces QET for CQs by increasing *work_memory*.

RAW-HF core algorithms reside in ORR and MUER modules. Therefore, both of them have been explained in detail below.

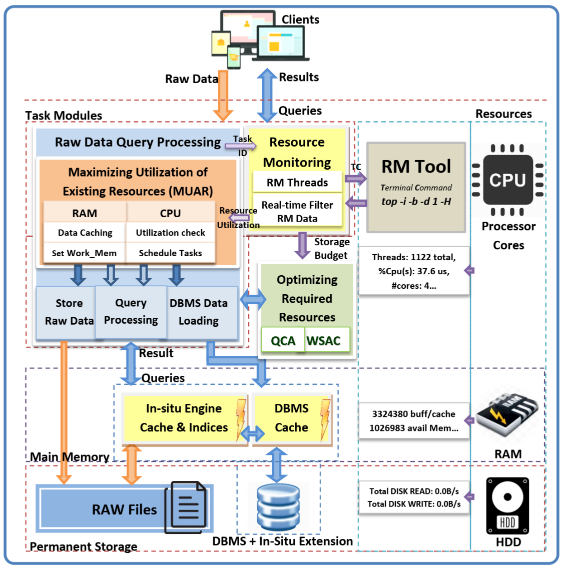


Figure 1: RAW-HF Architecture.

3.1.1 Optimizing Required Resources (ORR)

The proposed workload and query complexity aware algorithms of ORR use lightweight query identification and partitioning steps to partition the dataset for a hybrid system. The idea behind the query complexity aware algorithm is to partition the dataset and distribute the workload in such a way that queries performing faster on a given tool can be allocated to that tool. The analysis has shown that zero-join queries perform faster while multiple joins queries are slower in in-situ engines than in traditional DBMS.

The Query Complexity Identification (QCI) algorithm first identifies the simple queries (SQ) and complex queries (CQ) type queries stored in the workload list. It uses the simple logic of counting no. of tables present in the query statement. The query is classified as complex; when two or more table instances are found in the query statement. The single table instance queries are classified as simple queries. The GRA function groups attributes accessed by SQs and CQs in two different lists. The SQ partition is *QT_P0*, and the CQ partition is *QT_P1* after the first round of partitioning. SQ partition can be stored in raw format, while the CQ partition needs to be loaded in DBMS. ORR further refines the partitions based on different storage budget B in QC function. The partially covered queries (PCQ) list is new input query type *QT*. QC function can be called until all attributes are covered for different storage budgets. Most frequent queries and storage budget list can be sorted in descending order to reduce iterations & cover frequent queries first.

Algorithm 1: ORR Module Partitioning.

Data: *w_l* = Workload List; *QT* = Query Types Dictionary; *q_l* = Query List; *B* = Storage budget B in MB; *que_d* = Dictionary of Queries; *s_d* = Schema Dictionary; *QT_P* = Query Type Partitions; *QT_P'*= Final Query Type Partitions for given budget B; *q_l* = Query List; *PCQ* = Partially Covered Queries List; *ca_l* = List of covered Attributes; *rqa_l* = List of Remaining Query Attributes; *cq_l* = List of covered Queries;
Result: SQ-Raw, CQ-DB & CAP Partitions;

```
        # Query Complexity Identification
1. def QCI(w_l, que_d, s_d):
2.      For each task T in w_l do
3.          If T.Statement has multiple tables
4.              QT[T.Q_ID] = 1
5.          Else
6.              QT[T.Q_ID] = 0
7.      End
8.      Get QT_P0, QT_P1 = GRA(que_d, QT)
9. Return final partitions QT_P';

        #Grouping of Attributes
10. def GRA(que_d, QT)
11.     For each query i in que_d:
12.         For each attribute j in que_d[i]:
13.             If QT[i] == 0
14.                 Add j in QT_P0
15.                     If budget B is limited
16.                         QT_P' = QC(i, ca_l, cq_l, B)
17.Repeat above step until PCQ is empty for different B
18.             Else
19.                 Add j in QT_P1
20.         End
21.     End
22. Return QT_P0, QT_P1,QT_P';

    #Grouping Attributes based on Budget B
23. def QC(i, ca_l, cq_l, B)
24.     If (SUM(size of all attributes of query que_d[i]))¡ B
25.         For each attribute A in que_d[i]
26.             If A is not in ca_l & size of A¡remaining budget B
27.                 Add A in rqa_l list
28.         If size of rqa_l ¡ remaining budget B
29.             Move all attributes of rqa_l in ca_l & update B
30.             Add query q in cq_l
31.         Else
32.             Add query q in PCQ
33. Return ca_l, cq_l, rqa_l;
```

