Current Trends in Technology and Science
www.ctts.in, ISSN: 2279-0535, Volume: 12, Issue: 6, November 2023

Efficient Big Data Processing and Clustering using AsterixDB

Shani Prakash[1], Gagan Sharma[2], Narendra Parmar[3],
[1,2,3]Department of Computer Science and Engineering,
RKDF University, Bhopal, India

Abstract

As the volume and complexity of data continue to surge, the need for efficient big data processing and clustering methodologies becomes increasingly critical. This research paper presents a comprehensive exploration of the utilization of AsterixDB, an open-source, scalable, and highly extensible big data management system, to achieve efficient data processing and clustering. The study delves into the unique features and capabilities of AsterixDB, examining its ability to handle large-scale datasets with a focus on scalability, performance, and adaptability. This research introduces an innovative software stack to construct scalable Big Data systems. The focus is specifically on two key components within this stack. First, Hyracks stands out as a novel partitioned-parallel runtime layer, offering an efficient and versatile model for executing data-processing tasks across a cluster of commodity machines. Second, Algebricks is a compiler framework crucial in constructing compilers for high-level declarative languages tailored for parallel processing, all built on top of the Hyracks infrastructure.

Keywords: Big Data, MapReduce, Clustering, AsterixDB, Hyracks

1 Introduction

Initially, programmers used the MapReduce system to implement tools to quickly comb through very large amounts of data stored on clusters of commodity machines. Over time, it was clear that writing these tools in imperative languages, for analysis, was not the best use of programmers' time. To further improve programmer productivity, higher-level declarative languages were implemented (Sawzall [1] and Tenzing [2], at Google, and Hive [3], at Facebook). The runtime layer used to process queries by these systems was still the MapReduce platform. While MapReduce was a simple model for programmers writing simple tools, we believed that it was the wrong layer for query language compilers to target, as their runtime layer. Sticking to the MapReduce abstraction forced queries to be evaluated as a sequence of MapReduce jobs with intermediate data materialized in a distributed filesystem, thus using a whole lot of unneeded resources. Several decades of database research has shown that pipelined architectures for executing queries consume fewer system resources, making better use of available hardware. One other problem with the MapReduce model was the reliance on unary operators for implementing Map and Reduce functions.

This limitation of the model forces the need to express binary operations (such as joins) in an unnatural convoluted fashion. For example, Hive implements join over the MapReduce platform by expressing the operation as a grouping of the tagged union of the two relations being joined. Another type of inefficiency introduced by the MapReduce platform has to do with the limited types of query processing algorithms that can be implemented within the framework, owing to the strict map-followed-by-sort-followed-by-reduce contract. For example, aggregation in MapReduce is implemented using a sort-based strategy because sorting is built into the system. A traditional parallel relational database system might have chosen a hash-based strategy to perform aggregation. This choice would require that the platform provide data redistribution primitives that offer more control. More details of these inefficiencies are described in Chapter 4. Based on these observations, Hyracks is designed and implemented, a high-performance parallel dataflow runtime layer.

2 Related Work

Various tools and techniques have been proposed by the researchers to enhance the performance of the big data processing and clustering.

Peng *et al.* [4] suggested a clustering technique for intrusion detection systems called Mini Batch Kmeans with PCA (PMBKM). Both the 10% dataset and the entire dataset, IDS classic dataset KDDCUP99, are tested. Prior to using the PCA approach to lower the dimension and increase the clustering performance, they preprocessed the provided dataset. Furthermore, the transformed dataset is clustered using the Mini Batch Kmeans algorithm. In contrast to Kmeans (KM), Kmeans with PCA (PKM), and Mini Batch Kmeans (MBKM), the experimental findings demonstrated the efficacy and efficiency of their suggested

PMBKM. The most important application for PMBKM is as an intrusion detection system in big data environments.

