**PROG8410-23S-Section 2 – NoSQL Database Implementation**

**LAB 4 Assignment**

**Features of Cassandra:**

Keeping latency at acceptable levels requires a data architecture that can handle the demands of globally deployed real-time applications. Apache Cassandra is perfectly suited to meet all these needs as it’s geographically distributed, and it can respond to spikes in traffic without any adverse effects.

Here are some of the massive features described below:

1. **Distributed:**

Each node in the cluster has the same role as the other clusters. There is no question of failure, and the data set will be distributed across the cluster, but one issue is that the master is not present in each node to support the service requests.

1. **Supports replication & Multi data centre replication:**

The replication factor comes with the best configurations in Cassandra. Cassandra is designed to have a distributed system for deploying many nodes across multiple data centres and other various vital features.

1. **Scalability:**

It is designed to r/w throughput, increase gradually as new machines are added without interrupting other applications.

1. **Fault-tolerance:**

Data is automatically stored & replicated for fault tolerance. If a node fails, then it is replaced within no time. Cassandra boasts a fault-tolerant design, ensuring the absence of any singular point of failure, thus guaranteeing uninterrupted availability for mission-critical applications that cannot tolerate any form of failure.

1. **MapReduce Support:**

It supports Hadoop integration with MapReduce support. Apache Hive & Apache Pig is also supported.

1. **Query Language:**

Cassandra has introduced the CQL (Cassandra Query Language). It is a simple interface for accessing Cassandra.

[**Cassandra Query Language**](https://www.geeksforgeeks.org/additional-functions-in-cql-cassandra-query-language/) [**(CQL)**](https://www.geeksforgeeks.org/additional-functions-in-cql-cassandra-query-language/)**:**

CQL has a simple interface for accessing Cassandra, an alternative to traditional SQL. CQL adds an abstraction layer to hide the implementation of structure & also provides the native syntax for collections.

1. **Transaction support:**

Cassandra supports properties like Atomicity, Consistency, Isolation, and Durability (ACID) properties of transactions.

**Citation -**[**https://www.geeksforgeeks.org/introduction-to-apache-cassandra/?ref=ml\_lbp**](https://www.geeksforgeeks.org/introduction-to-apache-cassandra/?ref=ml_lbp)

**Citation -** [**https://www.geeksforgeeks.org/introduction-to-apache-cassandra/?ref=ml\_lbp**](https://www.geeksforgeeks.org/introduction-to-apache-cassandra/?ref=ml_lbp)

**The Power of Cassandra:**

NoSQL databases evolved over the last decade as an alternative to single-instance relational database management systems (RDBMS), which needed help keeping up with the throughput demands and sheer volume of web-scale internet traffic.

They solve scalability problems through a process known as horizontal scaling, where multiple database server instances are linked to each other to form a cluster.

Some NoSQL database products were also engineered with data centre awareness, meaning the database is configured to logically group certain instances to optimize the distribution of user data and workloads. Cassandra is both horizontally scalable and data-centred aware.

Cassandra’s seamless and consistent ability to scale to hundreds of terabytes, along with its exceptional performance under heavy loads, has made it a crucial part of the data infrastructures of companies that operate real-time applications. This kind is expected to be highly responsive, regardless of the scale at which they operate. Consider the modern applications and workloads that must be reliable, like online video streaming apps, online banking services, or those that operate at massive, distributed scales, such as airline booking systems or popular retail apps.

The year 2007 witnessed the debut of Netflix's streaming service.; it used an Oracle database in a single data centre. As the number of users and devices (and data) proliferated, the limitations on scalability and the potential for failures seriously threatened Netflix’s success. With its distributed architecture, Cassandra was a natural choice, and by 2013, most of Netflix’s data was housed there. Netflix still uses Cassandra today, but not only for its scalability and rock-solid reliability. Its performance is crucial to the streaming media company – Cassandra runs 30 million operations per second on its most active single cluster, and 98% of its streaming data is stored on Cassandra.

Cassandra has demonstrated outstanding performance even when subjected to substantial workloads. It can consistently show high-speed throughput for writes per second on an essential commodity workstation. Cassandra’s desirable properties are maintained as more servers are added without sacrificing performance.

Business decisions that must be made in real-time require high-performing data storage, wherever the principal users may be. Cassandra enables enterprises to ingest and act on that data in real-time, at scale, worldwide. If acting quickly on business data is where an organization needs to be, then Cassandra can help you.

**Citation -** [**https://www.cio.com/article/402804/two-reasons-why-apache-cassandra-is-the-database-for-real-time-applications.html#:~:text=Cassandra%2C%20with%20its%20distributed%20architecture,scalability%20and%20rock%2Dsolid%20reliability**](https://www.cio.com/article/402804/two-reasons-why-apache-cassandra-is-the-database-for-real-time-applications.html#:~:text=Cassandra%2C%20with%20its%20distributed%20architecture,scalability%20and%20rock%2Dsolid%20reliability)**.**

**Successful Implementation of Cassandra by Netflix: A Case Study**

**Introduction:**

This case study examines the successful implementation of Apache Cassandra by Netflix and highlights the key factors that contributed to its successful adoption. By leveraging Cassandra's distributed architecture and scalability, the company achieved significant improvements in performance, reliability, and scalability in their data management system. This case study aims to provide insights and lessons learned from the implementation process, serving as a valuable resource for organizations considering the adoption of Cassandra.

**Background:**

In 2007, Netflix made a ground-breaking move by introducing its streaming service, revolutionizing the way people consumed entertainment. At that time, the platform relied on an Oracle database housed in a single data centre. Initially, this setup served Netflix well as it catered to a relatively small user base and manageable data requirements.

