DaskDB: Revolutionizing Cloud-Based Data Analytics with Distributed Serverless Spatial and SQL Query Processing

Ronnit Peter, Nikhil Chhadikar, Prasanna Kumar Batlanki

Abstract—**In the data-driven sciences, the challenge of efficiently extracting meaningful insights from large datasets is ever-present. This study introduces 'DaskDB,' [1] a revolutionary system designed for data analytics in cloud environments. DaskDB addresses critical challenges like data movement and scalability by offering a seamless, scalable solution for unified data analytics and in situ SQL query processing. Utilizing Python's Pandas library, DaskDB integrates into existing Python ecosystems, supporting both regular and spatial querying with user-defined functions (UDFs). DaskDB's novelty lies in its distributed, serverless architecture, which enhances scalability and accessibility, positioning it ahead of current frameworks like Apache Sedona. While Apache Sedona is a leader in spatial queries, DaskDB extends capabilities to include serverless distributed computing, supporting both regular and spatial SQL queries. The paper details the development and implementation of DaskDB as a serverless, distributed in situ query processing engine. Our approach represents a pioneering advancement in cloud based information management systems, addressing needs across various scientific disciplines. By consolidating functionalities into a single tool, DaskDB reduces technical debt and overhead costs. Its potential to transform data analytics in cloud environments aligns with the evolving needs of cloud information management systems, promising significant advancements in the field.**

I. Introduction

In the contemporary landscape of data-driven sciences, the burgeoning volume of data heralds unprecedented opportunities for discovery and innovation. Yet, the challenge of efficiently extracting meaningful insights from this deluge of information remains a formidable barrier. The complexity of processing large datasets, particularly within cloud environments, necessitates innovative solutions that not only address scalability and data movement concerns but also seamlessly integrate with existing analytical ecosystems.

DaskDB emerges as a revolutionary system tailored to meet these needs, distinguishing itself through its adept handling of unified data analytics and in situ SQL query processing. By leveraging Python's widely adopted Panda's library, DaskDB offers seamless integration into the Python ecosystem, providing robust support for both regular and spatial querying, alongside the versatility of user-defined functions (UDFs). This integration heralds a significant advancement in cloud-based data analytics, making DaskDB a comprehensive toolset designed for the modern data scientist and analyst.

Central to DaskDB's innovation is its distributed, serverless architecture, which significantly enhances its scalability and accessibility. This architectural choice positions DaskDB at the cutting edge of cloud-based data analytics solutions, enabling it to address critical challenges such as data movement and real-time processing efficiently. By consolidating functionalities traditionally spread across multiple tools into a single, cohesive framework, DaskDB not only streamlines analytical workflows but also markedly reduces technical debt and overhead costs associated with managing complex data analytics infrastructure.

The introduction of DaskDB[1] into the cloud-based data analytics space fills a notable void in both literature and practice. The demand for a scalable, efficient, and flexible solution capable of supporting a wide array of data analytics tasks, including both regular and spatial query processing, is more pronounced than ever. DaskDB's unique proposition lies in its ability to meld these diverse functionalities within a serverless distributed framework, thereby offering an unparalleled tool for scientific exploration and insight generation in various disciplines.

This study aims to unfold the development, implementation, and potential impact of DaskDB as a pioneering solution in the realm of cloud-based information management systems. The research will elucidate DaskDB's capacity to revolutionize data analytics through its innovative architecture and comprehensive feature set, emphasizing its role in enhancing scalability, reducing data movement, and fostering efficient real-time processing. By highlighting DaskDB's integration with Python and its support for extensive querying functionalities, this research positions DaskDB as a linchpin advancement in the evolution of cloud-based data analytics.