Ma *et al.* [5] proposed an interdisciplinary architecture aimed at fields such as big data, energy, manufacturing, clustering, and correlation analysis. This architecture serves as a valuable resource for both industrial production and academic research, offering both practical and theoretical insights. While clustering and correlation analysis are widely utilized in energy-intensive manufacturing industries (EIMIs), their combination in studies remains limited. Additionally, the dynamic uncertainty and complexity of continuous production processes constrain data mining and analysis efforts, making it challenging to extract insights from multi-source heterogeneous data. Incorporating advanced technologies into multi-source heterogeneous data mining requires significant infrastructure support. Hence, there's a need to develop a simple yet efficient approach to uncover the underlying connections between various energy and resource variables. To address this, they proposed a big data-driven correlation analysis framework based on clustering analysis.

The traditional algorithm for clustering efficient distributed databases often incurs long processing times and achieves low accuracy. In response to these challenges, Liantian Li [6] proposed a novel algorithm tailored for clustering efficient distributed databases in big data processing scenarios. This algorithm involves computing the eigenvalues of the database and associating the efficient distributed database with similar characteristics. Additionally, it employs a cross-correlation matrix to ensure consistency in cluster labeling. To enhance the performance of the K-means algorithm, the algorithm takes the database to be clustered as input, outputs k clustering centers, and partitions the data into clustering groups. By mapping the database to clustering centers, the algorithm effectively clusters low-dimensional big data. Experimental findings demonstrate that the proposed algorithm can significantly reduce both the runtime and mean square error of data clustering while improving the efficiency and accuracy of the clustering process.

Paulraj *et al.* [7] introduced the Rank-Revealing RRQR-SDM technique, which presents several advantages compared to existing methods. Firstly, by leveraging the inherent low-rank structure, it decreases the computational complexity associated with large datasets. This approach not only uncovers the rank of the input matrix but also facilitates dimensionality reduction and effective data compression. Secondly, through Schur decomposition, it enhances data interpretability by distinguishing between relevant and irrelevant components clearly. This characteristic renders the RRQR-SDM method especially suitable for tasks such as data mining and clustering, where identifying significant features is crucial. Extensive experiments conducted on diverse big data sets validate the superiority of the proposed method in terms of computational efficiency and clustering accuracy over current techniques.

Liu *et al.* [8] proposed Dynamic Cluster Scheduling Algorithm(DCSA) for parallel optimization of big data tasks, leveraging data correlation. Initially, a dynamic data queue is generated based on the server's request database, considering factors such as data item priority and size for data clustering association. Introducing weights ensures equalization of dynamic data items to facilitate multi-channel optimal scheduling. Subsequently, based on data item relevance, a mechanism for optimized data placement aggregates data within the same frame. Following placement, dynamic data is uniformly scheduled to minimize migration costs, considering local data item characteristics as constraints. Through iterative targeting, the optimal scheduling scheme is refined, ultimately achieving multi-channel optimal scheduling. Experimental results demonstrate that the proposed method effectively achieves optimal scheduling of dynamic data.

Tian *et al.* [9] proposed an effective ensemble hierarchical clustering algorithm, leveraging a MapReduce-based clusters clustering technique and a novel similarity criterion. Ensemble clustering involves amalgamating outcomes from diverse single clustering methods, typically yielding superior results due to the amalgamation of multiple learning approaches. Consequently, combining hierarchical clustering methods is anticipated to enhance clustering quality further. Additionally, MapReduce, a model for big data application implementation, is employed to execute hierarchical clustering methods. Simultaneously, sample similarity is assessed using an innovative similarity criterion. The proposed methodology unfolds in three sequential steps. Initially, data undergo clustering via several individual hierarchical clustering methods. Subsequently, hyper-clusters are derived through application of the clusters clustering technique in the second step. Finally, in the third step, final clusters are formed by assigning samples to hyper-clusters. This process concludes the formation of final clusters. Simulation conducted on multiple real-world datasets demonstrates superior performance of the proposed approach compared to algorithms such as CHC and RCESCC.

