AWS Whitepaper

Optimizing AWS Database Migration Service Performance with Amazon Redshift as Target

Optimizing AWS Database Migration Service Performance with Amazon Redshift as Target: AWS Whitepaper

Table of Contents

Optimizing AWS Database Migration Service Performance with Amazon Redshift as Target

Publication date: **July 15, 2022** (*Document revisions*)

Abstract

This whitepaper is intended for Amazon Web Services (AWS) customers who are migrating data from any supported source database to Amazon Redshift data warehouse using AWS Database Migration Service (AWS DMS). AWS DMS helps customers migrate databases to the AWS Cloud quickly and securely by replicating data from any supported source to any supported target. AWS DMS also supports continuous data capture (CDC) functionality, where it replicates data from source to target on an ongoing basis.

A data migration that is not well planned or well tested, whether it is a one-time event or ongoing, may cause issues with respect to AWS DMS source and target latencies, which can potentially delay or stall the migration efforts. This whitepaper presents information about important migration considerations, highlights the best practices to be followed during the migration process, and provides an optimal strategy for a successful database migration to the Amazon Redshift data warehouse.

Are you well-architected?

The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems in the cloud. The six pillars of the Framework allow you to learn architectural best practices for designing and operating reliable, secure, efficient, cost-effective, and sustainable systems. Using the AWS Well-Architected Tool, available at no charge in the AWS Management Console, you can review your workloads against these best practices by answering a set of questions for each pillar.

For more expert guidance and best practices for your cloud architecture—reference architecture deployments, diagrams, and whitepapers—refer to the AWS Architecture Center.

Introduction

It is no secret that customers have strained relationships with their traditional database vendors. The pattern is the same: those databases are expensive, proprietary, designed for lock-in, and come

with punitive licensing terms. That's why so many customers have accelerated their migration to AWS Cloud databases. They want the benefits of reduced capital and operational costs, increased IT staff productivity, scalability, a modern and open architecture, a pay-as-you-go model that charges only for services used, and the business value made possible by the unequaled pace of innovation at AWS.

In the past few years, hundreds of thousands of customers have migrated their databases using AWS Database Migration Service (AWS DMS). AWS DMS is a fully managed service that allows customers to migrate their relational databases, non-relational databases, and data warehouses to AWS with virtually no downtime.

As of June 2022, more than 685,544 databases have been migrated to AWS using AWS Database Migration Service. One of the common targets for AWS DMS for data migration is Amazon Redshift. Amazon Redshift is a fully managed, petabyte-scale data warehouse service in the cloud. With an Amazon Redshift database as a target, you can migrate data from all of the supported source databases.

Database migrations tend to be complex and error-prone, so AWS advocates taking a planned and well-tested approach. Various factors such as database performance, network throughput, AWS DMS resources such as memory, CPU, number of instances, Amazon Redshift queue and memory management, load on the Amazon Redshift cluster at the time of migration, and other conditions influence the migration process. The most common issues seen during the migration process are AWS DMS source and target latency issues, where the process of capturing changes from the source database or the process of applying the changes to the target is delayed due to various reasons, affecting the entire migration process.

This whitepaper provides an optimal path for AWS DMS migration with Amazon Redshift as a target. With any deployment on AWS, there are many different considerations and options, so your approach may vary slightly from the approach in this paper. The whitepaper explains the various phases involved in the migration process as well as storage and networking considerations, and provides pointers on optimizing the performance of different components in the architecture such as source database, connectivity, AWS DMS, and Amazon Redshift as the target. Lastly, the whitepaper emphasizes the importance of performing tests in your staging environments to understand how these components work under load. It includes information on the approach you can use to troubleshoot issues within your environments.

Terminology

The following definitions are for common terms that are referenced throughout this paper:

- **AWS DMS Tasks** – Database Migration Service tasks specify tables, views, and schemas to use for migration
- **Full load** – The full load of existing data performed by doing a source table scan
- **Source Latency/CDCLatencySource** – The gap in seconds between when the source database wrote an event to its transaction log and when AWS DMS captured that change
- **Target Latency/CDCLatencyTarget** – The gap between when a commit is seen by AWS DMS while reading the source transaction log and when the changes of that commit are seen in the target
- **IOPS** – Input/output (I/O) operations per second (Ops/s)
- **Throughput** – Read/write transfer rate to storage (MB/s)

Database migration considerations

A carefully planned data migration strategy is important to prevent a sub-optimal experience that ends up creating more problems than it solves. It is important to be aware of different phases in the migration process to plan for and to optimize the performance of each of those individual stages, to prevent any delays in the data migration. Accurately estimating the storage needs and choosing the right networking connectivity between components helps prevent migration issues.

Migration phases

Following are the various phases typically seen during the data migration process:

Assess source database

As a first step in the data migration process, gather information about the tables in the source database that need to be migrated to the target database. Understand the requirement of the organization, whether you need to perform a one-time migration (via full load) or whether you need to migrate the data on an ongoing basis (continuous data capture). It is important to be aware of the table sizes as well as database sizes, and verify whether any large objects (LOBs) are present in the migration scope. LOBs are generally migrated one at a time, piece by piece. They can significantly slow down the migration and require more processing resources than standard objects. For more information on how LOBs are handled by AWS DMS, refer to Setting LOB support for source databases in an AWS DMS task.