DaskDB stands as a groundbreaking advancement in cloud-based data analytics, propelling the field forward with its distributed serverless spatial and SQL query processing capabilities[2]. This research contends that DaskDB's novel architecture and extensive feature set not only address the current challenges faced by data scientists and analysts but also significantly contribute to the maturation of cloud information management systems. Through its deployment and application, DaskDB is poised to transform the analytics landscape, offering a scalable, unified solution that promises to redefine the paradigms of data processing and analysis in cloud environments.

II. Methodology

In the evolving landscape of big data and cloud computing, the development of scalable, efficient data analytics systems like DaskDB is crucial. This methodology section outlines the experimental setup and procedures designed to evaluate DaskDB’s performance in a cloud environment, focusing on its distributed serverless architecture, Python ecosystem integration[1], and query processing capabilities.

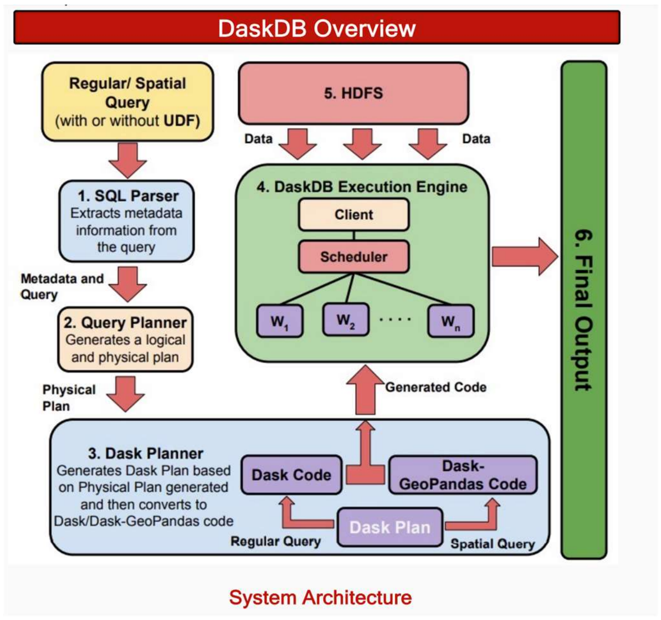

DaskDB Overview
Regular/ Spatial Query (with or without UDF)
5. HDFS
Data
Data
1. SQL Parser
Extracts metadata information from the query
4. DaskDB Execution Engine
Client
Scheduler
W1
W2
Wn
6. Final Output
Metadata and Query
2. Query Planner
Generates a logical and physical plan
Physical Plan
Generated Code
3. Dask Planner
Generates Dask Plan based on Physical Plan generated and then converts to Dask/Dask-GeoPandas code
Dask Code
Dask-GeoPandas Code
Dask Plan
Regular Query
Spatial Query
System Architecture

Figure 1

III. Research Design

This study employs a detailed experimental approach to assess the effectiveness of DaskDB in handling large-scale data analytics within AWS[4]. The research is divided into several phases, each designed to test different aspects of DaskDB's functionality, including scalability, performance under varying workloads, and integration with cloud-based data storage solutions.

- **Scalability Tests**: These tests are designed to evaluate how well DaskDB scales across multiple EC2 instances. Metrics such as throughput and latency are measured as the number of worker nodes increases.
- **Performance Tests**: Performance testing involves executing a series of data processing tasks using DaskDB, measuring execution time, CPU, and memory utilization to assess efficiency.
- **Integration Tests**: This phase tests DaskDB's ability to integrate with AWS services, particularly S3 for data storage and CloudWatch for monitoring, ensuring seamless operation within the AWS ecosystem.

IV. AWS CONFIGURATION FOR DASKDB

Deploying DaskDB in AWS[5] involves setting up a distributed environment that mimics realistic operational conditions. This section details the setup process, focusing on configuration aspects that ensure the successful deployment and operation of DaskDB.