Sharma and Patil [10] used a hybrid big data analytical model which integrates Support Vector Regression (SVR) with Auto-Regressive Integrated Mov-ing Average (ARIMA) is proposed to predict product sales and revenues. The simulation results show that the proposed model presents lower relative error rate and higher accuracy that can be utilized for business planning and strategies. They also applied the extreme gradient boosting (XGBoost) [11] based model to forecast sales growth of online products, specifically books and magazines, from massive datasets present in online shopping. PySpark, as the best suitable and compatible framework, is used for data analysis. The result shows that the proposed model has higher forecasting accuracy with a minimum error rate than other

models. A comparative visualization and conclusion are presented in terms of the proposed system's prediction accuracy, error rate, and efficiency.

3 Proposed Methodology

In the context of the AsterixDB project, Hyracks and Algebricks were developed. AsterixDB is a scalable Big-Data Management system built from the ground up to ingest, manage, index, query, and analyze mass quantities semi-structured data [12]. As a implementation, this research describes how Hyracks and Algebricks form the underpinnings of this Big-Data Management system. It is begin by first providing an overview of AsterixDB's data definition capabilities and its data manipulation capabilities. Algebricks and Hyracks come together to form the bottom half of AsterixDB.

3.1 Data Definition

In this section we describe AsterixDB's data definition features. We illustrate them by example through a scenario based on information about users and their messages from a hypothetical social network called Mugshot.com.

3.1.1 Dataverses, Datatypes, and Datasets

The top-level organizing concept in AsterixDB is the *Dataverse*. A Dataverse, short for "data universe", is a place (akin to a database in an RDBMS) within which one can create and manage the types, Datasets, functions, and other artifacts for a given application. Initially, an AsterixDB instance contains no data other than the system catalogs, which live in a system-defined Dataverse (the Metadata Dataverse).

In this chapter, we describe the design and implementation of the Algebricks layer. As we started to build the AQL compiler for the AsterixDB platform, we realized that having a generic compiler framework to compile declarative languages to evaluate using a parallel dataflow platform like Hyracks would be useful to the community at large. Based on prior research in the area of extensible systems and query algebras, we designed the Algebricks library to help query language implementors avoid spending time building a lot of boiler plate code that goes into implementing a full-fledged compiler. Algebricks, along with Hyracks, enables query language implementors to have a complete parallel query compiler in a matter of days.

3.2 The Algebricks Framework

Algebricks serves as an algebraic layer dedicated to optimization with parallel query processing. Its versatility in accommodating different data-intensive query languages is a result of its deliberate design choice to remain agnostic regarding the data model it processes. In a logical sense, operators within Algebricks function on tuples collections that contain data values. Algebricks purposely refrains from specifying the types and formats of the data values encapsulated within a tuple. This allows language implementors the freedom to define any value types as abstract data types.

For instance, when a language developer employs Algebricks to implement a SQL compiler, they can establish SQL's scalar data types as the data model. In this scenario, the developer is responsible for defining SQL expressions' type computation, implementing runtime functions like scalar and aggregate functions, and managing runtime operations such as comparison and hashing. Algebricks accommodates this flexibility, enabling the seamless integration of diverse data models.

A noteworthy example is the integration of AQL [13] with Algebricks, wherein AQL introduces a broader array of data types, including various collection types and nested types. These additional data types have been successfully implemented using the Algebricks API, showcasing the adaptability and extensibility of the Algebricks toolkit.

The Algebricks framework usually contains of the following essential components:

- A set of logical operators,
- A rewrite rule framework,
- A set of general rewrite rules,
- A set of physical operators,
- An API for metadata provisioning that exposes catalog information to Algebricks, and,
- A mapping of physical to runtime operators and connectors in Hyracks.

Compilation Flow. Figure 1 illustrates the standard sequence of compilation steps undertaken by a query processor constructed with Algebricks. The process begins with the lexical analysis and parsing of an incoming query string, leading to the construction of an abstract syntax tree (AST). Subsequently, this AST is transformed into the Algebricks logical plan, comprising logical operators. This logical plan acts as an intermediate representation for the ensuing stages of query compilation.

4 Experimental Evaluation

In this section, we demonstrate experimentally the parallel efficiency of Hivesterix, AsterixDB, and VXQuery. In addition to showing proof of their existence, these experiments aim to expose the direct benefits of using Algebricks in achieving good parallelization characteristics. We report performance for a representative set of queries for each system for different sizes of data and different numbers of nodes, with the goal of showing the scale-up and speed-up characteristics of each system.