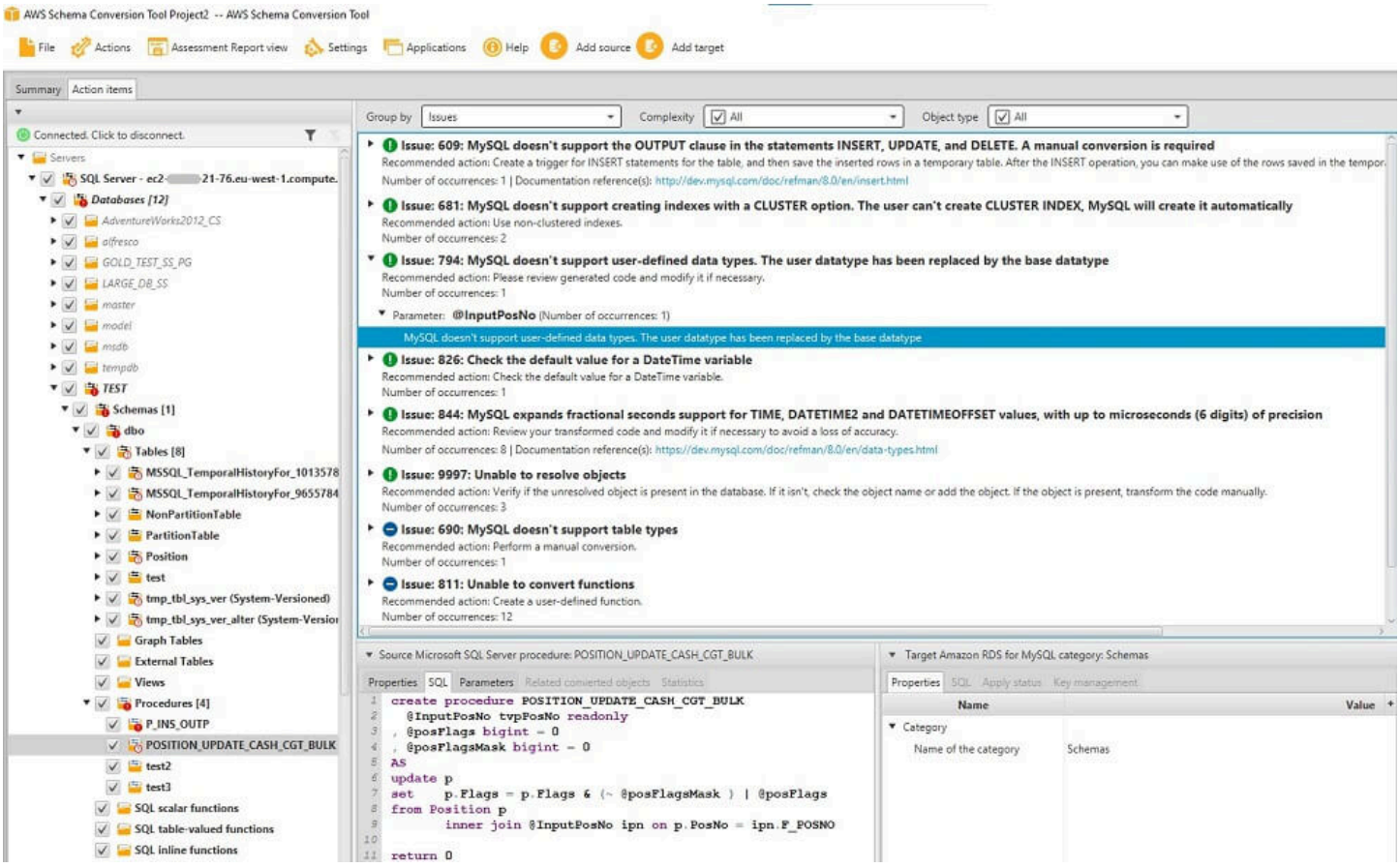
Also understand the nature of transactions (Data Manipulation Language (DML), Data Query Language (DQL), Data Definition Language (DDL) and so on.) happening on your database and calculate Transactions per Second (TPS). Based on this information, you can isolate the busy tables which have a higher number of transactions to prevent them from impacting the migration of other tables, and you can allocate additional resources for migrating these tables. Be aware of indexes, triggers, and referential integrity database constraints, which can affect the migration performance. It's a best practice to use native monitoring for your source database (DB) engine to be sure that your DB isn't experiencing a performance bottleneck; for example, CPU or memory contention, or input/output (IO) saturation.

If the source database is already constrained on resources, the data migration will be further slowed down. It is important to understand the behavior of your source database engine to come

up with an optimal migration plan. For a comprehensive list of valid sources supported by AWS DMS, refer to Sources for AWS DMS.

Since the target engine (Amazon Redshift) is different from the source database engine, you might need to convert the source database schema and a majority of database code objects, including views, stored procedures, and functions, to a format compatible with the target. AWS Schema Conversion Tool (AWS SCT) can help out with this requirement. When using SCT, you can create mapping rules and any objects that cannot be automatically converted are clearly marked so that they can be manually converted to complete the migration. You can review the database migration assessment report which summarizes all of the action items for schemas that can't be converted automatically to the target engine. Once schema conversion is complete, SCT can help migrate data from a range of data warehouses to Amazon Redshift using built-in data migration agents.

The following screenshot shows the list of action items shown by the database migration assessment report in SCT.



The list of action items shown by the database migration assessment report in SCT

Determine the target

AWS DMS can use many of the most popular databases as a target for data replication. For a comprehensive list of valid targets, refer to Targets for AWS DMS. It's important to be aware of the latency requirements of your organization and the business use-cases while choosing a target for migrating your data. For example, Amazon Redshift is optimized for online analytical processing (OLAP), meaning it is used to aggregate large quantities of data across long time periods and may not require real-time updates. To ensure minimal disruption to those processes, consider loading in batches with larger latencies.

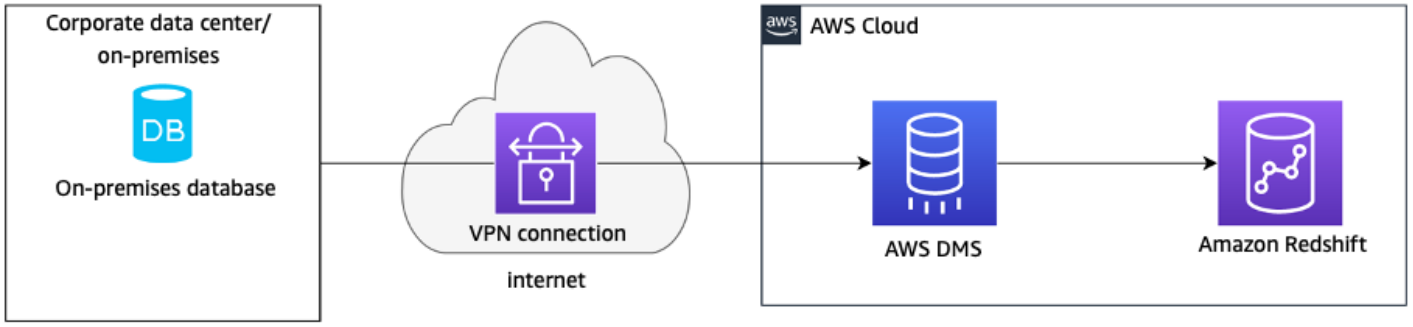
For real-time updates, consider using online transaction processing (OLTP) systems such as Amazon Relational Database Service (Amazon RDS) as a target.

