RESUME: SUNDAR R

SOLUTION ARCHITECT

CONTACT INFORMATION

- Email: sundar4consulting@gmail.com

OBJECTIVE

To leverage my skills and experience in Casandra and golang to design and implement scalable and reliable solutions for large data sets.

WORK EXPERIENCE

A leading logistics supply chain provider - Inc.: Solution Architect (01/01/2022 - Present)

- Designed and developed a data pipeline using Casandra and golang to process and analyze billions of records from various sources. Implemented best practices for data modeling, performance tuning, and security. Led a team of developers and coordinated with stakeholders to deliver the project on time and within budget.

Largest retailer in UK : Solution Architect (01/01/2020 - 31/12/2021)

- Developed and maintained a web application using golang and Casandra that provided real-time insights and analytics for customers. Integrated the application with various APIs and services. Troubleshot and resolved issues related to data quality, performance, and availability.

A leading Financial institution: Solution Architect (01/01/2018 - 31/12/2019)

- Contributed to the development of a distributed system using Casandra and golang that handled large volumes of data from multiple sources. Implemented features and enhancements based on user feedback and business requirements. Wrote unit tests and documentation for the code.

EDUCATION

- University of Madras, India (01/01/2000 - 31/12/2004)
- Bachelor of Technology, Computer Science

SKILLS

- Casandra

- Golang
- Data Engineering
- Data Modeling
- Data Analysis
- Web Development
- API Integration
- Team Leadership
- Agile Methodology

INTERESTS

- Data Science
- Machine Learning

LANGUAGES

- English
- Hindi

CERTIFICATIONS

- Casandra Developer Certification
- Golang Developer Certification

PUBLICATIONS

- Sundar R., and Rajesh, R. (2017). A scalable and reliable data pipeline using Casandra and golang. International Journal of Data Engineering, 9(1), 15-25.

Appendix – A

Optimizing Cassandra for Efficient Data Retrieval and Processing

To achieve optimal performance in Cassandra-based projects, several key techniques can be employed to ensure efficient data retrieval and processing.

DATA MODELING

Proper data modeling is crucial for efficient data retrieval in Cassandra. This involves designing a data model that aligns with how the data will be accessed . A well-designed data model can significantly improve query performance.

QUERY PATTERN OPTIMIZATION

Optimizing query patterns is essential for efficient data retrieval. This involves understanding how Cassandra handles queries and designing queries that minimize the amount of data being scanned

PARTITIONING DATA

Partitioning data effectively is critical for efficient data retrieval. This involves dividing data into smaller, more manageable pieces that can be stored across multiple nodes

INDEXING AND CACHING

Implementing indexing and caching strategies can significantly improve query performance. Indexing allows Cassandra to quickly locate specific data, while caching reduces the number of requests made to the database

DATA TYPE OPTIMIZATION

Using appropriate data types can also improve query performance. This involves selecting data types that minimize storage requirements and optimize query execution

COMPRESSION

Compressing data can reduce storage requirements and improve query performance. Cassandra provides various compression algorithms that can be used to compress data

Appendix – B

Challenges of Handling Large Datasets with Cassandra

Handling large datasets with Cassandra can be challenging, and some common issues that arise include:

- **Long pause times**: Cassandra deployments can experience long pause times, which can impact performance
- **Heap pressure**: Cassandra can also experience heap pressure, which can lead to performance issues
- **Query optimization**: Efficiently querying and managing large datasets across multiple nodes requires optimized query processing techniques

Solutions for Handling Large Datasets with Cassandra

To mitigate these challenges, several solutions can be employed:

- **Tuning options**: Tuning options can be used to mitigate long pause times and heap pressure in Cassandra deployments
- **Bulk upload API**: Implementing a bulk upload API with rate limiting can help handle large volumes of data writes consistently [3]
- **Distributed architecture**: Cassandra's distributed architecture allows it to manage massive volumes of data across numerous machines, making it well-suited for handling large datasets
- **Using GoLang**: GoLang can be used to build solutions that ingest data and apply user-defined transforms, making it a suitable choice for handling large datasets
- **Data processing**: Solutions like Hadoop and Cassandra can be used to store and process big data
- **NoSQL database**: Cassandra's NoSQL database capabilities make it well-suited for handling large amounts of unstructured data [59].
- **Query optimization techniques**: Optimized query processing techniques are crucial for efficiently querying and managing large datasets across multiple nodes

Appendix C :

Data Consistency and Consistency Models :

Apache Cassandra, a distributed data store, prioritizes availability over consistency, allowing it to respond with possibly stale data .This is a trade-off to achieve high availability, which involves some read and write latencies . According to the CAP theorem, a distributed data store can only provide two of three guarantees: consistency, availability, and partition tolerance

Consistency Models in Cassandra

Cassandra allows users to choose the consistency level for their read and write operations, balancing between consistency and availability Higher consistency levels ensure stronger data consistency but may impact performance. The system can be tailored to achieve a balance between performance and consistency.

Trade-offs between Consistency, Availability, and Partition Tolerance

In a distributed system, partition tolerance is a must, leaving a choice between availability and consistency .Cassandra's tunable consistency model enables users to make this tradeoff The PACELC theorem describes the tradeoff between availability, consistency, and latency

Replication and Data Availability

Replicas ensure data availability in Cassandra . The system can be distributed across multiple data centers, ensuring high availability .

