Cloud Native NoSQL Database vs SQL Databases

Ashik Adnan , S M Mahsanul Islam , Fahim Hasnat ,
Nazifa Khanom , Md Humaion Kabir Mehedi , Shadab Iqbal , and Annajiat Alim Rasel

Department of Computer Science and Engineering
Brac University
66 Mohakhali, Dhaka - 1212, Bangladesh
{*ashik.adnan, sm.mahsanul.islam, md.fahim.hasnat, nazifa.khanom, humaion.kabir.mehedi, shadab.iqbal*}*@g.bracu.ac.bd*
annajiat@gmail.com

***Abstract*—Software development companies faces a plethora regarding cloud migration possibilities because the cloud computing technologies advance for better accommodation of hosted software applications. Data management is one of the main issues. While there isn't yet a concept for cloud-native databases, research on cloud-native apps has inspired the development of extremely elastically scalable and robust stateless systems. Additionally, SQL (short for Structured Query Language) has become the de facto language for expressing database requests during the past few decades. SQL makes it simple to explain simple and mid-level data searches, while at the same time gives experienced SQL developers the flexibility and capacity to write complicated and very expressive queries. Even non-technical people could easily understand and write SQL when it was first created. In this research, we attempted to compare Cloud Native Databases to SQL Databases**

***Index Terms*—NoSQL, SQL, Cloud Native, Spanner, MongoDB, DynamoDB, Aurora, PostgreSQL, BigQuery, BigTable**

I. Introduction

No-SQL databases are high-performance and non-relational data repositories. Column stores, key value stores, document stores, and graph databases are the four basic categories of NoSQL databases. There is currently no native Graph NoSQL database option available on Google Cloud. It does, however, support Janusgraph and Neo4j, two popular Kubernetes-hosted open source graph database solutions. The storage backend for the Kubernetes implementation of Janusgraph can be Bigtable. To satisfy these needs, NoSQL databases have grown in popularity. They use key-value databases or JSON-based databases, to name a couple of the main storage types, to store their data differently than relational databases do. PostgreSQL, which uses an OORDMS-based architecture and supports JSON, illustrates how these NoSQL databases can persist.

The backend enterprise's tried-and-true workhorse for years, SQL databases are at the core of all we do in this digital age. Early on in the 1970s, IBM developed SQL as a way to access their System R database system. PostgreSQL, MariaDB, and MySQL are just a few examples of SQL databases. Because of how deeply ingrained SQL databases are in our daily lives, these highly functioning and durable RDBMS will continue to be a mainstay of the business for many years to come. In this paper, we compared SQL databases to Cloud Native NoSQL databases.

II. Literature Review

Goldschmidt, T. et al. [1] propose a cloud-native monitoring system architecture. First, they benchmarked three open source time-series databases (OpenTSDB, KairosDB, and Databus) on cloud infrastructures with up to 36 nodes and industrial workloads. KairosDB meets our original scalability and reliability hypotheses. They published a benchmark to evaluate its time-series database scalability and robustness using realistic industrial workloads. Maulidin, A. J. et al. [2] measure the synchronization system execution time and test data integration on web services. Data integration for 2 web services passed. The synchronization system's execution time is then measured. This work tries to automatically integrate two separate datasets online. They presented a Web services-based method for aggregating databases. Ghule et al. [3] suggested a Data Adapter system to promote a hybrid Relational/NoSQL database design. It can synchronize database transformations. It enables Data Analytics on Scaled-out Object Storage Systems. On an Object-based Storage Cluster, a Data Analytics Layer performs in-place Map-Reduce on existing data. It retains RDBMS's ACID features while giving NoSQL's benefits through an intermediate layer that tracks running transactions and manages concurrent transactions. Hsu, TC. et al. [4] employed MongoDB and PosgresSQL as test databases, and the study will use a local PostgresSQL database as a control group. They investigated cloud computing and the credibility of database information. In the test case, they wrote, read, and queried data. The results justify a new form of cloud salesforce. Heroku subordinate mode helps understand cloud architecture, design structure in multiple databases, and data use analysis reliability. According to LI Chaokui et al. [5], due to performance limitations, relational databases cannot satisfy high concurrency read/write and large scale spatial data organization and management. Through debris type partition technique for picture division, they proposed a non-relational database of remote sensing image data storage based on Mongo DB, with high concurrent access and no drop in read/write speed. Compared to the remote sensing image storing approach based on SQL Server database, Mongo DB's multiple concurrent access efficiency is considerably superior.