Connectivity options

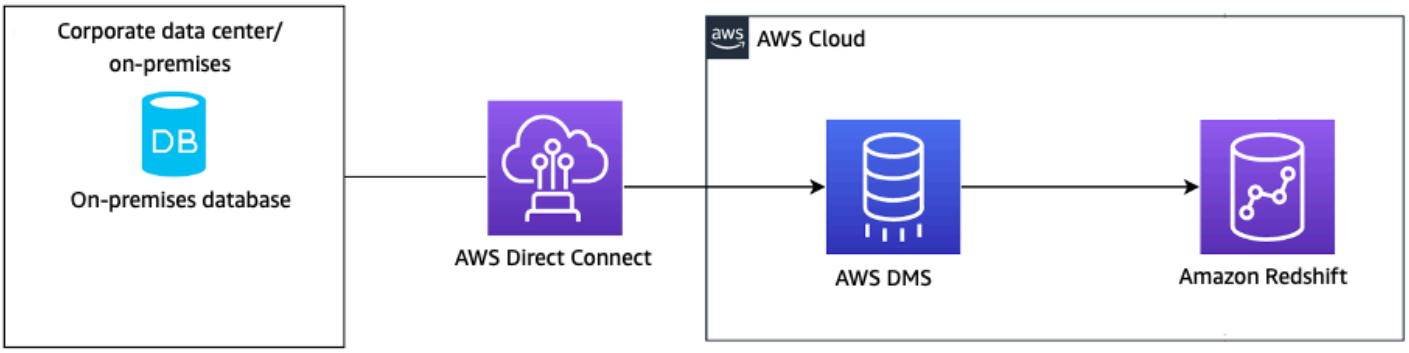
It is important to validate and monitor connectivity between the source database and AWS DMS, and between AWS DMS to the target, to avoid any performance issues. Connectivity options vary depending on where the different components are deployed.

The location of the AWS DMS replication instances also varies based on the target chosen. For example, if an Amazon Redshift cluster is used as a target, it must be in the same AWS account and same AWS Region as the replication instance. If the source endpoint, the replication instance, and the target endpoint are in the same VPC, traffic flows within the VPC. If different VPCs are used for deployment, you can link these VPCs together by using VPC peering or AWS Transit Gateway. If the source endpoint is an on-premises database in your corporate data center, it can be connected to AWS resources using AWS Direct Connect, a VPN connection, or even via the public internet.

However, it is highly recommended to choose the right options to connect resources, to avoid any latency issues. AWS Direct Connect provides the shortest path to AWS resources, because it is a dedicated connection between your corporate data centers and AWS. While in transit, your network traffic remains on the AWS global network and never goes over the internet. This reduces the chance of hitting bottlenecks or unexpected increases in latency when compared to using VPN or the public internet.



VPN connection going over the public internet



Using AWS Direct Connect as a dedicated connection between an on-premises database and AWS

Full load (initial load) phase

Full load is used for loading all the data that is available in the tables in the source database to the target, and is typically used for one-time data migrations. During a full load task, AWS DMS performs a full table scan for each of the source tables in parallel. Because the entire data is being loaded from the tables, it is important to optimize the full load settings to accelerate the migration of large or partitioned tables. These configurations may vary depending on the type of target chosen. AWS SCT has agents which can help with the full load to Amazon Redshift. During a full load migration task, you can accelerate the migration of large or partitioned tables by splitting the table into segments and loading the segments in-parallel in the same migration task. We will discuss more about table segmentation later in the whitepaper.

CDC phase (insert, update, delete)

If an ongoing data migration is required from the source database to the target, you can leverage CDC functionality provided by AWS DMS to capture incremental changes. This process works by

collecting changes to the database logs using the database engine's native API. If the CDC process is not tuned accurately, your workloads may face high source or target latencies.

Storage

It is a good idea to be aware of amount of data that needs to be migrated from the source database to plan accordingly for the storage required by AWS DMS replication instances as well as by Amazon Redshift (target).

For example, if you are using Oracle database as a source, be aware of the log switches and the amount of redo logs generated per day by the database to evaluate the storage needs.

Date	log switches	~Approx Redo Per Day (MB)
3/22/21 0:00	361	184832
3/21/21 0:00	235	120320
3/20/21 0:00	254	130048
3/19/21 0:00	349	178688
3/18/21 0:00	410	209920
3/17/21 0:00	233	119296
3/16/21 0:00	235	120320
3/15/21 0:00	238	121856
3/14/21 0:00	212	108544
3/13/21 0:00	208	106496

Example showing the number of log switches and redo generation per day

In AWS DMS, most of the processing happens in memory. However, large transactions might require some buffering on disk. Cached transactions and log files are also written to disk. If your source system is busy or takes large transactions, you might need to increase your storage on DMS as well as target systems. If you're running multiple tasks on the replication server, you might also need a storage increase.

If you have detailed debug logging turned on for DMS, ensure that you turn it off when you don't need it, because detailed debug logging requires a large amount of storage space and can potentially fill up the disk, leading to DMS task failures. For more information about storage issues, refer to Why is my AWS DMS replication DB instance in the storage-full status?.

Based on the amount of data that needs to be migrated into your Amazon Redshift cluster, you can choose the right instance types. You can use the Amazon Redshift sizing calculator (sign-in required) available in the Amazon Redshift console, which estimates your cluster configuration based on data size and data access frequency.

Networking bandwidth

If you are using VPN tunnel between your on-premises source database and AWS DMS, ensure that the bandwidth is sufficient for your workload. Each AWS Site-to-Site VPN connection has two tunnels and each tunnel supports a maximum throughput of up to 1.25 Gbps. To be highly available and to have no disruptions, ensure that you utilize both the VPN tunnels. If you are using AWS Direct Connect, you can either have a dedicated connection supporting one Gbps, 10 Gbps, 100 Gbps, or a hosted connection which is sourced from AWS Direct Connect Partners that have network links between themselves and AWS.

To optimize performance, ensure that your network throughput is fast. When you address network performance, get a network sniffer trace to analyze network performance. In addition, use network monitoring tools to identify issues with bandwidth performance limits being exceeded on the network, the client, or the server. You can determine if issues are related to throttling, bandwidth, utilization, or any other configuration-related issues. There are many utilities such as netstat, traceroute, and ipconfig, and port scanners available for troubleshooting or checking on the connection to look up anomalies. If possible, engage network or systems engineers early in the planning process.