Appendix D :

DATA CONSISTENCY STRATEGIES IN CASSANDRA

Cassandra provides various consistency levels to manage availability versus data accuracy .The consistency level determines how many replica nodes must respond to a write operation before it is considered successful. The commonly used consistency levels are:

- **LOCAL_QUORUM**: 51% or a majority of replica nodes within the same data center
- **Eventual Consistency**: a weak guarantee, where the system will eventually reach a consistent state
- **Strong Consistency**: a stronger guarantee, where all nodes agree on the latest state
- **Causal Consistency**: ensures that the order of updates is preserved

IMPLEMENTING CONSISTENCY MODELS IN GOLANG APPLICATIONS

To implement data consistency models in GoLang applications, you can use the following strategies:

1. **Configure consistency levels**: Set the consistency level for a session or per individual read or write operation
2. **Use data modeling**: Design your data model to ensure data consistency, such as using a single row for each entity
3. **Implement compaction strategies**: Regularly compact data to ensure consistency and prevent data inconsistencies
4. **Monitor data consistency**: Implement comprehensive monitoring to detect and resolve data inconsistencies
5. **Use replication factor**: Configure the replication factor to control how many times data is replicated

Appendix E :

Optimizing GoLang Applications for High-Throughput and Low-Latency Data Ingestion into Cassandra

To optimize GoLang applications for high-throughput and low-latency data ingestion into Cassandra, several strategies and tools can be employed.

Choose Optimized Libraries and Frameworks

Select libraries and frameworks that are designed for high-performance and real-time applications, such as OpenCV . For Cassandra, use a high-performance driver like `gocql` or `cassandra-gocql` to interact with the database.

Optimize Data Ingestion Workflows

Mitigate the impact of state management in cloud stream processing to achieve low-latency data ingestion . Implement efficient data ingestion workflows that can handle high-throughput and minimize latency.

Leverage GoLang Performance Optimization Techniques

Apply GoLang performance optimization strategies, tools, and methods to build high-performance applications . This includes optimizing memory allocation, reducing garbage collection pauses, and using efficient data structures.

Cassandra Performance Optimization

Optimize Cassandra performance by tuning configuration settings, such as increasing the number of threads, adjusting memory allocation, and using efficient data models .Regularly monitor Cassandra performance metrics to identify bottlenecks and optimize accordingly.

Efficiency and Performance Optimizations

Apply efficiency and performance optimizations at all layers of the system, from the lowest levels to the highest . This includes optimizing data ingestion, processing, and storage layers.

IMPLEMENTING QUORUM CONSISTENCY MODEL IN GOLANG APPLICATION USING CASSANDRA DRIVER

To implement quorum consistency model in a GoLang application using the Cassandra driver, you need to understand the concept of quorum consistency and its trade-offs.

WHAT IS QUORUM CONSISTENCY?

Quorum consistency is a consistency model in Cassandra that ensures a certain number of nodes respond when defining the quorum . It provides a balance between availability and consistency by requiring a minimum number of nodes to acknowledge a write operation before considering it successful .

CONFIGURING QUORUM CONSISTENCY IN CASSANDRA

To configure quorum consistency in Cassandra, you need to set the replication strategy on your keyspaces to `NetworkTopologyStrategy` . This strategy allows you to define the number of replicas for each data center.

SETTING WRITE CONSISTENCY LEVEL

The write consistency level determines the number of replica nodes that must acknowledge a write operation before it is considered successful . You can set the write consistency level to `QUORUM` to ensure that a quorum of nodes respond to a write operation.

TRADE-OFFS OF QUORUM CONSISTENCY

Quorum consistency provides a balance between availability and consistency, but it also introduces trade-offs. With quorum consistency, you may experience higher latency and lower availability compared to other consistency models . Additionally, quorum consistency may not be suitable for applications that require strong consistency guarantees.

IMPLEMENTING QUORUM CONSISTENCY IN GOLANG USING CASSANDRA DRIVER

To implement quorum consistency in a GoLang application using the Cassandra driver, you need to:

1. Import the Cassandra driver package in your GoLang application.
2. Create a Cassandra session with the `NetworkTopologyStrategy` replication strategy.
3. Set the write consistency level to `QUORUM`.
4. Execute write operations with the configured consistency level.

Here is an example code snippet:

```
1
2import (
3       "github.com/gocql/gocql"
4)
5
6func main() {
7       cluster := gocql.NewCluster("localhost")
8       cluster.Keyspace = "my_keyspace"
9       cluster.Consistency = gocql.Quorum
10      session, _ := cluster.CreateSession()
11      defer session.Close()
12
13      // Execute write operation with quorum consistency
14      session.Query("INSERT INTO my_table (id, name) VALUES (?,?)", "id",
"name").Exec()
15}
```

Design Considerations:

GoLang libraries for efficient data serialization and deserialization to reduce overhead during data ingestion into Cassandra?

Implement idempotent operations in my GoLang application to ensure data consistency and prevent data duplication during high-throughput data ingestion into Cassandra?