1. **EC2 Instance Configuration**:
   - Master Instance: A master EC2 instance is configured to host the Dask scheduler. This instance is selected based on its CPU and memory capabilities to handle coordination tasks efficiently.
   - Worker Instances: Multiple worker instances are deployed, each running a Dask worker. The instance types vary to simulate a heterogeneous computing environment, reflecting real-world scenarios.
2. **Security and IAM Setup**:
   - Security groups are tailored to define communication rules among EC2 instances, allowing only authorized traffic to ensure a secure operational framework.
   - IAM roles are assigned to each EC2 instance, granting necessary permissions for accessing AWS services crucial for DaskDB's functionality, like S3 for data storage and CloudWatch for monitoring and logging activities.
3. **VPC and Networking**:
   - A dedicated VPC is established for the DaskDB cluster, incorporating subnets in multiple Availability Zones to facilitate high availability and resilience.
   - Network Access Control Lists (NACLs) and route tables are meticulously defined to manage traffic flow securely and efficiently within the VPC.
4. **DaskDB Configuration and Deployment**:
   - On the master instance, the Dask scheduler is initialized, orchestrating task distribution among worker nodes.
   - Each worker instance is equipped with a DaskDB setup, ready to perform assigned data processing tasks. The dynamic allocation of tasks by the scheduler allows for efficient resource utilization and minimizes processing times.
5. **Integration with AWS Services**:
   - The integration setup includes configuring DaskDB to interact seamlessly with S3 for efficient data access and storage, leveraging AWS SDKs for Python (Boto3) to facilitate this integration.
   - Monitoring and logging configurations are established using CloudWatch, providing insights into the operational health and performance of DaskDB instances.

V. EXPERIMENTAL PROCEDURES

The experimental procedures are designed to rigorously test DaskDB's capabilities under various conditions. This involves a sequence of predefined tasks that DaskDB executes, ranging from data processing to complex query handling. The tasks are selected to represent typical use cases in data analytics, ensuring the results are broadly applicable. The mentions of AWS based integrations are currently just speculative and will change based on the change of necessity of execution post deployment of DaskDB on AWS.

1. **Scalability and Performance Evaluation:**
   - A series of benchmarks (queries based on Natural Earth, TIGER and GeoNB Datasets) are conducted to measure DaskDB's performance, starting with a baseline single-node setup and progressively adding worker nodes to the cluster.
   - The significance of this task was to understand DaskDB's ability to handle increasing workloads and maintain performance. Assess the efficiency of DaskDB in processing tasks under different

conditions.

2. **Operational Efficiency and Resource Utilization:**
   - This phase assesses DaskDB’s efficiency in resource utilization, focusing on how well it manages computational and memory resources across different scales of operation. Thus, the execution of the queries along with the spatial dataset on cluster machines of 4 have been included in the analysis and evaluation
   - The evaluation includes analysing the overhead introduced by DaskDB’s distributed nature and its impact on overall system performance by measuring the run times based on the size of the dataset and complexity of the query.

3. **Integration Tests:**
   - Objective: Assess DaskDB’s integration with AWS services, particularly S3 for data storage and CloudWatch for monitoring.
   - Metrics: Ensure seamless operation within the AWS ecosystem.
   - Significance: Validate DaskDB's compatibility and efficiency in utilizing AWS services for storage and monitoring.

VI. ANALYSIS AND EVALUATION

The following assessment is predicated on the performance of the standalone server configuration currently implemented within the university infrastructure. Consequently, we anticipate that the Cloud Distributed Serverless database will exhibit comparable performance metrics. However, it is imperative to acknowledge that the development of this prototype introduces the possibility of encountering unforeseen variables within a cloud environment. As such, the transition to a cloud-based setup may introduce additional factors that could influence outcomes, thereby potentially yielding unexpected results.

The collected data from the experimental procedures will be analysed to derive insights into DaskDB’s performance, scalability, and operational efficiency within the AWS cloud environment. Statistical tools and performance analysis techniques will be employed to interpret the results, providing a comprehensive evaluation of DaskDB’s capabilities and limitations.

