ALIBABA CLOUD

Alibaba Cloud

ApsaraDB for Cassandra

Best practices

Document Version: 20211216

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

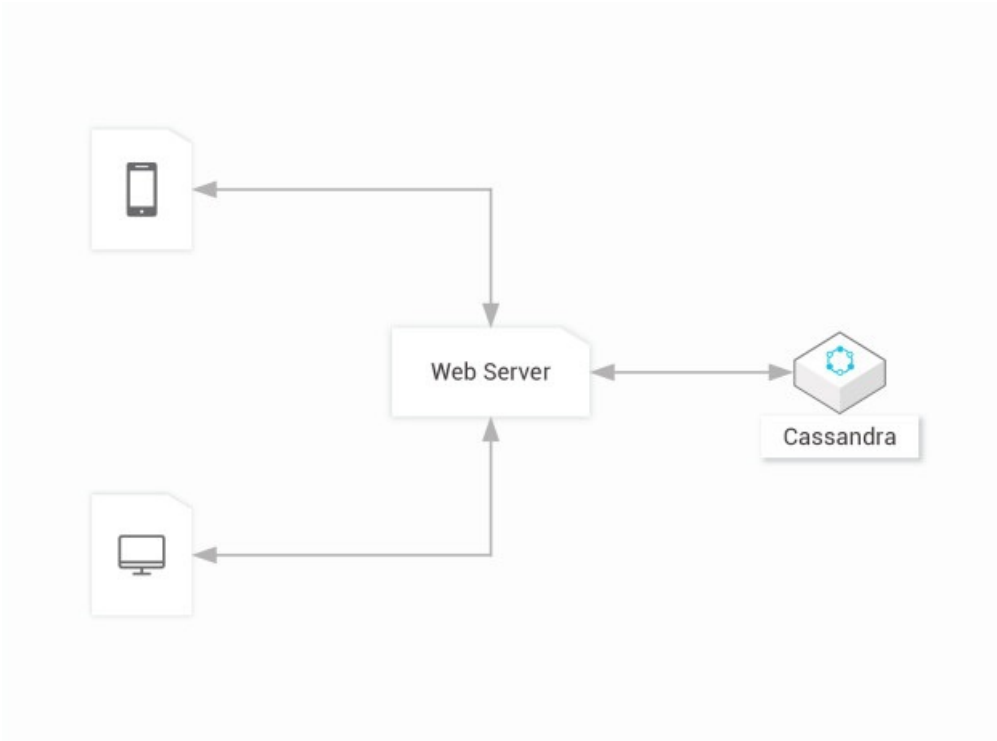
Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.ApsaraDB for Cassandra scenarios

ApsaraDB for Cassandra supports access requests that require high concurrency and low latency by providing high availability and elastic scaling features. ApsaraDB for Cassandra is applicable to online Internet scenarios that involve a large amount of data, such as messages, feed streams, order queries, and websites.

Online scenarios



Highlights

- High availability: Your business is not affected by single points of failure.
- Low latency: 99.9% of network latency is in milliseconds.
- Elasticity: The computing and storage capabilities can be flexibly scaled as your business grows.
- Multi-active data centers: Native Apache Cassandra can be deployed across multiple data centers. Applications use the LOCAL_QUORUM consistency level to access ApsaraDB for Cassandra, and ApsaraDB for Cassandra automatically forwards the requests to another data center. This provides better availability and disaster recovery capabilities.
- Disaster recovery: Data and service availability are not affected by physical damage.
- Simple implementation: Data synchronization between data centers is implemented by native Cassandra mechanisms. Few changes are made for applications.
- Data-driven services - risk control, recommendation, and IoT: The user behavior data of multiple data sources is stored in ApsaraDB for Cassandra in real time, and user profiles are created based on the user behavior data. Services such as risk control and recommendation are provided by using the user profiles.

- Low storage costs: Sparse storage schemes are used. This is suitable for storing highly compressed user profile data.
- High scalability: Linear scalability is available. The computing and storage capabilities can be flexibly scaled.
- Low latency: 99.9% of network latency is in milliseconds.