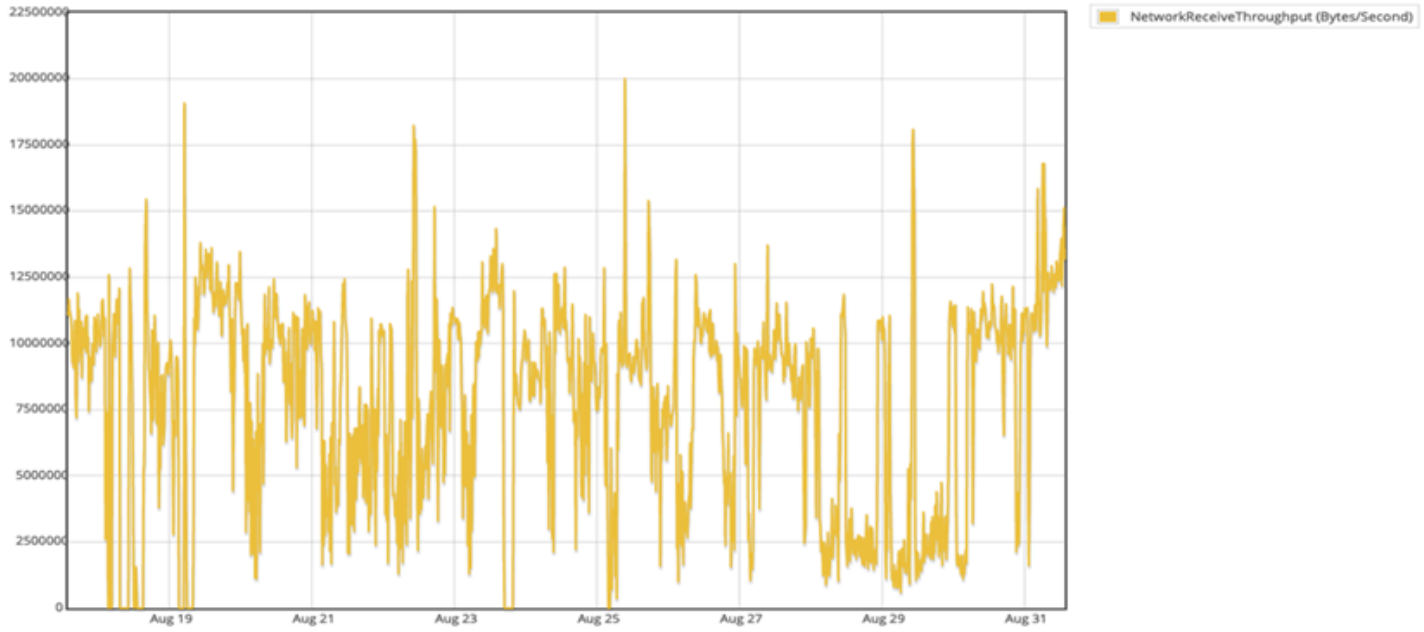
Another simple test for finding out the networking speed is done by transferring a file from your source database to an Amazon Elastic Compute Cloud (Amazon EC2) instance in AWS within the same virtual private cloud (VPC) as the AWS DMS service, and noting the amount of time the

process takes. The transfer speed can be calculated by dividing the amount of data transferred with the time taken to transfer:

Source: ***172.26.246.70***
Destination: ***10.0.3.108***
File: ***agent.zip 100% transferred***
File size: ***705MB***
Time taken to transfer the file: ***38 min 16 sec***
Between: ***12:01pm -12:41 pm PST***
Speed: *Size of the file (KB) / Time to transfer (seconds) =* **314.5 KB/s**

Source database statistics

Verify the `NetworkReceiveThroughput` metric on DMS end indicating the incoming traffic on replication instance.



An example of average network traffic of 11 MB/s on the DMS instance

Pricing for migration

AWS DMS pricing varies based on the type of replication instances chosen and any additional log storage. Each database migration instance includes storage sufficient for swap space, replication logs, and data cache for most replications and inbound data transfer is free. All data transfer into AWS DMS is free, and data transferred between AWS DMS and Amazon RDS, Amazon Redshift and Amazon EC2 instances in the same Availability Zone also is free. Standard AWS data transfer rates apply when you migrate your source database to a target database in a different Availability Zone, Region, or outside of AWS. Standard AWS data transfer rates are listed on the EC2 instance pricing page.

Preparing source database

AWS DMS can use many of the most popular data engines as a source for data replication. The database source can be a self-managed engine running on an Amazon EC2 instance or an on-premises database. It can also be a data source on an AWS service such as Amazon RDS or Amazon Simple Storage Service (Amazon S3). For a comprehensive list of valid sources, refer to Sources for AWS DMS.

Apart from the table sizes and types of objects discussed in the previous *Assess Source Database* section, there are different configuration steps used to migrate data from each of the source databases. However, the following steps list the basic actions that are typically applicable to all the data sources:

- Ensure that your source database version is among the list of supported versions by AWS DMS.
- Create a source endpoint for your database.
- Create a user with appropriate permissions for AWS DMS to access the source database.
- Enable logging on the database (supplemental, binary, write-ahead, and so on) depending on the source database.
- You can use Secure Socket Layers (SSL) to encrypt connections between your source endpoint and the replication instance.
- Understand the Recovery Time Objective (RTO) and Recovery Point Objective (RPO) goals of your organization. Based on these goals, you can decide on the acceptable lag values. For example, if the source from where you are capturing the data is not the primary database instance, ensure that you are factoring in its data freshness threshold while calculating the time it takes for the data to make its way into the target.

  For more information on specific configuration steps for various data sources, refer to the Sources for data migration.

Optimizing AWS DMS performance

After preparing the source database, you can set up the AWS DMS service to create a task to perform one-time migration of data or to perform change data capture on an ongoing basis (full load, full load plus CDC or CDC only task). You will need to configure both the source and target endpoints. The only requirement to use AWS DMS is that one of your endpoints must be on an AWS service. You can't use AWS DMS to migrate from an on-premises database to another on-premises database.