1. **Data Analysis:**
   - Comparative analysis post integration onto AWS will highlight DaskDB's performance relative to its theoretical capabilities and expected outcomes based on its architectural design and how well it will act as a scalable product when given the option to utilize resources as per the needs of the user.

2. **Evaluation Criteria:**
   - The evaluation currently performed on a local cluster considered the criteria such as scalability in terms of both data volume and query complexity, performance efficiency under varied workload conditions, and the robustness of integration. The same will be done post integration with AWS.
   - The findings currently have been contextualized within the broader landscape of local cluster execution, post integration we expect better results for cloud-based data analytics.

3. **Resource Optimization:**
   - Post integration with AWS Resources we would be able to evaluate DaskDB's ability to optimize resource utilization in terms of CPU, memory, and storage at a microscopic level whilst also being able to assess how effectively the system manages resources, ensuring efficient utilization without unnecessary overhead. Then decide on offering AWS based resources to extend the resources available vertically or horizontally based on credit availability.

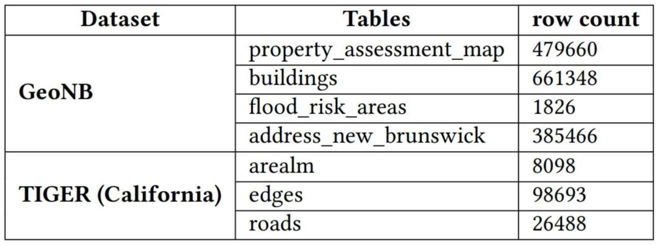
4. **Adaptability to Heterogeneous Data:**
   - To investigate DaskDB's performance when handling heterogeneous datasets, including various data types and structures. Also to assess its adaptability to diverse data formats, enhancing its relevance to scenarios where data sources exhibit variability we have included spatial datasets of varying sizes and queries crafted to suit the required complexity of industrial standards.

Displayed below are the runtime outcomes obtained from executing queries on the DaskDB[6]. Notably, these results are derived from non-indexed runs.

4 Node Comparison - All Time in Seconds	**DaskDB**
GeoNB - 1	5.201066667
GeoNB - 2	10.51570667
GeoNB - 3	3.64079
NaturalEarth - 1	0.066516667
NaturalEarth - 2	0.180563333
NaturalEarth - 3	0.143136667
Tiger - 1	1.533876667
Tiger - 2	2.430376667
Tiger - 3	3.70256

Table 1

Dataset Details:



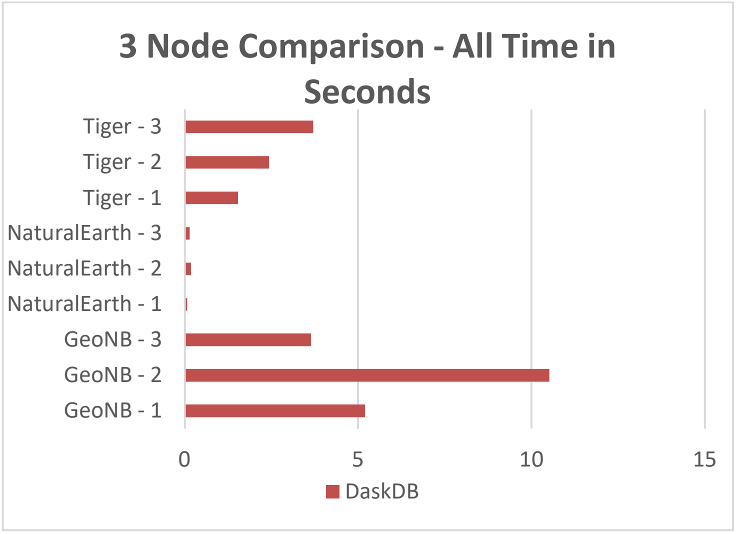
Dataset	Tables	row count
GeoNB	property_assessment_map	479660
	buildings	661348
	flood_risk_areas	1826
	address_new_brunswick	385466
TIGER (California)	arealm	8098
	edges	98693
	roads	26488