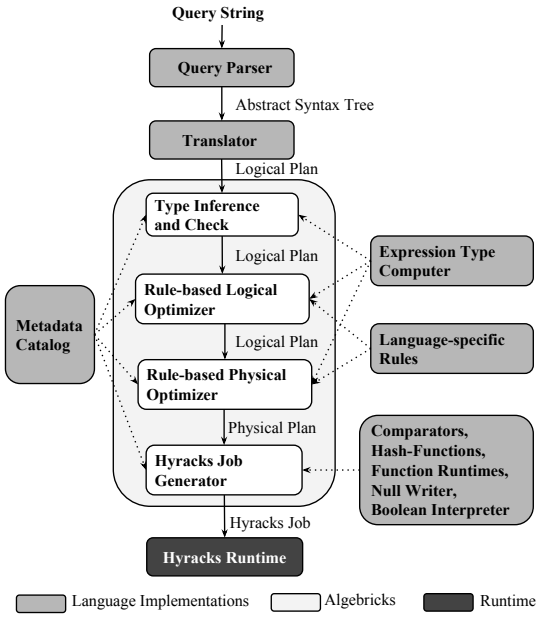

Figure 1: Flowchart of a typical Algebricks-based compiler.

4.1 Hivesterix

The experiments were conducted using following configurations:

- **Cluster**: Hivesterix experiments worked on a 40-node cluster with a Gigabit Ethernet switch. Each node had a Quadcore Intel Xeon CPU E3-1230 V2 3.30GHz, 16GB of RAM, and 3TB RAID0 (3x1TB disks, linux software RAID). We compared Hivesterix and Hive-on-Hadoop (Hive-0.12.0). In the experiments, the RAID0 disk on each node is used for the HDFS data directory and the spilling workspace for query processing.

- **Data**: For speedup experiments, we used TPC-H 250× (250GB). For scaleup experiments, TPC-H 250×, 500×, 750×, 1000× (1TB) were used for 10, 20, 30, and 40 machines respectively.

- **Configuration**: Eight partitions per machine.

- **Queries**: Three representative queries were reported from the TPC-H benchmark, a filter and aggregate query, a group-by query, and a join with group-by query, shown below.

As indicated by Figures 2 and 3, all queries show good speed-up and scale-up characteristics. All three benefit from scanning blocks of HDFS data in parallel. In HQ1 and HQ2, filtering is parallelized and aggregation is done in two phases, reducing the amount of data transferred across machines. HQ3 benefits from parallelizing joins across the cluster. We have also executed TPC-H using Hive-on-Hadoop (Hive-0.12.0) and a comparison with Hivesterix is shown in Figure 4. (An interesting future exercise might include the new generation of "SQL on Hadoop" systems.)

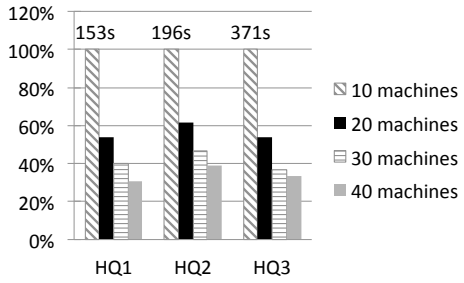

Figure 2: Hivesterix cluster speed up (percentage of 10 machines).

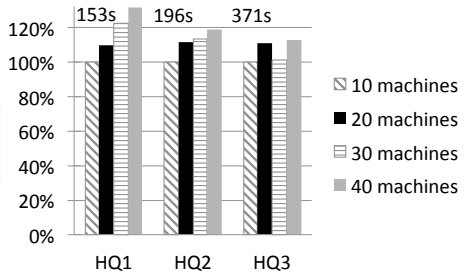

Figure 3: Hivesterix cluster scale up (percentage of 10 machines)