3.1.2 Maximizing Utilization of Existing Resources (MUER)

This module implements Maximizing Utilization of Available Resources (MUAR) technique to improve the utilization of existing resources. MUAR considers real-time resource monitoring values of CPU, RAM, and IO resource utilization stored in the global structure *RM_AR*. *RM_AR* is continuously updated by the resource monitoring module. MUAR adds a new task for processing if all three values of *RM_AR* are greater than the minimum required CPU, RAM & IO resources stored in *Min_RR*. Before executing the query in the new thread, the *WM_Query* function is called to set the work memory for each complex query to increase RAM utilization.

The *WM_Query* function first counts the number of joins used in the given query and stores the count in *J_C*. The *WM_Value* in line 18 calculates the work memory value for new queries based on available memory *RM_AR.RAM*, process count (*P_C*), total RAM (*TR*), and join count *J_C*. The first part of the equation divides the available RAM between the maximum processes that the available CPU cores can handle. The second part defines the maximum RAM that can be assigned to a thread, while the third part helps in allocating more RAM to complex queries considering join count *J_C*. MUAR also tries to estimate required work memory considering previous *work_mem*, disk writes, current & past record count ratio for frequent queries to achieve the best QET.

4 EXPERIMENTAL SETUP

This section provides details of experimental setup.

4.1 Hardware & Software Setup

The experimental machine uses a quad-core Intel i5-6500 CPU clocked at 3.20GHz. It has 16GB of RAM. The operating system of the machine is running a 64-bit Ubuntu 18.04 LTS. The machine has a 500GB SATA hard disk drive to store raw datasets and DBMS databases. The disk rotation speed is 7200RPM. The RAW-HF framework is implemented by combining and updating the raw data query processing, resource monitoring, ORR, and MUER modules. The updated framework uses Eclipse to run Java code connected with state-of-the-art open-source DBMS PostgreSQL and NoDB. NoDB has data caching capability and executes SQL queries on raw files. The *top* and *iotop* tools provide real-time resource utilization data of CPU, RAM & IO resources to the RM module.

4.2 Dataset & Query Set

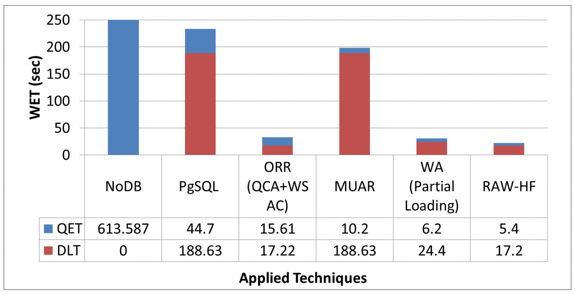
Data release 16 of Sloan Digital Sky Survey (SDSS) (Ahumada et al., 2020) has been used to check the ORR and MUER phases of RAW-HF. 19GB partition having 4M records of *PhotoPrimary* view having 509 attributes has been used to represent SDSS dataset. We have extracted the top 1000 unique queries executed on *PhotoPrimary* view. The queries have been grouped based on the similarity of the attributes and query type, forming 12 query groups. One representative query has been chosen from each group.

The 8GB partition with 35M records containing descriptions of blizzard and hurricane observations has been extracted from Linked Observation Data (LOD) (Patni et al., 2010; Padiya and Bhise, 2017) (Ahumada et al., 2020). LOD is a benchmark RDF triple store dataset. It has been used to investigate the performance of the MUER module of RAW-HF. Nine queries out of 16 queries of LOD workload have 5 joins. Queries with multiple self-joins have been considered because these queries need to process complex join operations, requiring significant resources.

5 RAW-HF RESULTS

This section presents the results of RAW-HF after combining all techniques proposed in ORR & MUER.