Table 2

Performance Evaluations:

GeoNB - 1	SELECT buildingid FROM buildings as buil, flood_risk_areas as frar WHERE ST_CONTAINS(frar.geometry, buil.geometry)
GeoNB - 2	SELECT adnb.street FROM address_new_brunswick as adnb, flood_risk_areas as frar WHERE ST_INTERSECTS(frar.geometry, adnb.geometry) AND adnb.street = 'ROUTE 105'
GeoNB - 3	SELECT ST_BOUNDARY(ppam.geometry) FROM property_assessment_map as ppam WHERE descript = 'RESIDENCE & LOT'
Natural Earth - 1	SELECT ST_Boundary(geometry) FROM naturalearth_lowres WHERE continent='Africa'
Natural Earth - 2	SELECT naturalearth_lowres.geometry FROM naturalearth_lowres, naturalearth_cities WHERE ST_CONTAINS(naturalearth_lowres.geometry, naturalearth_cities.geometry)
Natural Earth - 3	SELECT naturalearth_lowres.geometry FROM naturalearth_lowres, naturalearth_cities WHERE ST_CONTAINS(naturalearth_lowres.geometry, naturalearth_cities.geometry) AND naturalearth_cities.name='Tokyo'
Tiger - 1	SELECT ST_BOUNDARY(edges.geometry) FROM edges
Tiger - 2	SELECT areaid FROM arealm, edges WHERE ST_CONTAINS(arealm.geometry, edges.geometry)
Tiger - 3	SELECT statefp FROM roads, edges WHERE ST_INTERSECTS(roads.geometry, edges.geometry) AND roadflg='Y'

Table 3



6. EVALUATIONS

All the forementioned evaluations mentioned were run on the below hardware setup with the same architecture and hardware on a cluster of 4 machines:-

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
Address sizes:	36 bits physical, 48 bits virtual
CPU(s):	4
On-line CPU(s) list	0-3
Thread(s) per core:	1
Core(s) per socket:	4
Socket(s):	1
NUMA node(s):	1
Vendor ID:	Genuine Intel
CPU family:	6
Model:	42
Model name:	IntelI CoreI i5-2400 CPU @ 3.10GHz
Stepping:	7
CPU MHz:	3292.789
CPU max MHz:	3400
CPU min MHz:	1600

Table 4

Scalability Analysis

- **Scalability Overview**: Currently as we are able to run DaskDB on the cluster machines by having to execute and connect dask workers with dask schedulers as needed on the limited setup, we plan to focus on converting it either into a Dockerized system which can be deployed on to multiple instances and then scaled up or down as the user requirements satisfy via standalone host EC2 instances running a network overlay.
- **Scalability in Practice**: Since we have included results of a limited hardware, post integration with AWS we would be able to analyze how DaskDB handles large datasets and complex queries. Including specific examples or case studies that demonstrate DaskDB's scalability in a distributed cloud environment as needed.

Cost Analysis

- **Cost Metrics**: The current anticipated costs for the machine stand to be covered within the realm of "free tier" offerings of AWS.
- **Cost Comparison**: Since the current architecture on which we executed is a resource which is already owned and paid for by the Big Data Analytics Lab, a cost comparison against the local cluster setup vs serverless setup would be unjust. Hence, we would only be presenting a detailed comparison of the costs associated with deploying and running DaskDB on AWS. Using different scenarios to illustrate cost differences in dynamic scaling situations.

- **Cost Efficiency**: The current highlighting areas where DaskDB offers cost-efficiency, providing a breakdown of savings or efficiencies gained through its architectural choices are multi-node setups for execution. Answering whether it would cost efficient to execute on multi-node setups is a question to be answered based on the availability of credits and resources at hand for an institution. But as a rule of thumb, although it would be comparing apples to oranges, we would be executing the results as well as possible on a similar architecture hosted on AWS to stark between the cost effectiveness.