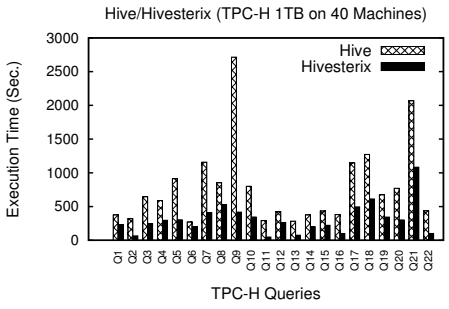

Figure 4: Hivesterix and Hive-on-Hadoop Comparison on TPC-H

4.2 Apache AsterixDB

The Apache AsterixDB were configured as:

- **Cluster**: We ran the reported experiments on a 10-node IBM x3650 cluster with a Gigabit Ethernet switch. Each node had one Intel Xeon processor E5520

2.26GHz with four cores, 12GB of RAM, and four 300GB, 10K RPM hard disks. On each machine 3 disks were used for data. The other disk was used to store 'system's data" (transaction logs and system logs).

- **Data**: We used a synthetic data generator to create records for the collections related to these tests ("GleambookUsers" etc.) For the speedup experiments, we generated 338GB of data, loaded on 9, 18 and 27 partitions. For the scaleup experiments, we generated 169GB, 338GB and 507GB of data for 9, 18 and 27 partitions respectively. A secondary index was constructed on the user_since field of the GleambookUsers dataset.
- **Configuration**: We ran three partitions on each machine. We assigned a maximum of 6GB of memory to each node controller. The buffercache size for each node controller was set to be 1GB.
- **Queries**: We executed four representative AQL queries, as follows.

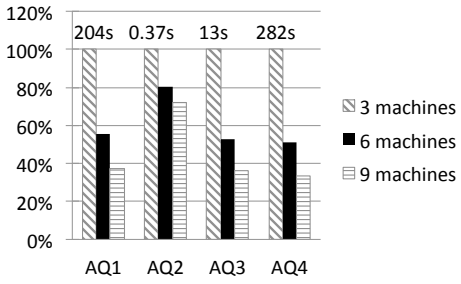

Figure 5: AsterixDB cluster speed up (percentage of 3 machines)

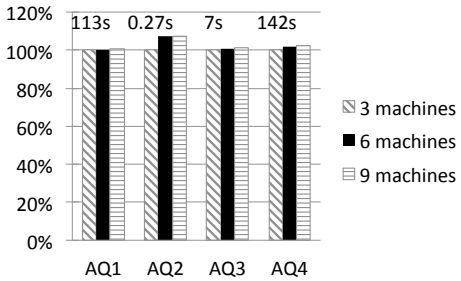

Figure 6: AsterixDB cluster scale up (percentage of 3 machines)

Figure 5 and Figure 6 depict the parallel speedup and scaleup of the four AQL queries. AQ1 and AQ3 benefit from the same rules in Algebricks that helped HQ1 and HQ2 in Hivesterix. Scanning partitions and filtering is done in parallel on the different nodes of the cluster. In AQ3, the aggregation is performed in two phases to reduce network transfer. Join parallelism allows AQ4 to use the cluster effectively. AQ2 uses an index in AsterixDB to evaluate filters instead of scanning all the data. The index range scan is performed in parallel on the different partitions. Note that Algebricks includes facilities for utilizing indexes for query processing, but AsterixDB is currently the only system built on top of Algebricks that implements indexing at the storage level. Detailed AsterixDB performance characteristics can be found in [13].

4.3 Apache VXQuery

Apache VXQuery were configured as:

- **Cluster**: Experiments were run on a cluster whose nodes have two Dual-Core AMD Opteron(tm) processor 2212 HE CPUs, 8GB of RAM, and two 1TB hard drives.
- **Data**: We used NOAA's Global Historical Climatology Network (GHCN)-Daily dataset that includes daily summaries of climate recordings (e.g., high and low temperatures, wind speed, rainfall). The complete XML data definition can be found on NOAA's site [14]. For the speed-up experiments, we used 57GB of weather XML data partitioned over the number of machines (varied from 1 to 8) for each run. The scale-up experiments were performed while keeping the amount of data per machine constant at 7.2GB and varying the number of machines from 1 to 8.
- **Configuration**: We ran four partitions per machine.
- **Queries**: We used the following three XQuery queries:

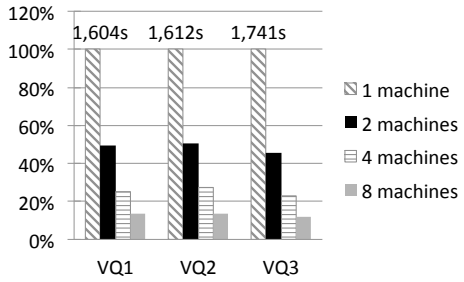

Figure 7: VXQuery cluster speed up (percentage of 1 machine)

Figure 7 and Figure 8 show the parallel speedup and scaleup of Apache VXQuery. The optimizations implemented in Algebricks help VQ1, VQ2, and VQ3 to achieve good parallel performance by parallelizing scans, filters, aggregations, and joins. More performance results for the

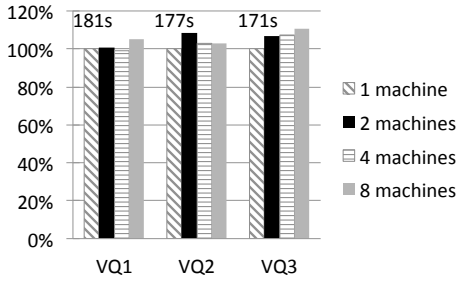

Figure 8: VXQuery cluster scale up (percentage of 1 machine)

current Apache VXQuery implementation on top of Algebricks can be found in [15].

Table 1: Code metrics as a proxy for the effort required to build query compilers using Algebricks

	Algebricks	AsterixDB	Hivesterix	VXQuery
LOC	42K	40K	3.4K	12.8K
# Rules	50	61	4	29
# Logical Ops	39	2	0	0
# Physical Ops	44	6	0	0

Algebricks has served as a useful tool to build not just the AsterixDB query compiler, but also query compilers for HiveQL and XQuery. Table 1 shows the lines of code, the number of rewrite rules, the number of logical operators, and the number of physical operators that were provided by Algebricks, and those that needed to be additionally implemented in AsterixDB, Hivesterix, and VXQuery. Algebricks provides about 50 rewrite rules, 39 logical operators, and 44 physical operators that are helpful for all three query processors. Additionally, AsterixDB required only 2 logical operators and 6 physical operators. These operators were AsterixDB specific and related to accessing indexes. The extensibility offered by Algebricks allowed the AsterixDB platform to implement these operators and plug them into the Algebricks framework. 61 rewrite rules were implemented in the AsterixDB platform to deal with rewrites specific to the Asterix data model. Hivesterix and VXQuery did not require any additional operators. Operators available in Algebricks were sufficient to implement the entire HiveQL and XQuery language compilers. Hivesterix and VXQuery required 4 and 29 rewrite rules, respectively, that were specific to the datamodel semantics of the two systems.

5 Conclusion and Future Work

This research presented Hyracks, a high-performance parallel dataflow runtime layer, and Algebricks, a parallel query compilation layer, both of which underpin the AsterixDB stack. Hyracks in conjunction with Algebricks also support quite a few other query processing systems. In addition to AsterixDB, Hivesterix and Apache VXQuery are two other query processing engines that have been built on top of Hyracks + Algebricks. Hyracks has also served as a research platform for building specialized Big Data processing systems. Hyracks has also been used to implement a highly scalable version of the Batch Gradient Descent Algorithm for machine learning.

In the future, Hyracks under AsterixDB stack could benefit from an automatic resource management architecture, so that jobs could be optimized using various policies without the over (or under) consumption of available resources.

References

[1] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan, "Interpreting the Data: Parallel Analysis with Sawzall," *Scientific Programming*, vol. 13, no. 4, pp. 277–298, 2005.

[2] L. Lin, V. Lychagina, and M. Wong, "Tenzing A SQL Implementation On The MapReduce Framework," *Proceedings of the VLDB Endowment*, vol. 4, no. 12, pp. 1318–1327, 2011. [Online]. Available: http://research.google.com/pubs/pub37200.html

[3] "The Hive Project," http://hive.apache.org/.