Apart from the source database configuration, the networking between source database and AWS resources, and the target configuration, it is important to monitor the AWS DMS performance and ensure that you have chosen the right replication instance type to prevent any latency issues. The key bottlenecks are normally CPU, memory and network throughput of the selected replication instance. You will need a review of the CloudWatch metrics such as SwapUsage, CPUUtlization, NetworkTransmitThroughput, and NetworkReceiveThroughput (to name a few) to ensure the replication instance is not under resource pressure. Ideally, swap usage should be zero. You can find more replication instance metrics in the AWS DMS documentation. Choosing the appropriate replication instance depends on several factors of your use case. Refer to Choosing the best size for a replication instance for more information.

It is important to take into account the data volume while selecting the replication instance. The amount of parallelism is determined by CPU utilization. This also helps determines the number of tasks you can possibly have. Large objects (LOBs) also impact performance and memory consumption. Load frequency and transactions per second (TPS) influence memory usage. A high number of TPS or data manipulation language (DML) activities leads to high usage of memory. This happens because AWS DMS caches the changes until they are applied to the target. During CDC, this leads to swapping (writing to the physical disk due to memory overflow), which causes latency.

To speed up full load and also improve the CDC process, it’s a good practice to create separate AWS DMS tasks for tables which have a huge number of records or high volume of DML activities to prevent data migration from other smaller tables from slowing down. Note that too many DMS tasks will result in more concurrent transactions on your Redshift Datawarehouse which could impact performance of the data migration as well as impact other workloads using the Redshift cluster.

During a full load migration task, you can accelerate the migration of large or partitioned tables by splitting the table into segments and loading these segments in-parallel into the target. This

can be accomplished by creating a rule of type table-settings with the parallel-load option in the migration task. Note that parallel load for use with table-setting rules are supported for some target endpoints. Using this parallel-load process, you can first have multiple threads unload multiple tables and views in parallel from the source endpoint. You can then have multiple threads migrate and load the same tables and views in parallel to the target endpoint. With this option, AWS DMS splits a full-load task into threads, with each table segment allocated to its own thread.

For some database engines, you can leverage existing partitions or sub-partitions itself to perform the parallel load. For tables with no partitions, you have the option of creating logical segments by defining boundaries explicitly in the form of ranges which can be loaded in parallel to improve the performance and to reduce the amount of time it takes for the full load.

For more examples, refer to Table and collection settings rules and operations. To indicate the maximum number of tables to load in parallel during the full load, set the `MaxFullLoadSubTasks` option.

For example, we have seen scenarios where a full load for a table with 1.2 billion records on source (Oracle) took 48 hours. The Parallel-load feature was used and the table was split into logical partitions by specifying ranges of column values with 0.5 million values in each segment, as shown in the following code snippet.

Ensure that the columns you choose to partition the data is either the primary key of the table, or a partition key or are indexed columns to significantly improve performance.

```
{
   "rule-type":"table-settings",
   "rule-id":"4",
   "rule-name":"4",
   "object-locator":{
      "schema-name":"*****",
      "table-name":"*****"
   },
   "parallel-load":{
      "type":"ranges",
      "columns":[
         "PURCHASE_ORDERS_ID"
      ],
      "boundaries":[
         [
            "500000"
         ],
```

```
            [
               "1000000"
            ],
            [
               "1500000"
            ],
            [
               "2000000"
            ],
                  .
                  .  // more segments
                  .
            [
               "22500000"
            ]
         ]
      }
}
```

Along with the previous configuration, `MaxFullLoadSubTasks` was also increased from default eight to 44 in the **Full Load** task settings. With these settings, the full load time reduced from 48 hours to two hours 38 minutes.

Schema name	Table	Load state	Elapsed load time	Inserts	Deletes	Updates	DDLs	Full load rows
[redacted]	[redacted]	Table completed	2 h 38 m 3 s	0	0	0	0	1,204,462,028

Full load of a table having 1.2 billion rows completed in two hours 38 min

AWS periodically releases new versions of the AWS DMS replication engine software, with new features and fixes for known issues. Each version of the replication engine software has its own version number, to distinguish it from other versions. It is recommended to use latest version of AWS DMS service to leverage performance improvements as well as to get access to the new functionality.

Preparing Amazon Redshift as a target for AWS DMS

For the purpose of this whitepaper, assume Amazon Redshift is a target for the AWS DMS service to migrate data from any source database. Refer to the complete list of AWS DMS targets: Targets for AWS DMS.

As mentioned before, the Amazon Redshift cluster must be in the same AWS account and the same AWS Region as the replication instance. This is because, during the data migration, AWS DMS copies data in form of .csv files over to an Amazon S3 bucket on your account before moving it to the tables in AWS Redshift data warehouse. For more information, refer to Prerequisites for using an Amazon Redshift database as a target for AWS Database Migration Service.

During the CDC phase of a task, AWS DMS uses a single thread to read change data from the source and apply changes to the target. Because the thread can handle only a certain number of transactions at a time, depending on the rate of changes on the source, sometimes the thread can't keep the target in sync with the source. This happens more often when Amazon Redshift is the target for AWS DMS because, batch transactions are performed more efficiently in OLAP engines. Amazon Redshift is optimized to run complex analytic queries and is optimized for aggregations on large data sets. Amazon Redshift performance can be affected when running transactional changes one-by-one from an OLTP database. You may see high target latencies during the duration of the time where AWS DMS runs transactions in a one-by-one mode.

Also, the level of parallelism applied depends on the correlation between the total batch size and the maximum file size used to transfer data. When using multithreaded CDC task settings with a Redshift target, benefits are gained when batch size is large in relation to the maximum file size.

Based on the information above, you can use the following extra connection attributes and task settings to tune for optimal CDC performance.

```
// Redshift endpoint setting

maxFileSize=250000;dateFormat=auto;timeFormat=auto;acceptAnyDate=
true;fileTransferUploadStreams=20;compUpdate=false
```

maxFileSize (default value 32MB) specifies the maximum size (in KB) of any .csv file used to transfer data to Amazon Redshift. Dateformat by default is “YYYY-MM-DD”. Using “auto” for the dateformat string helps recognize several different formats. Similarly, using “auto” for timeformat field recognizes different formats. fileTransferUploadStreams specifies the number of threads used

to upload a single file. compUpdate=false disables the automatic compression and existing column encodings aren't changed.

```
// AWS DMS Task settings

BatchApplyEnabled= true
BatchSplitSize= 0
BatchApplyTimeoutMax = 1800
BatchApplyTimeoutMin = 1800
BatchApplyMemoryLimit = 1024
ParallelApplyThreads = 32
ParallelApplyBufferSize = 100
```

Regarding the Batch Apply Task settings, you enable the Batch Apply mode, and set the value for the amount of time DMS has to wait between each application of batch changes via the `BatchApplyTimeoutMin` and `BatchApplyTimeoutMax` settings. It is important to determine the replication "requirements" and to set this value accordingly. You can set these parameters to a higher value if you need the data to be refreshed more than once every 30 minutes. You also configure the `BatchSplitSize` as 0, which implies that there is no limit on the maximum number of changes applied in a single batch. Increase the `BatchApplyMemoryLimit` to 1024 so that more records can be processed in each commit.