2.Use static columns

Scenario

Assume that a Cassandra table stores the user information (such as email address and password) and user status updates. Generally, the user information remains almost unchanged when the user status frequently changes. If the user information is updated along with each status update, a large amount of storage space is used.

Cassandra provides the static column feature to solve this problem. The static column that is declared in the same partition key has only one value. This indicates that only one set of data in this column is stored.

Define a static column

Add STATIC to the end of a column name to define the column as a static column, as shown in the following example:

```
CREATE TABLE "iteblog_users_with_status_updates" (
  "username" text,
  "id" timeuuid,
  "email" text STATIC,
  "encrypted_password" blob STATIC,
  "body" text,
  PRIMARY KEY ("username", "id")
);
```

The preceding command sets both email and encrypted_password fields to STATIC. This indicates that a username has only one email value and one encrypted_password value.

Limits on static columns

You cannot add the STATIC keyword to a column in each table. Static columns have the following limits:

- Clustering columns (also known as clustering keys) are not defined for the table, as shown in the following example:

```
cqlsh:iteblog_keyspace> CREATE TABLE "iteblog_users_with_status_updates_invalid" (
                    ...   "username" text,
                    ...   "id" timeuuid,
                    ...   "email" text STATIC,
                    ...   "encrypted_password" blob STATIC,
                    ...   "body" text,
                    ...   PRIMARY KEY ("username")
                    ... );
InvalidRequest: Error from server: code=2200 [Invalid query] message="Static columns are on
ly useful (and thus allowed) if the table has at least one clustering column"
```

You cannot create static columns in the iteblog_users_with_status_updates_invalid table because the table has only a primary key and clustering columns are not defined for the table. The more rows of data for the same partition key result in the better performance for the static column. If no clustering columns are defined, a primary key value identifies a unique row in a partition and no static columns are required.

- The table is set to a COMPACT STORAGE table, as shown in the following example:

```
cqlsh:iteblog_keyspace> CREATE TABLE "iteblog_users_with_status_updates_invalid" (
                    ...   "username" text,
                    ...   "id" timeuuid,
                    ...   "email" text STATIC,
                    ...   "encrypted_password" blob STATIC,
                    ...   "body" text,
                    ...   PRIMARY KEY ("username", "id")
                    ... )WITH COMPACT STORAGE;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Static columns are no
t supported in COMPACT STORAGE tables"
```

- Columns used as part of partition keys or clustering columns cannot be set to STATIC, as shown in the following example:

```
cqlsh:iteblog_keyspace> CREATE TABLE "iteblog_users_with_status_updates_invalid" (
                    ...   "username" text,
                    ...   "id" timeuuid STATIC,
                    ...   "email" text STATIC,
                    ...   "encrypted_password" blob STATIC,
                    ...   "body" text,
                    ...   PRIMARY KEY ("username", "id")
                    ... );
InvalidRequest: Error from server: code=2200 [Invalid query] message="Static column id cann
ot be part of the PRIMARY KEY"
cqlsh:iteblog_keyspace> CREATE TABLE "iteblog_users_with_status_updates_invalid" (
                    ...   "username" text,
                    ...   "id" timeuuid,
                    ...   "email" text STATIC,
                    ...   "encrypted_password" blob STATIC,
                    ...   "body" text,
                    ...   PRIMARY KEY (("username", "id"), email)
                    ... );
InvalidRequest: Error from server: code=2200 [Invalid query] message="Static column email c
annot be part of the PRIMARY KEY"
```

Insert data into a table that contains static columns

Data is inserted into a table that contains static columns in a similar way in which data is inserted into a standard table. For example, you can execute the following statements to insert data into the iteblog_users_with_status_updates table:

```
cqlsh:iteblog_keyspace> INSERT INTO "iteblog_users_with_status_updates"
                    ... ("username", "id", "email", "encrypted_password", "body")
                    ... VALUES (
                    ...   'iteblog',
                    ...   NOW(),
                    ...   'iteblog_hadoop@iteblog.com',
                    ...   0x877E8C36EFA827DBD4CAFBC92DD90D76,
                    ...   'Learning Cassandra!'
                    ... );
cqlsh:iteblog_keyspace> select username, email, encrypted_password, body from iteblog_users
_with_status_updates;
 username | email                      | encrypted_password                 | body
----------+----------------------------+------------------------------------+--------------
-------
  iteblog | iteblog_hadoop@iteblog.com | 0x877e8c36efa827dbd4cafbc92dd90d76 | Learning Cass
andra!
(1 rows)
```