[4] K. Peng, V. C. M. Leung, and Q. Huang, "Clustering approach based on mini batch kmeans for intrusion detection system over big data," *IEEE Access*, vol. 6, pp. 11 897–11 906, 2018. [Online]. Available: https://doi.org/0.1109/ACCESS.2018.2810267

[5] S. Ma, Y. Huang, Y. Liu, H. Liu, Y. Chen, J. Wang, and J. Xu, "Big data-driven correlation analysis based on clustering for energy-intensive manufacturing industries," *Applied Energy*, vol. 349, p. 121608, 2023. [Online]. Available: https://doi.org/10.1016/j.apenergy.2023.121608

[6] L. Li, "Efficient distributed database clustering algorithm for big data processing," in *6th International Conference on Smart Grid and Electrical Automation (ICSGEA)*, 2021, pp. 495–498. [Online]. Available: https://doi.org/10.1109/ICSGEA53208.2021.00118

[7] D. Paulraj, K. Mohamed Junaid, T. Sethukarasi, M. Vigilson Prem, S. Neelakandan, A. Alhudhaif, and N. Alnaim, "A novel efficient rank-revealing qr matrix and schur decomposition method for big data mining and clustering (rrqr-sdm)," *Information Sciences*, vol. 657, p. 119957, 2024. [Online]. Available: https://doi.org/10.1016/j.ins.2023.119957

[8] F. Liu, Y. He, J. He, X. Gao, and F. Huang, "Optimization of big data parallel scheduling based on dynamic clustering scheduling algorithm," *Journal of Signal Processing Systems*, vol. 94, no. 11, pp. 1243–1251, Nov 2022. [Online]. Available: https://doi.org/10.1007/s11265-022-01765-4

[9] P. Tian, H. Shen, and A. Abolfathi, "Towards efficient ensemble hierarchical clustering with mapreduce-based clusters clustering technique and the innovative similarity criterion," *Journal of Grid Computing*, vol. 20, no. 4, p. 34, Sep 2022. [Online]. Available: https://doi.org/10.1007/s10723-022-09623-0

[10] G. Sharma and S. Patil, "Big data analysis for revenue and sales prediction using support vector regression with auto-regressive integrated moving average," *SAMRIDDHI: A Journal of Physical Sciences, Engineering and Technology*, vol. 15, no. 1, pp. 1–8, 2023. [Online]. Available: https://doi.org/10.18090/samriddhi.v15i01.01

[11] G. Sharma and S. Patil, "Extreme gradient boosting model-based forecasting of big data online sales record," *SAMRIDDHI: A Journal of Physical Sciences, Engineering and Technology*, vol. 14, no. 1, pp. 112–119, 2023. [Online]. Available: https://doi.org/10.18090/samriddhi.v14i01.18

[12] A. Behm, V. Borkar, M. Carey, R. Grover, C. Li, N. Onose, R. Vernica, A. Deutsch, Y. Papakonstantinou, and V. Tsotras, "ASTERIX: Towards a Scalable, Semistructured Data Platform for Evolving-world Models," *Distributed and Parallel Databases*, vol. 29, no. 3, pp. 185–216, 2011.

[13] S. Alsubaiee, Y. Altowim, H. Altwaijry, A. Behm, V. R. Borkar, Y. Bu, M. J. Carey, I. Cetindil, M. Cheelangi, K. Faraaz, E. Gabrielova, R. Grover, Z. Heilbron, Y. Kim, C. Li, G. Li, J. M. Ok, N. Onose, P. Pirzadeh, V. J. Tsotras, R. Vernica, J. Wen, and T. Westmann, "AsterixDB: A Scalable, Open Source BDMS," *PVLDB*, vol. 7, no. 14, pp. 1905–1916, 2014.

[14] "National Climate Data Center: Data Access," http://www.ncdc.noaa.gov/data-access/.

[15] E. P. Carman, Jr., T. Westmann, V. R. Borkar, M. J. Carey, and V. J. Tsotras, "Apache VXQuery: A Scalable XQuery Implementation," *CoRR*, vol. abs/1504.00331, 2015.