5.1 RAW-HF: Workload Execution Time



	NoDB	PgSQL	ORR (QCA+WSAC)	MUAR	WA (Partial Loading)	RAW-HF
🟦 QET	613.587	44.7	15.61	10.2	6.2	5.4
🟥 DLT	0	188.63	17.22	188.63	24.4	17.2

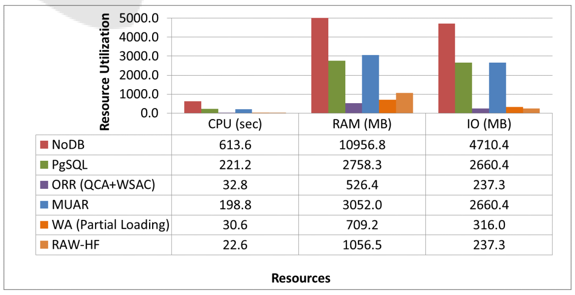
Figure 2: WET: Comparison.

Figure 2 shows the comparison of all the techniques with RAW-HF. First column shows the raw data query processing time using NoDB, which has zero data loading time. The 2nd column shows WET time required by traditional DBMS PgSQL. The 3rd and 4th columns show the WET results for individual ORR and MUER phases of RAW-HF. RAW-HF takes only 22.6 sec to complete the execution compared to 30.6sec required by workload aware partitioning techniques like Partial Loading (WA)(Zhao et al., 2015). It can be observed that the combination of all techniques achieved total reduction of 96.32% using RAW-HF compared to NoDB and 26.14% compared to the WA. RAW-HF benefits from low DLT achieved by only loading attributes used by complex queries in the ORR phase for SDSS.

5.2 RAW-HF: Resource Utilization

Figure 3 shows the comparison of resources utilized by ORR, MUAR, and RAW-HF (ORR+MUER) with NoDB , PostgreSQL , and WA. Here, the 1M records dataset used in experiment utilized 4.7GB of space on IO device. MUAR results showed that CPU utilization time is reduced by only 6.34% because most of the time is spent in data loading process compared to PgSQL. MUAR can only utilize other CPU cores to execute read queries in parallel. Figure 3 confirms that CPU utilization is reduced by 77% only during query processing tasks due to parallel processing.

The QCA with WSAC technique in ORR reduced the required DB partition size(IO) by 91.08%, reducing WET, CPU, and RAM utilization by 85.9%, 85.1%, and 80.9%. WA (Partial loading) reduced the CPU and RAM utilization by 81.3% and 86.4%. The RAW-HF experiments combined ORR and MUAR techniques, which showed a 32.8% increase in RAM utilization compared to the Partial Loading technique due to parallel processing of queries. However, RAW-HF improved DLT, QET, WET, CPU, and DB Size(IO) requirements by 29.5%, 12.9%, 26.14%, 26.14%, 24.92% compared to Partial Loading technique (Zhao et al., 2015) executing all read queries in parallel after data loading is complete.

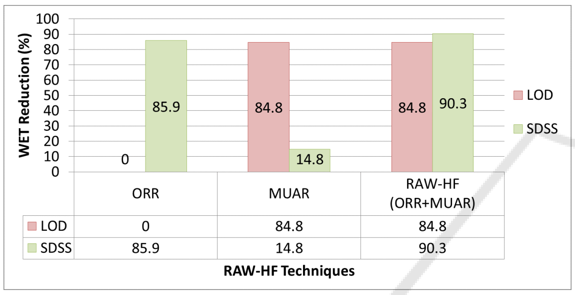


	CPU (sec)	RAM (MB)	IO (MB)
🟥 NoDB	613.6	10956.8	4710.4
🟩 PgSQL	221.2	2758.3	2660.4
🟪 ORR (QCA+WSAC)	32.8	526.4	237.3
🟦 MUAR	198.8	3052.0	2660.4
🟧 WA (Partial Loading)	30.6	709.2	316.0
🟧 RAW-HF	22.6	1056.5	237.3

Figure 3: RAW-HF: Resource Utilization.