According to Vial, G. et al. [6], DBMSs offer relational schemes for transactional applications, non-relational systems, and streaming-oriented technologies for quick analytics. Users can choose DBMS technologies to meet their online, social, mobile, analytics, embedded, real-time, and cloud application needs. In this issue of Software Technology, Gregory Vial presents an overview of DBMS technologies and providers and a case study of an Internet application. SQL Injection is a risky hacking tactic, according to Gupta et al. [7]. In this paper, they proposed a novel SQL Injection Attack mitigation method. They used an integrated encryption mechanism to prevent SQL Injection Attack on web application databases. A database invader can inflict catastrophic damage and steal data. To mitigate these attacks, a hybrid strategy is proposed, using Advanced Encryption Standard (AES) upon login to prevent unauthorized access to databases and Elliptical Curve Cryptography (ECC) to encode the database so that without the key no one can access the database information. Sharma, M. et al. [8] compared RDBMS, MongoDB, and graph-based No SQL databases (Neo4j). Neo4j took longer than MongoDB and PostGre SQL, causing unexpected results. Further examination revealed that the cypher Query caused it. In real life, a user will access the Graph DB through an application, and Neo4j's response time will alter. They directly employed the query access platform, and these findings recommend MongoDB for huge records.

III. Databases

A. What is Spanner?

Google's Spanner is a system for managing and storing SQL databases across several servers. In terms of functionality, [9] it supports things like cross-regional transactions, multi-site replication and failover, and strongly consistent reads. Google Ads' database, Google F1, makes use of Spanner. In 2012, Google's internal data centers were the original focus of the Spanner description [10]. A SIGMOD 2017 paper details the addition of SQL support for Spanner in 2017. In 2017, it was released as "Cloud Spanner" [11] as part of Google Cloud Platform. Massive volumes of structured data that can be changed often are kept in Spanner. With synchronous replication, Spanner ensures robust consistency and high availability for relational data while allowing users to run arbitrary SQL queries on it. Spanner's main characteristics are: Within a Spanner universe, transactions can span not only cells but also rows, columns, tables, and databases. Automatic multi-site replication and failover puts the client in charge of data replication and placement. All copies of an object are created at the same time and are very consistent with one another. Data is versioned to permit stale reads, so clients can access older versions of data within the constraints of garbage collection windows, and reads are highly consistent. Provides an internal SQL interface for reading and writing data.

B. What is MongoDB?

MongoDB is a cross-platform, open-source DBMS. MongoDB employs JSON-like documents with optional schemas. MongoDB is licensed under the Server Side Public License (SSPL), which is non-free. MongoDB supports field, range, and regex queries. Query results can include user-defined JavaScript functions. Queries can return a random sample of a particular size. Primary and secondary indices can index MongoDB document fields. MongoDB's replica sets increase availability [12], and a replica set contains duplicate data. MongoDB can run on numerous servers, balancing load or duplicating data in case of hardware failure. GridFS, a MongoDB file system, offers load balancing and data replication across several machines. MongoDB's drivers include grid file system. MongoDB gives developers file and content functions. Mongofiles or Nginx and lighttpd plugins can access GridFS. GridFS maintains each file chunk as an independent document.

C. What is DynamoDB?

Amazon DynamoDB is a document and key-value store that guarantees performance in the millisecond range, regardless of database size. It is a fully-managed, multi-region, multi-master, persistent database with in-built security, backup and restore features, and in-memory caching designed for internet-scale applications. DynamoDB can handle over 10 trillion requests daily and has peak rates of over 20 million requests per second [13]. Amazon DynamoDB was born out of a need for scalable, reliable cloud computing services and a focus on building large-scale, non-relational databases for Amazon.com. Managed, scalable, fast, durable and highly available, flexible, and low-cost services are just some of the benefits of using Amazon DynamoDB's NoSQL database service. Amazon DynamoDB is great for quickly prototyping and deploying a key-value store database that can scale to gigabytes or terabytes of data. DynamoDB's scalability and high availability make it suited for "always on" use cases with big transactional queries (reads and writes). DynamoDB's high operational expenses make it unsuitable for petabyte-sized data sets and frequent transactions. DynamoDB, a NoSQL database with a JSON-based query API, should be utilized when data models don't require normalized data with cross-table joins.