`ParallelApplyThreads` specifies the number of concurrent threads that AWS DMS uses during a CDC load to push data records to the Amazon Redshift target endpoint. `ParallelApplyBufferSize` specifies the maximum data record requests while using parallel apply threads with Redshift target.

Using the settings above, customers with heavy transactional workloads can benefit by their having an unlimited batch buffer getting filling up to the extent possible in 1800 seconds, utilizing 32 parallel threads with a 250 MB maximum file size.

Regarding the memory allocation guidelines for CDC, when `ParallelApplyThreads` > 0, configure the memory parameters based on the `maxFileSize` value mentioned previously. For example, if you are using ten parallel apply threads with `maxFileSize` as default, AWS DMS will be generating ten CSV files each of 32MB size in memory. Therefore, memory consumption will be 32MB * ten = 320MB. As a result, based on the available memory on the DMS replication instance, it is a good idea to allocate higher value for `BatchApplyMemoryLimit` in the batch settings (at least 320MB in this example) to enable all the ten threads to create full 32MB files in memory rather than disk, thereby improving performance.

For speeding up **full load** process, apart from creating table segments which can be loaded in parallel as mentioned in the section above, you can also define `ParallelLoadThreads`, which specifies the number of concurrent threads that AWS DMS uses during a full load, to push data records to an Amazon Redshift target endpoint. When using parallel load, depending on the number of parallel threads configured, the memory requirement changes. The formula below can be useful for computing the amount of memory required by DMS replication instance.

Minimum memory consumed:

```
Number of MaxFullLoadSubTasks * number of ParallelLoadThreads * maxfilesize
```

Recommendation for DMS replication instance memory:

```
2 * Minimum memory consumed
```

For example:

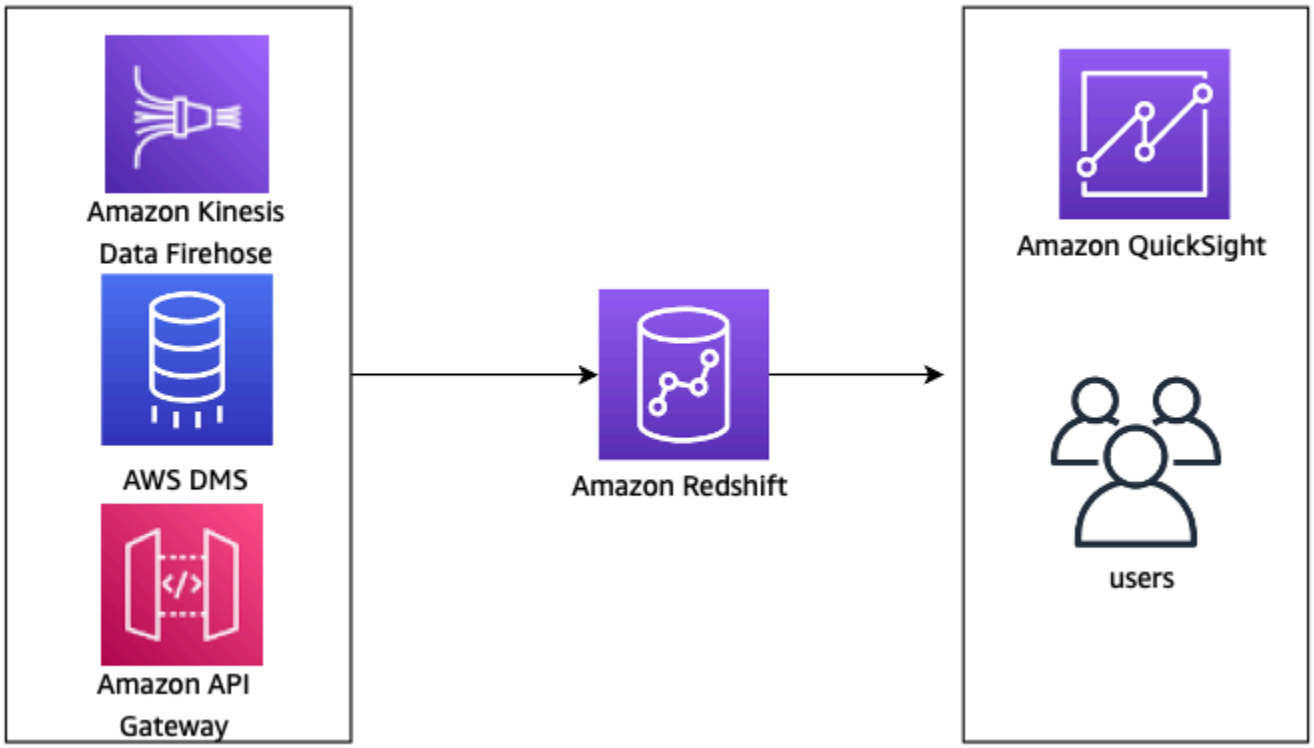
```
8 (MaxFullLoadSubTasks) *32 (ParallelLoadThreads) * 250MB = 64GB (minimum)

64 * 2 = 128 GB (recommended)
```

When eight threads are used to pull data from the source (`MaxFullLoadSubTasks`) and 32 threads on DMS end (`ParallelLoadThreads`) are configured to create one .csv file per thread for loading into Amazon Redshift, with the size of each csv file set as 250MB (`maxfilesize`), the minimum memory consumed by DMS replication instance is 64GB. It is recommended to select a DMS instance with 128GB memory in this scenario.

If memory on the replication instance is not sufficient, you can set `enableParallelBatchInMemoryCSVFiles` to false for the files to be written to disk. Note that, if disk is used, there may be a performance drop of about 20% when compared to using memory.