5.3 RAW-HF for Different Datasets

This section discusses the impact of RAW-HF on WET for different types of datasets like LOD & SDSS. Table 1 compares LOD and SDSS datasets based on the Fraction of Attributes Accessed (FAA) by workload queries and the Fraction of Attributes Loaded (FAL) by RAW-HF. It can be seen that SDSS is a broad table dataset. All the SDSS workload queries access only 10.6% of attributes. On the other hand, LOD dataset is a narrow table dataset containing only three attributes. Due to fewer attributes, almost all queries use two or more attributes. The impact of broad and narrow tables and queries accessing only a small part or entire of the dataset can be seen in the ORR results for SDSS in Figure 4. MUAR allocated maximum available resources to CQs and reduced WET significantly for the LOD dataset because 87% of LOD workload queries consist of two or more joins. In comparison, SDSS had only one query with more than one join.



	ORR	MUAR	RAW-HF (ORR+MUAR)
🟥 LOD	0	84.8	84.8
🟩 SDSS	85.9	14.8	90.3

Figure 4: Impact of RAW-HF on WET for LOD & SDSS datasets.

RAW-HF only loads attributes required by complex queries to reduce DLT time. This helps datasets like SDSS, which requires only a small fraction of the dataset (10.6%) to answer queries by loading only 6.7% of attributes. The remaining 93.3% of attributes are not loaded into DBMS, reducing WET by 85.9% for the SDSS dataset. On the other hand, vertical partitioning cannot help datasets like LOD that access 100% of attributes. Therefore, the WET reduction achieved by applying ORR phase is 0% for LOD. However, the MUAR in MUER phase achieves 84.8% reduction in WET by efficiently utilizing existing CPU and RAM resources for complex queries.

5.4 RAW-HF: Dynamic Resource Allocation

Figure 5 illustrates the comparison between the dynamic task and resource allocation of RAW-HF and static allocation in PostgreSQL. The results are obtained through the execution of the LOD workload on a single core (1 thread) and multiple cores (three threads), varying the values of *work_memory*. In contrast to RAW-HF's query-specific memory allocation, the static parallel execution method maintains a constant value of *work_memory* across different queries.

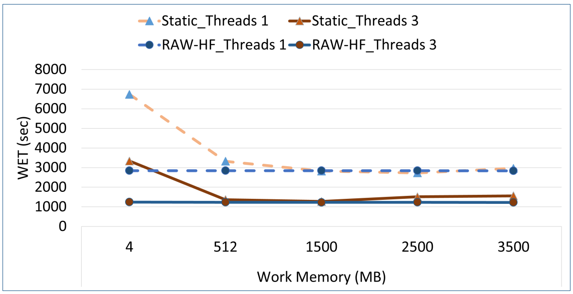


Figure 5: Impact of RAW-HF on WET compared to Static task & resource allocation.

The comparison reveals that the static multi-thread execution of the workload, with a default 4 MB *work_memory* allocation, takes more time than RAW-HF's single-thread dynamic memory allocation. Increasing the *work_memory* allocation to 3500 MB with the static approach resulted in a 22% increase in workload execution time compared to a 1500 MB allocation. This increase is attributed to the over-allocation of memory for certain queries, leading to reduced memory availability for complex queries running in parallel. RAW-HF reduces WET by 4-67% compared to the static approach.

6 COMPARISON WITH STATE-OF-THE-ART

This section presents a comparison of the RAW-HF technique with other state-of-the-art techniques and tools like NoDB (Alagiannis et al., 2012), PostgreSQL (Pos, 2022), WA (Partial Loading) (Zhao et al., 2015) , and PCC (Pimpley et al., 2022). NoDB is an open-source in-situ processing engine with main memory caching and indexing features (Alagiannis et al., 2012). Partial Loading technique (Zhao et al., 2015) loaded only **10.6%** of original dataset into DBMS, which reduced the WET time by 88.1% compared to NoDB. In comparison, RAW-HF loads only **6.7%** of dataset. Only RAW-HF can execute join queries on data residing in DBMS and CSV files. Unlike PCC, RAW-HF allocates appropriate resources to ad-hoc queries.

Table 2 presents a comparison of state-of-the-art raw data query processing techniques with RAW-HF. Resource Utilization Aware (RUA) shows whether the technique considered resource utilization information or not. The Multi-Format Join column represents whether the tool can execute join queries on data residing in raw and database formats. PDC uses a static partition of main memory (50%) to keep lookup tables. PCC needs historical data for allocating ap-

Table 1: ORR: Fraction of Attributes Accessed (FAA) and Loaded (FAL).