D. What is AWS Aurora?

Amazon Aurora is a relational database engine that is fully managed and is compatible with MySQL and PostgreSQL (Aurora). You are aware that MySQL and PostgreSQL are two examples of open-source databases that also share the speed and dependability of commercial databases without the hefty price tag. Aurora may be used with the same applications and tools as your current MySQL and PostgreSQL databases. Aurora can deliver up to five times the throughput of MySQL and up to three times the throughput [14] of PostgreSQL for certain workloads, without requiring changes to the bulk of your existing applications. Aurora includes a high-speed storage subsystem. It is compatible with the MySQL and PostgreSQL database management systems, and its fast distributed storage is optimized for both systems. The storage underneath it grows automatically when more space is required. There is a hard limit of 128 terabytes on the size of data stored in an Aurora cluster (TiB). Aurora also automates and standardizes database clustering and replication, which are typically among the most challenging aspects of database implementation and maintenance.

E. What is PostgreSQL?

PostgreSQL is an open-source RDBMS that prioritizes extensibility and SQL compliance. Since it was designed to replace the Ingres database, also created at Berkeley, its original name, POSTGRES, reflected that fact. The project's name was changed to PostgreSQL in 1996 to better represent the fact that it was a SQL database. The development team opted in 2007 to keep using PostgreSQL and the Postgres alias after an evaluation. PostgreSQL has automated view updates, materialized views, triggers, foreign keys, and stored procedures, as well as ACID-compliant transactions. From single workstations to data warehouses or Web services with thousands of users, it can manage it everything. Available for multiple operating systems including macOS Server, Windows, Linux, FreeBSD, and OpenBSD, this database is the de facto standard for macOS Server.

F. What is BigQuery?

The query engine in Google BigQuery is built right into the database, making it a serverless data warehouse that can handle massive amounts of data. The query engine can handle terabytes of data in seconds, and petabytes of data in just a few minutes. This performance can be attained without the need for index creation or maintenance. The capacity to scale and the lightning-fast performance of BigQuery are only two of its many impressive features. Because serverless, ad hoc [15] querying is so simple, the absence of infrastructure management is even more revolutionary because it makes way for novel approaches to work. The use of data to inform business decisions is becoming more common, and companies are promoting an environment in which information is shared freely across teams. BigQuery makes a large impact on the rate of innovation because it provides the technological means for a culture shift toward agility and openness.

G. What is BigTable?

BigTable is a NoSQL column-wide database, therefore it can store a lot of data with relatively few values in each column and row [16]. This system is also optimized for fast data reads and writes. The amount of information generated and stored increases as the number of people using popular apps rises. Since this is the case, commercial databases are inadequate. For the purpose of managing massive amounts of structured data, Google developed BigTable. The original purpose of the system was to manage Google's own data-intensive services like Google Earth and Google Analytics. Where does BigTable put all that information? The data items in BigTable are stored as key-value pairs in rows and columns. Multiple pieces of data can be stored in a single table cell, and column families are possible. Each piece of information also has a timestamp associated with it, which is used to track changes over time.

IV. MongoDB vs SQL

There are primarily two types of databases that are often used: SQL databases and NoSQL databases. There are some significant distinctions between these two databases that should be taken into account while choosing a database. Starting with storage architecture, relational databases like SQL are preferred over non-relational databases like NoSQL. In accordance with these architectures, NoSQL databases feature dynamic schema to handle unstructured data, while SQL databases employ Structured Query Language with a predetermined schema. NoSQL databases are horizontally scalable, whereas SQL databases are vertically scalable in terms of scaling. In contrast to NoSQL, where the databases are document, key-value, graph, or wide-column stores, SQL databases are table-based structures. While processing multi-row transactions is the primary use case for SQL, unstructured data like documents and JSON are better suited for NoSQL.