A row of data is inserted into the table. The preceding statements have performed the following two operations:

- All the data in the email and encrypted_password columns for the iteblog username is set to iteblog_hadoop@iteblog.com and 0x877e8c36efa827dbd4cafbc92dd90d76.
- A row for the Learning Cassandra! body is added to the iteblog partition. Execute the following statements to insert another row into the table:

```
cqlsh:iteblog_keyspace> INSERT INTO "iteblog_users_with_status_updates"
                    ... ("username", "id", "body")
                    ... VALUES ('iteblog', NOW(), 'I love Cassandra!') ;
cqlsh:iteblog_keyspace> select username, email, encrypted_password, body from iteblog_users
_with_status_updates;
 username | email                      | encrypted_password                 | body
----------+----------------------------+------------------------------------+--------------
-------
  iteblog | iteblog_hadoop@iteblog.com | 0x877e8c36efa827dbd4cafbc92dd90d76 | Learning Cass
andra!
  iteblog | iteblog_hadoop@iteblog.com | 0x877e8c36efa827dbd4cafbc92dd90d76 |   I love Cass
andra!
(2 rows)
cqlsh:iteblog_keyspace>
```

The statements show that the email and encrypted_password columns are not specified when data is inserted into the table. However, the query result shows that the values of the two columns in the new row are the same as the values that are specified in the preceding statements.

The user modifies the email address due to some reasons, as shown in the following example:

```
cqlsh:iteblog_keyspace> UPDATE iteblog_users_with_status_updates SET email = 'iteblog@itebl
og.com'
                    ... WHERE username = 'iteblog';
cqlsh:iteblog_keyspace> select username, email, encrypted_password, body from iteblog_users
_with_status_updates;
 username | email               | encrypted_password                 | body
----------+---------------------+------------------------------------+---------------------
  iteblog | iteblog@iteblog.com | 0x877e8c36efa827dbd4cafbc92dd90d76 | Learning Cassandra!
  iteblog | iteblog@iteblog.com | 0x877e8c36efa827dbd4cafbc92dd90d76 |   I love Cassandra!
(2 rows)
```

The query result shows that all the emails of the iteblog username are modified as the new email because the email column is static.

The table stores user information such as the email address and password. If a user modifies the email address and password on the frontend page, the backend system must obtain the current email address and password by executing the following statement:

```
cqlsh:iteblog_keyspace> SELECT "username", "email", "encrypted_password"
                    ... FROM "iteblog_users_with_status_updates"
                    ... WHERE "username" = 'iteblog';
 username | email               | encrypted_password
----------+---------------------+------------------------------------
  iteblog | iteblog@iteblog.com | 0x877e8c36efa827dbd4cafbc92dd90d76
  iteblog | iteblog@iteblog.com | 0x877e8c36efa827dbd4cafbc92dd90d76
(2 rows)
```

All the rows of the email addresses and passwords for the iteblog username are returned. This result is unexpected. You can execute the SELECT DISTINCT statement to eliminate duplicate rows:

```
cqlsh:iteblog_keyspace> SELECT DISTINCT "username", "email", "encrypted_password"
                    ... FROM "iteblog_users_with_status_updates"
                    ... WHERE "username" = 'iteblog';
 username | email               | encrypted_password
----------+---------------------+------------------------------------
  iteblog | iteblog@iteblog.com | 0x877e8c36efa827dbd4cafbc92dd90d76
(1 rows)
```

All the duplicate rows for the iteblog username are eliminated, and a unique row appears in the end.

Cassandra does not perform the DISTINCT operation for all the data of the iteblog username. This is because a set of the static column data is stored in the underlying system.

3.Lindorm, making ApsaraDB for Cassandra more powerful

This topic introduces ApsaraDB for Lindorm (Lindorm) and describes the benefits of ApsaraDB for Cassandra optimized by Lindorm.