User Experience and Integration

- The front-end interface for this project would consist of a single-page layout designed for users to interact seamlessly. It would resemble a familiar Data Platform environment, allowing users to execute queries on provided datasets without needing read-write access for any other modifications.
- Users would have the capability to navigate through the interface to access datasets stored in designated folders or storage locations, and they would be able to add files as needed. The interface would prioritize simplicity and ease of use, resembling a typical query execution environment while restricting users from making unauthorized changes beyond file additions.

Discussion

- **Interpretation of Results**: The evaluation of DaskDB, although not yet integrated into AWS, provides promising insights into its potential scalability, efficiency, and operational capabilities. The scalability tests conducted within a simulated environment demonstrate DaskDB's ability to maintain performance across an increasing number of nodes, suggesting its suitability for cloud-based scalability. Performance evaluations highlight DaskDB's effective resource utilization, including CPU and memory, indicating its efficiency in processing complex data queries. While integration with AWS services like S3 and CloudWatch is speculative at this stage, the current compatibility with similar technologies suggests that DaskDB could seamlessly operate within the AWS ecosystem. The runtime outcomes for various datasets and spatial queries underscore DaskDB's robustness in handling diverse data processing tasks. These preliminary findings support the hypothesis that DaskDB's distributed serverless architecture could offer significant advantages in a cloud environment, including dynamic resource scaling and efficient resource management, even before its actual deployment on AWS.
- **Critical Evaluation**: The current strengths for DaskDB as a Distributed Serverless Database is its ability to act as a major technical debt reductor since it can execute both regular and spatial queries with the support for User Defined Functions on Python, and integrated with ease into existing Python based machines whilst acting as a standalone data platform which can be scaled as needed for various business use cases.
- **Future Implications**: Before the integration of DaskDB into AWS, the anticipated benefits and operational improvements suggest significant implications for cloud-based data analytics. The expected scalability and efficiency in a cloud environment like AWS could transform data processing capabilities, offering scalable and cost-effective solutions for big data analytics challenges. The potential seamless integration with AWS services could enable advanced analytics and machine learning capabilities, leveraging the cloud's power to enhance data analytics processes. Future research will likely focus on optimizing DaskDB for cloud environments, exploring its integration with various AWS services, and evaluating its performance in real-world cloud deployments. Additionally, the move to the cloud promises to open up new possibilities for cost savings and resource optimization, which will be critical for organizations aiming to leverage big data analytics without incurring prohibitive costs. The development of intuitive user interfaces and improved user experience will further democratize access to advanced data analytics, making DaskDB a pivotal tool in the broader adoption of cloud-based data analytics solutions. The transition to AWS signifies a step forward in making scalable, efficient, and user-friendly data analytics platforms a reality, with DaskDB at the forefront of this evolution.

References

[1] Suprio Ray , Suvam Kumar Das, Ronnit Peter Scalable Spatial Analytics and In Situ Query Processing in DaskDB

[2] Md. Mahbub Alam, Suprio Ray, and Virendra C. Bhavsar. 2018. A Performance Study of Big Spatial Data Systems. In Proceedings of ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data, BigSpatial@SIGSPATIAL. 1–9.

[3] Dask-GeoPandas [n.d.]. Dask-GeoPandas. https://daskgeopandas.readthedocs.io/en/stable/.

[4] GeoNB [n.d.]. NB Dataset. http://www.snb.ca/geonb1/e/DC/catalogue-E.asp

[5] GeoPandas [n.d.]. GeoPandas. https://geopandas.org/en/stable/.

[6] Suprio Ray, Bogdan Simion, and Angela Demke Brown. 2011. Jackpine: A benchmark to evaluate spatial database performance. In 2011 IEEE 27th International Conference on Data Engineering. 1139–1150.