The most popular SQL databases are MySQL and PostgreSQL, whereas the most popular NoSQL databases are MongoDB and ElasticSearch. MongoDB adheres to a cloud-agnostic architecture, placing an emphasis on operating without difficulty in any cloud environment and independent of any particular cloud platform's constrained functionality. Tools that are cloud-agnostic can operate without issue on any cloud provider, including Google, AWS, and Microsoft Azure, which facilitates smooth service portability. No matter how the tools are moved, the outcome will remain the same because they are not dependent on the platform's core bespoke features. The SQL database, on the other hand, is a cloud-native database that is reliant on a certain cloud environment. One can effortlessly benefit from all the advanced features and plugins within that cloud platform thanks to this integration.

A. Ability to scale

When scalability is a concern, SQL databases can typically only be scaled vertically, which simply refers to expanding the server's processing power. The cost of this option is high, and the upgrading or downgrading process takes time. Multiple read replicas in MySQL can improve user experience. MongoDB, unlike SQL, offers horizontal scaling through the use of sharding. Data distribution across numerous hosts is called sharding. While the application is being scaled, MongoDB divides a huge dataset into smaller datasets across numerous instances without experiencing any downtime. Using a shard key, sharding manages horizontal scaling among servers. Sharding, then, copies bits of data called shards across many replica sets as opposed to copying data entirely. Together, these replica sets make use of all the data.

B. *Accessibility and Dependability*

The dependability and availability indicators are also impacted by the architecture of data distribution. While SQL databases were initially intended for isolated servers and later turned toward distributed databases to tackle risks, NoSQL databases, like MongoDB, were initially constructed with resilience in mind. In general, SQL servers make use of replication's power. Data is simply copied from the primary server node to the secondary server nodes during replication. In the event that the primary server malfunctions, this can help boost data availability and serve as a backup. A sharded cluster is created when replication and sharding combine, with each shard being replicated in turn to maintain the same high availability. While MySQL servers like AWS RDS don't have automatic replication for data availability and require explicit setup, MongoDB provides automatic sharding.

C. *Distribution of Transactions*

An operation on a single document in MongoDB is atomic. MongoDB employs embedded documents and arrays to capture relationships between data in a single document structure, obviating the requirement for multi-document transactions for most use cases, as opposed to normalizing across numerous documents and collections. However, there are some circumstances when atomic reads and writes to many documents are necessary. The multi-document transaction is supported by MongoDB. Transactions can be used across numerous activities, collections, databases, documents, and shards when they are dispersed.

Distributed transactions were launched by Amazon RDS for SQL Server in 2020 with the assistance of Microsoft Distributed Transaction Coordinator (MSDTC). Using the AWS Directory Service for Microsoft Active Directory, you may run distributed transactions on domain-joined DB instances. With MSDTC, you can either promote MSDTC operating on the same host as the Client application to the position of Transaction manager, or you can execute the transaction using SQL Server as the Transaction manager using connected servers.

V. Aurora vs the default MySQL and PostgreSQL

When configuring new database servers with Amazon RDS, the Aurora database engine is selected. Aurora utilizes the well-known Amazon Relational Database Service (Amazon RDS) management and administrative features. Aurora can manage provisioning, patching, backup, recovery, failure detection, and repair via the Amazon RDS AWS Management Console, AWS CLI commands, and AWS API activities. Aurora administrative activities frequently demand entire clusters of synchronized database servers, as opposed to individual database instances. The automatic clustering, replication, and storage allocation make the installation, operation, and scalability of your largest MySQL and PostgreSQL deployments simple and economical. To import data from Amazon RDS for MySQL and Amazon RDS for PostgreSQL into Aurora, you can set up one-way replication or generate and restore snapshots. Using one-click conversion tools, your existing Amazon RDS for MySQL and Amazon RDS for PostgreSQL applications can be converted to Aurora. Aurora's design maintains the essential strengths of transactional consistency in relational databases. It uses storage layer innovation to create a cloud-based database that can service modern applications without sacrificing performance. Customers appreciate Aurora because it offers the performance and availability of commercial-grade databases at a fraction of the expense. Since its inception, Aurora has enjoyed the most rapid growth of any service in AWS's history.