Introduction to Lindorm

Lindorm is a cloud-native, multi-model database service that is applicable to all scales. Lindorm provides cost savings in data storage and processing, and supports the pay-as-you-go billing method. Lindorm is compatible with the standard APIs of multiple open source platforms, such as Apache HBase, Apache Cassandra, Apache Phoenix, OpenTSDB, Apache Solr, and SQL. Lindorm uses the following data models to organize and standardize data: time series, wide table, search, and file. Lindorm is intended for enterprises in the Internet, Internet of Things (IoT), Internet of Vehicles (IoV), advertising, social networking, monitoring, gaming, and risk control sectors. Lindorm also provides strong support for business-critical systems of Alibaba Group.

Lindorm uses a cloud-native, multi-model architecture that decouples compute and storage. Lindorm provides the following benefits: elasticity, cost savings, ease of use, high compatibility, and stability. Lindorm allows you to store and analyze the following types of data: metadata, log files, bills, tags, messages, reports, dimension tables, results tables, feeds, user profiles, device data, monitoring data, sensor data, small-sized files, and small-sized images. Lindorm provides the following key features:

- Multi-model integration: Lindorm supports the following types of data models: wide table, time series, search, and file. Lindorm is compatible with the APIs of multiple open source database management systems, and also provides a separate API. Data can be transformed and synchronized among different models. This makes application development more agile, flexible, and efficient.
- Cost savings: Lindorm is capable of handling tens of millions of concurrent requests and responds with a latency of a few milliseconds. Lindorm greatly reduces the storage cost by using high-density and cost-effective media, automatic cold and hot data separation, and adaptive compression.
- Cloud-native support and high scalability: Lindorm allows you to scale computing and storage resources separately. Lindorm also provides serverless services which you can scale and pay for based on actual usage.
- Open data ecosystem: Lindorm provides multiple easy-to-use features, such as data exchanges, processing, and subscription. In addition, Lindorm is highly compatible with MySQL, Spark, Flink, and Kafka.

Challenges with Apache Cassandra

Apache Cassandra is an open source, distributed NoSQL database that is designed based on Amazon DynamoDB and Google Bigtable. Apache Cassandra is decentralized, supports consistency level customization, and provides a SQL-like language Cassandra Query Language (CQL). However, you may encounter several challenges when you use Apache Cassandra. For example, Apache Cassandra does not decouple compute and storage. If you want to expand storage, you must first migrate data. This complicates resource scaling and makes Apache Cassandra weak in handling traffic spikes. Lindorm uses an architecture that decouples compute and storage to support elastic scaling.

Apache Cassandra requires you to periodically repair all replicas. Otherwise, issues such as ghost keys may occur. However, the repairing task may compromise service stability because it consumes large amounts of system resources. Lindorm uses storage plug-ins to maintain consistency. When you write data to Lindorm, the data is written to a specified number of replicas. This prevents data inconsistency and saves you the hassle of repairing replicas.

Benefits of ApsaraDB for Cassandra optimized by Lindorm

- Better performance: Compared with Apache Cassandra, ApsaraDB for Cassandra optimized by Lindorm provides higher data throughput and lower latency when handling large amounts of data.
- Separation of cold and hot data: ApsaraDB for Cassandra optimized by Lindorm uses various storage media and compression algorithms to store cold and hot data separately. This improves the efficiency of retrieving hot data and reduces the expenses on cold data storage.
- Pay-as-you-go: ApsaraDB for Cassandra optimized by Lindorm provides the Cluster Edition and Serverless Edition. When you use the Serverless Edition, you do not need to manage resources. The Serverless Edition scales resources based on workloads and is billed based on the number of requests and databases. The Serverless Edition supports the pay-as-you-go billing method and is easy to use and cost-effective.
- Indexes and wide tables: ApsaraDB for Cassandra optimized by Lindorm uses full-text indexes to accelerate queries on wide tables, and supports queries that are written in CQL.
- Various data channels: ApsaraDB for Cassandra optimized by Lindorm provides various data channels, such as search engines, online transaction processing (OLTP) systems, data warehouses, and log queues.
- Enterprise-class features: ApsaraDB for Cassandra optimized by Lindorm provides a variety of enterprise-class features. For more information, refer to the following figure. For more information about the enterprise-class features, go to the product page of Lindorm.