	Total Attributes	Accessed Attributes	FAA (%)	Loaded Attributes	FAL (%)	DLT (%)	QET (%)	WET (%)
SDSS	509	54	10.6	34	6.7	90.9	87.9	85.9
LOD	3	3	100.0	3	100.0	0	0	0

Table 2: RAW-HF Technique Comparison with State-of-the-art.

#	**Technique / Tool**	**Partitioning**	**DBMS Data %**	**Workload Aware**	**Ad-hoc queries**	**RUA**	**Multi-format Join**	**Remarks**
1	NoDB (Alagiannis et al., 2012)	None	0	No	NA	No	No	Required more memory. High QET.
2	PostgreSQL (Pos, 2022)	None	100	No	NA	No	No	High DLT, Low QET & memory.
3	WA (Zhao et al., 2015)	VP	10.6	Yes	No	Yes	No	Technique Not Lightweight
4	PCC (Pimpley et al., 2022)	-	100	Yes	No	Yes	No	Resource Intensive, for freq. queries
5	**RAW-HF (Hybrid)**	**VP**	**6.7**	**Yes**	**Yes**	**Yes**	**Yes**	**Lightweight, works for ad-hoc & freq. queries**

Table 3: RAW-HF Performance Parameters Comparison.

#	**Technique/ Tool**	**Query Performance %**			**Resource Utilization %**		
		DLT (sec)	QET (sec)	WET (sec)	CPU (sec)	RAM (MB)	IO (MB)
1	NoDB	0	**613.59 (99.12%)**	**613.59 (96.32%)**	**613.59 (96.32%)**	**10956.8 (90.36%)**	**4710.4 (94.96%)**
2	DBMS (PostgreSQL)	**188.63 (90.88%)**	**44.7 (87.92%)**	**233.33 (90.31%)**	**233.33 (89.78%)**	**2758.3 (61.70%)**	**2660.4 (91.08%)**
3	WA (Partial Loading)	**24.4 (29.51%)**	**6.2 (12.90%)**	**30.6 (26.14%)**	**30.6 (26.14%)**	**709.2 (+32.87%)**	**316.0 (24.92%)**
4	**RAW-HF**	**17.2**	**5.4**	**22.6**	**22.6**	**1056.5**	**237.3**

propriate resources to each query. However, it does not work well for ad-hoc queries. In comparison, although RAW-HF performs workload aware partitioning, resource allocation is done based on query complexity. Therefore, RAW-HF is capable of allocating appropriate resources to ad-hoc queries.

Figure 2 & 3 presented comparison of these techniques with RAW-HF for SDSS dataset. Table 3 shows the RAW-HF performance parameters comparison with state-of-the-art techniques.

7 CONCLUSION

This work presented a Resource Availability and Workload aware Hybrid Framework (RAW-HF) to process raw datasets efficiently. It optimized required resources using ORR module developed by integrating lightweight algorithms of Query Complexity Aware (QCA) and Workload and Storage aware Cost-based (WSAC) techniques. This work also addresses the issue of underutilized resources during data loading and query processing using MUAR module, considering the availability of hardware resources in real-time. RAW-HF allows the execution of join queries on data stored in DBMS and raw format.

RAW-HF never loads partitions used by simple queries, reducing DBMS data loading requirements by 93.3% for SDSS. ORR phase works better for broad table datasets like SDSS, while MUAR is capable of improving WET for workloads with complex multi-join queries like LOD. RAW-HF reduced WET by 84.8% for SDSS and 90.3% for LOD compared to the PostgreSQL. It reduced the WET by 26%, 96%, and 90% compared to the state-of-the-art Partial

Loading techniques, NoDB, and PostgreSQL respectively for SDSS.

REFERENCES

(2022). Postgresql: The world's most advanced open source database. https://www.postgresql.org/.

Abouzied, A., Abadi, D. J., and Silberschatz, A. (2013). Invisible loading: Access-driven data transfer from raw files into database systems. In *Proceedings of the 16th International Conference on Extending Database Technology - EDBT '13*, volume 20. ACM Press.