VI. BigQuery vs Bigtable

First, BigQuery is an OLAP (Online Analytical Processing) solution. Due to its high query latency, BigQuery is best suited for queries with considerable workloads, such as OLAP reporting and archiving operations. OLTP-style queries are discouraged by the architecture of BigQuery. A simple read-write operation in BigQuery requires nearly 1.8 seconds, whereas the identical action in Bigtable requires only 9 milliseconds. The quick read-by-key and update operations of Bigtable make it the ideal database for OLTP workloads. In the data model, tables contain data and rows contain columns (Type Array or Struct). Similar to a persistent map, the structure enables the addition of columns to rows. Each entry's unique primary key is displayed in rows to facilitate reading and updating. Bigtable can be utilized to manage reporting and OLAP workloads due to its efficient support for key-range iteration. Use BigQuery if interactive querying in an online analytical processing environment is a top priority. Secondly, BigQuery enables us to do complex, analytical SQL-based searches on massive datasets. Nevertheless, users might utilize NoSQL approaches. For performance reasons, it is recommended to denormalize data while establishing schemas and importing data into BigQuery. In contrast, the Bigtable NoSQL database service does not enable SQL or multi-row transactions. This is not a relational database. Bigtable is therefore unsuitable for many applications; it should only be used for data sets that can be modified and are at least 1 Terabyte in size, as anything smaller incurs significant overhead. After being uploaded to BigQuery, data is also immutable; it cannot be deleted or modified for a defined period of time. If an existing record requires modification, the division must be redone. BigQuery is a ”append-only” database that reduces data storage by removing partitions older than the chosen time to live. Contrary to. Bigtable organizes data into scalable tables, each of which is an ordered key/value map indexed by a timestamp, column key, and row key. This enables quick key-based lookups and data modifications. Typically, each row represents a unique object, and each column has unique data for each row. Read and write operations on rows are atomic, independent of the number of columns being read or written.

VII. MongoDB vs Spanner

Google Cloud Spanner is a globally consistent, horizontally scalable relational database service. It is the external manifestation of the core Google database that powers the company's most prominent features, such as Ads and Google Play. Relational Database is Spanner's major focus, making it a fundamental database model. Both as a fully managed cloud service and for implementation on self-managed infrastructure, MongoDB is currently one of the most popular document stores accessible. MongoDB's principal database model is the document store database model, while its subsidiary database models are Spatial DBMS, Search Engine, and Time Series DBMS. ScaleGrid for MongoDB Database and MongoDB Atlas are available from MongoDB. ScaleGrid for MongoDB Database is a fully managed hosting service for MongoDB Database on a broad selection of cloud providers and On-Premises. Automate your administration, scaling, and backups via a centralized platform. MongoDB Atlas is a global multi-cloud database with unparalleled distribution of the data and mobility across AWS, Azure, and Google Cloud, as well as automation for resource and workload optimization. MongoDB is a schema-free database that supports Linux, OS X, Solaris, and Windows server operating systems. Alternatively, Spanner is a flawed database system. It supports the programming languages Go, Java, JavaScript (Node.js), and Python. Spanner lacks server-side scripts, although it does allow Multi-source replication with three replicas for regional instances. Spanner also supports ACID transactions. MongoDB is implemented in C++, however it supports a wide variety of programming languages. MongoDB supports Multi-Source operations with MongoDB Atlas Global Clusters Source-replica replication and Multi-document ACID Transactions with snapshot isolation and uses JavaScript on its server side.

VIII. DynamoDB vs Spanner

Amazon DynamoDB stores information in Amazon's cloud and is a hosted, scalable database solution. DynamoDB is primarily a document store and key value store database. DynamoDB does not accept XML and is therefore schema-free. In contrast to SQL and server-side scripts, it allows partitioning, replication, ACID transactions, and secondary indexes. The Google Cloud The Spanner database service is a horizontally scalable relational database that maintains data consistency across all nodes in the network. It's the public face of the internal database that drives Google's flagship products like Google Ads and Play. Because of its emphasis on the relational paradigm, Spanner is a basic database system. Despite allowing Multi-source replication with three copies for regional instances, Spanner lacks server-side scripts. ACID transactions are also supported in Spanner.