However, as the popularity of Netflix soared and the service gained millions of users worldwide, the limitations of their infrastructure began to emerge. The rapid proliferation of users, devices, and data put immense strain on the Oracle database and highlighted its inherent scalability constraints. Netflix faced the daunting challenge of ensuring seamless streaming experiences for an ever-increasing audience while maintaining reliability, performance, and uninterrupted service.

The reliance on a single data centres further heightened the risk of failures. Any disruptions or any outages in that central location had the potential to halt Netflix's operations entirely, resulting in frustrated customers and severe financial implications. The company recognized that to sustain its growth and maintain its position as a leading streaming platform, a fundamental shift in their database architecture was necessary.

In response to these challenges, Netflix began exploring alternative solutions that could offer the scalability, performance, and fault tolerance required to support its massive user base and data demands. This search eventually led them to Apache Cassandra, a highly scalable and distributed database system known for its ability to handle large volumes of data across multiple nodes.

**Identification of Requirements:**

It is important to figure out the exact requirements of the organization before Cassandra is implemented. This involves identifying the constraints of the present infrastructure, the expected growth rate, the anticipated data volumes, and the desired degree of scalability and performance.

In order to satisfy the growing needs of its streaming business, Netflix realized the necessity for a highly scalable, fault-tolerant, and performant database system.

**Evaluation and Planning:**

Once the requirements have been identified, the next thing to do is to compare multiple database management systems and choose the most suitable one. Upon evaluating Cassandra's applicability, Netflix considered factors such its distributed architecture, scalability potential, fault tolerance features, community support, and potential for integration with their current infrastructure. Cassandra was selected as the most suitable option after performing this evaluation.

**Architecture and Data Modelling:**

The architecture and data modelling for Netflix's Cassandra implementation needed an extensive amount of development.

To determine how to select the most optimal data model and partitioning techniques, they studied the data access patterns of their application while taking read and write workloads into account.

Understanding Cassandra's denormalized data model, designing primary keys and secondary indexes, and constructing tables that fit the specifications of the application were all required for this.

Netflix mainly emphasized on utilizing Cassandra's abilities in data distribution and partitioning in order to make sure efficient data storage and retrieval.

**Implementation and Integration:**

The implementation phase involved setting up Cassandra clusters and incorporating them into Netflix's present infrastructure. For the purpose of achieving maximum availability and data redundancy, Netflix established various Cassandra clusters across different divisions.

Replication factors, consistency levels, and various other performance-driven characteristics have been meticulously taken into consideration while building the clusters.

For a program to communicate with Cassandra, the source code of the application had to be altered. This included implementing various queries, employing Cassandra drivers or APIs, and establishing connections to the database.

**Performance Optimization:**

By implementing Cassandra, Netflix was able to address the scalability limitations that plagued their previous infrastructure. The distributed nature of Cassandra allowed them to distribute data across multiple nodes, enabling horizontal scaling and ensuring that the system could handle the ever-growing user base and data volume effectively.

Additionally, Cassandra's fault-tolerant design offered significant benefits to Netflix. The decentralized architecture reduced the risk of single points of failure, as data was replicated across multiple nodes. Even if one node experienced issues or went offline, the data remained accessible from other nodes, ensuring continuous service availability.

**Testing and Validation:**

Netflix conducted an extensive testing along with quality assurance with the aim to ensure an effortless transition and verify the implementation.

To determine the system's functionality, scalability, and dependability in various circumstances, load, stress, and functional tests have been carried out and they have simulated in a real-world workloads and scenarios.

Before moving Cassandra onto production, Netflix was capable of to discover and rectify any anomalies or inefficiencies throughout this testing phase.

To achieve the intended levels of performance, the team adjusted several types of parameters, that includes the total number of nodes, compaction techniques, and caching settings.

**Results and Benefits:**

The adoption of Cassandra marked a pivotal moment for Netflix. The new database system provided the necessary scalability, fault tolerance, and performance to support the streaming platform's rapid growth. It allowed Netflix to overcome the challenges associated with handling a massive user base, diverse device ecosystem, and exponentially expanding data requirements.

With Cassandra in place, Netflix could deliver seamless streaming experiences to millions of users simultaneously, regardless of geographic location. The decentralized nature of the database system ensured high availability, resilience, and data redundancy, minimizing the risk of downtime and disruptions.

**Challenges and Lessons Learned:**

Cassandra implementation in an enormous infrastructure like Netflix involves various problems like operational complexity, migration complexity, data model design, data consistency, monitoring and maintenance, and team collaboration.

Investing in knowledge, meticulous preparation, and evaluation, maintaining data integrity throughout migration, constant monitoring and maintenance, scaling issues, risk mitigation for vendor lock-in, and encouraging good team collaboration are some of the valuable lessons. These lessons help in improving the scalability, performance, and reliability of the system.

**Conclusion:**

Overall, the successful implementation of Cassandra by Netflix exemplifies the critical role that scalable and fault-tolerant database systems play in supporting the growth and sustainability of modern digital services. By recognizing the limitations of their initial infrastructure and proactively adopting a more suitable solution, Netflix was able to maintain its success, secure customer satisfaction, and remain at the forefront of the streaming industry.

**Citation: Welcome to Apache Cassandra’s documentation! | Apache Cassandra Documentation. (n.d.). Apache Cassandra.** [**https://cassandra.apache.org/doc/latest/**](https://cassandra.apache.org/doc/latest/)

**Citation: Netflix TechBlog. (2001, June 20). Netflix Techblog.** [**https://netflixtechblog.com/**](https://netflixtechblog.com/)