Ahumada, R., Prieto, C. A., and et al. (2020). The 16th Data Release of the Sloan Digital Sky Surveys: First Release from the APOGEE-2 Southern Survey and Full Release of eBOSS Spectra. *The Astrophysical Journal Supplement Series*, 249(1).

Ailamaki, A. (2015). Databases and hardware: The beginning and sequel of a beautiful friendship. *Proceedings of the VLDB Endowment*, 8(12).

Alagiannis, I., Borovica, R., Branco, M., Idreos, S., and Ailamaki, A. (2012). Nodb in action. *Proceedings of the VLDB Endowment*, 5(12).

Anneser, C., Tatbul, N., Cohen, D., Xu, Z., Pandian, P., Laptev, N., and Marcus, R. (2023). Autosteer: Learned query optimization for any sql database. *PVLDB*, 16:12.

Cheng, Y. and Rusu, F. (2015). Scanraw: A database metaoperator for parallel in-situ processing and loading. *ACM Transactions on Database Systems*, 40(3).

Dziedzic, A., Karpathiotakis, M., Alagiannis, I., Appuswamy, R., and Ailamaki, A. (2017). DBMS Data Loading: An Analysis on Modern Hardware. In *Data Management on New Hardware, volume 10195 LNCS, page 95–117*. Springer International Publishing.

Guo, H., Wang, L., and Liang, D. (2016). Big earth data from space: a new engine for earth science. *Science Bulletin*, 61(7).

Kaviani, N., Kalinin, D., and Maximilien, M. (2019). Towards serverless as commodity. In *Proceedings of the 5th International Workshop on Serverless Computing - WOSC '19*. ACM Press.

Li, Y., Wang, L., Wang, S., Sun, Y., and Peng, Z. (2022). A resource-aware deep cost model for big data query processing. *Proceedings - International Conference on Data Engineering*.

Maximilien, M., Hadas, D., Danducci II, A., and Moser, S. (2022). The future is serverless - ibm developer.

Olma, M., Karpathiotakis, M., Alagiannis, I., Athanassoulis, M., and Ailamaki, A. (2020). Adaptive partitioning and indexing for in situ query processing. *The VLDB Journal*, 29(1).

Padiya, T. and Bhise, M. (2017). DWAHP: Workload Aware Hybrid Partitioning and Distribution of RDF Data. In *In Proceedings of the 21st International Database Engineering and Applications Symposium (IDEAS)*. ACM.

Patel, M. and Bhise, M. (2022). Query complexity based optimal processing of raw data. In *2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC)*. IEEE.

Patel, M. and Bhise, M. (2023a). MUAR: Maximizing Utilization of Available Resources for Query Processing. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)*.

Patel, M. and Bhise, M. (2023b). Resource monitoring framework for big raw data processing. *International Journal of Big Data Intelligence, Inderscience*, 9(1).

Patel, M., Yadav, N., and Bhise, M. (2022). Workload aware cost-based partial loading of raw data for limited storage resources. In *Futuristic Trends in Networks and Computing Technologies FTNCT, Lecture Notes in Electrical Engineering*, vol 936. Singapore: Springer Nature Singapore.

Patni, H., Henson, C., and Sheth, A. (2010). Linked sensor data. In *2010 International Symposium on Collaborative Technologies and Systems*. IEEE.

Pimpley, A., Li, S., Sen, R., Srinivasan, S., and Jindal, A. (2022). Towards optimal resource allocation for big data analytics. In *International Conference on Extending Database Technology, EDBT*.

Raza, A., Chrysogelos, P., Anadiotis, A. C., and Ailamaki, A. (2020). Adaptive htap through elastic resource scheduling. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. ACM.

Tang, H., Byna, S., Dong, B., and Koziol, Q. (2020). Parallel query service for object-centric data management systems. *Proceedings - 2020 IEEE 34th International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2020*.

Viswanathan, L., Jindal, A., and Karanasos, K. (2018). Query and resource optimization: Bridging the gap. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE.

Zhao, W., Cheng, Y., and Rusu, F. (2015). Vertical partitioning for query processing over raw data. In *Proceedings of the 27th International Conference on Scientific and Statistical Database Management*, volume 29-June-20. ACM.