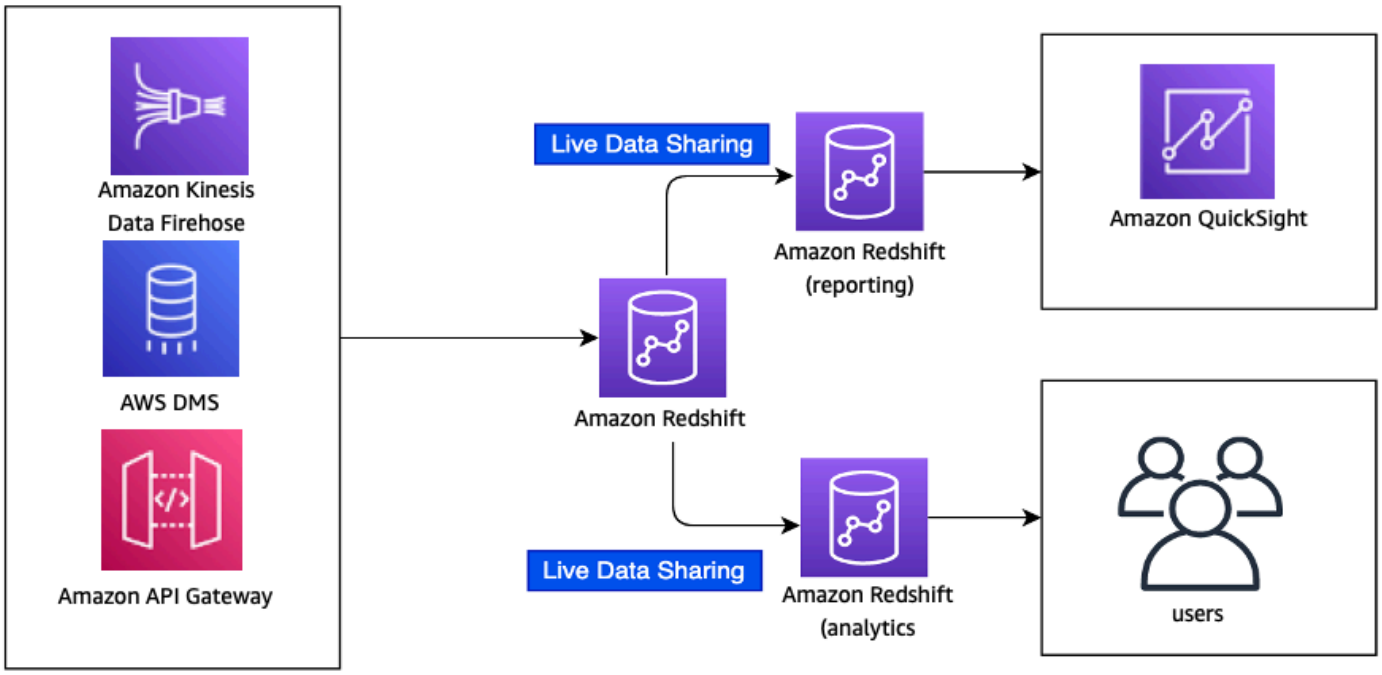
The other aspect which can improve the performance of data loading into Amazon Redshift is the resource utilization of the Amazon Redshift cluster. Monitor for memory and CPU utilization metrics on the Amazon Redshift cluster. It is recommended to identify outlier disk spill queries, which will cause concurrent ETL queries to slow down. You can use this query to identify Workload Management (WLM) queues where the value of query_temp_block_disk is high, indicating

intermediate results being written to disk due to insufficient memory. It's evident that most performance bottlenecks on the cluster are disk related, so reducing commit overhead and paging to disk by rebalancing WLM will give the cluster a throughput increase. You can use AutoWLM instead of customWLM to ensure that memory is allocated automatically to the Amazon Redshift queues based on usage.



A single Amazon Redshift cluster having multiple workloads

If the Redshift cluster is used by multiple workloads, and is affecting the AWS DMS performance, it's a good idea to separate out the workloads by using Amazon Redshift data sharing functionality.



Separating workloads and improving Amazon Redshift performance using data sharing functionality

Another aspect to consider is when you load data into a table, Amazon Redshift distributes the rows of the table to each of the compute nodes according to the table's distribution style. It's important for large tables with many updates/deletes to choose a distribution key that matches the column of the primary key with the highest cardinality to make it less resource intensive.

AWS recommends pre-creating the target tables on Amazon Redshift with the right distribution keys. If you allow AWS DMS to create the target tables, it does not currently choose a distribution key.

- Ensure the target tables have Primary keys.
- Ensure you have distribution keys created for the target tables. Refer to Choose the best distribution style for more information.

Testing the migration

It is highly recommended to run tests in your test or staging environments to determine how the source, networking, AWS DMS, and target components work under load. Testing the system with three-four times the load in production environments and stabilizing it ensures that, in case of unexpected peaks, the infrastructure will be able to handle the load without causing delays.

You can make use of tools that can help clone the traffic heading to the production database and replay it in the test environments. Doing prior testing will help you discover issues with your environment in early phases of your database migration.

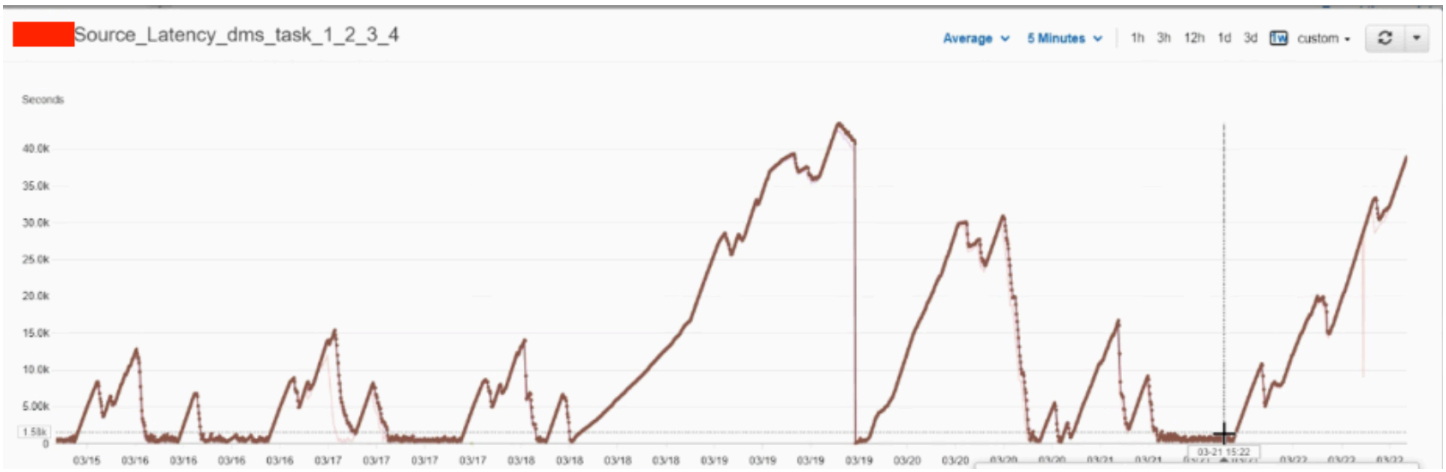
Start testing with a small amount of load, and then scale it up to run a full-scale test migration to measure whether AWS DMS can handle the throughput of your database over your network. During this time, AWS recommends benchmarking and optimizing your initial full load and ongoing replication. Doing this can help you to understand your network latency and gauge overall performance.