IX. Conclusion

A literature review on cloud native databases and the methodologies they use has been carried out for the purposes of this research. In addition to that, we went through some of the fundamentals of well-known databases like as Spanner, MongoDB, DynamoDB, Amazon Aurora, PostgreSQL, BigQuery, and BigTable. Despite this, in the relevant literature, we have pointed out certain significant difficulties that might be encountered when developing and using distributed Databases. Both of these databases have been compared, and the benefits and drawbacks of each have been outlined for you. In a later phase of this research project, an extension of the current investigation, we intend to do an analysis of the performance of these databases, as well as a full comparison of these databases, and we will then make some recommendations.

References

[1] T. Goldschmidt, A. Jansen, H. Koziolek, J. Doppelhamer, and H. P. Breivold, "Scalability and robustness of time-series databases for cloud-native monitoring of industrial processes," in *2014 IEEE 7th International Conference on Cloud Computing*, 2014, pp. 602–609.

[2] A. J. Maulidin, F. Renaldi, and F. R. Umbara, "Online integration of sql and no-sql databases using restapis: A case on 2 furniture e-commerce sites," in *2020 3rd International Conference on Computer and Informatics Engineering (IC2IE)*, 2020, pp. 261–266.

[3] S. Ghule and R. Vadali, "Transformation of sql system to nosql system and performing data analytics using svm," in *2017 International Conference on Trends in Electronics and Informatics (ICEI)*, 2017, pp. 883–887.

[4] T.-C. Hsu, D.-M. Chang, and H.-J. Lee, "The study of application and evaluation with nosql databases in cloud computing," in *2014 International Conference on Trustworthy Systems and their Applications*, 2014, pp. 57–62.

[5] C. Li and W. Yang, "The distributed storage strategy research of remote sensing image based on mongo db," in *2014 Third International Workshop on Earth Observation and Remote Sensing Applications (EORSA)*, 2014, pp. 101–104.

[6] G. Vial, "Different databases for different strokes," *IEEE Software*, vol. 35, no. 2, pp. 80–85, 2018.

[7] H. Gupta, S. Mondal, S. Ray, B. Giri, R. Majumdar, and V. P. Mishra, "Impact of sql injection in database security," in *2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, 2019, pp. 296–299.

[8] M. Sharma, V. D. Sharma, and M. M. Bundele, "Performance analysis of rdbms and no sql databases: Postgresql, mongodb and neo4j," in *2018 3rd International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE)*, 2018, pp. 1–5.

[9] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. C. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google's globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, p. 8, 2013.

[10] J. Shute, M. Oancea, S. Ellner, B. Handy, E. Rollins, B. Samwel, R. Vingralek, C. Whipkey, X. Chen, B. Jegerlehner, K. Littlefield, and P. Tong, "F1 - the fault-tolerant distributed rdbms supporting google's ad business," in *SIGMOD*, 2012, talk given at SIGMOD 2012.

[11] D. F. Bacon, N. Bales, N. Bruno, B. F. Cooper, A. Dickinson, A. Fikes, C. Fraser, A. Gubarev, M. Joshi, E. Kogan, A. Lloyd, S. Melnik, R. Rao, D. Shue, C. Taylor, M. van der Holst, and D. Woodford, "Spanner: Becoming a sql system," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 331–343. [Online]. Available: https://doi.org/10.1145/3035918.3056103

[12] L. de Espona Pernas and E. Pustulka[, "Document versioning for mongodb," 2022.

[13] M. Diogo, B. Cabral, and J. Bernardino, "Cbench-dynamo: A consistency benchmark for nosql database systems," in *Performance Evaluation and Benchmarking for the Era of Cloud(s)*, R. Nambiar and M. Poess, Eds. Cham: Springer International Publishing, 2020, pp. 84–98.

[14] A. Verbitski, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, T. Kharatishvili, and X. Bao, "Amazon aurora: Design considerations for high throughput cloud-native relational databases," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1041–1052. [Online]. Available: https://doi.org/10.1145/3035918.3056101

[15] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive analysis of web-scale datasets," in *Proc. of the 36th Int'l Conf on Very Large Data Bases*, 2010, pp. 330–339. [Online]. Available: http://www.vldb2010.org/accept.htm

[16] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," in *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006, pp. 205–218.