At this point, you also have an opportunity to understand your data profile and how large your database is, including the following:

- How many tables are large, medium, and small in size.
- How AWS DMS handles data type and character-set conversions.
- How many tables having large object (LOB) columns.
- How long it takes to run a test migration.

Troubleshooting issues

The most common issue seen with migration of data into Amazon Redshift clusters using AWS DMS is high source and target latencies.



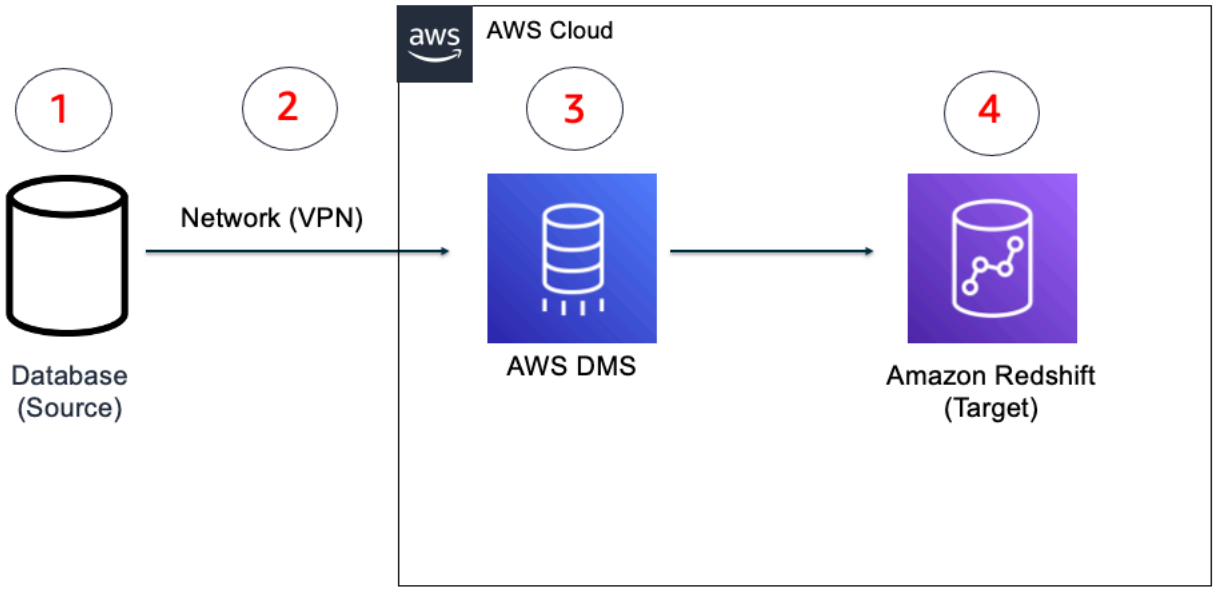
A screenshot showing high DMS CDC source latency in seconds

High `CDCLatencySource` means that the process of capturing changes from the source is delayed. High `CDCLatencyTarget` means that the process of applying the change events to the target is delayed. You can use Amazon CloudWatch metrics to monitor your replication task's statistics.

If you are unsure about why the latency issues are seen in your environment, as a starting step for troubleshooting these issues, understand what the acceptable latencies are for your business. If both `CDCLatencySource` and `CDCLatencyTarget` are higher than the acceptable latency, investigate `CDCLatencySource` first, because the target latency is usually the same as the source. If `CDCLatencyTarget` is significantly higher than the source, you can start troubleshooting by reviewing the target configuration.

You can use a divide and conquer approach to isolate and rule out the different aspects that can cause the source latencies. The four different areas that you can focus on are:

- The source database
- Networking
- AWS DMS configuration
- The target, which is the Amazon Redshift cluster



Using the Divide and Conquer approach for resolving DMS latency issues

Identify, with the help of your DBAs, whether there are any issues with your source database with respect to resource consumption or storage. If the data is being read from a standby database instance, ensure that the standby instance does not have any issues with respect to getting the data from the primary database instance.

Spikes in source latency may also be attributed to a sudden increase in the amount of data being migrated from the source database to AWS DMS service. It is important to monitor the amount of data generated by the source database to ensure that there is no unexpected activity happening on the source that could be contributing to the increasing latency.

Determine what type of connectivity option you are using to connect the source database with the AWS DMS service; for example, VPN or AWS Direct Connect. You can perform networking tests with the help of the AWS Support team by opening up support cases or by using other networking tools as stated previously. Check for any network connectivity issues by going through the AWS DMS logs. If you are using VPN connection, you can monitor VPN tunnels using Amazon CloudWatch, which collects and processes raw data from the VPN service into readable, near real-time metrics. You can check the VPN tunnel metrics to monitor the tunnel state and `DataIn/DataOut` metrics.

Monitor the bandwidth usage and try raising it if it is not sufficient. Ensure that you are using two VPN tunnels for redundancy and high availability. Because VPN connection has an upper limit of

1.25. Gbps, verify whether multiple workloads are using the VPN tunnel. You can find out this information by contacting your networking team within the organization. Because it is AWS Site-to-Site VPN, traffic cannot be differentiated just to identify data going through the VPN tunnel for the purpose of AWS DMS replication.

For a dedicated connection between the source and target endpoints, you can use a Direct Connect connection. This can significantly improve the networking performance.

Verify the AWS DMS replication instance metrics and the version being used. Upgrade to higher versions of AWS DMS to avail the latest fixes and newer functionality. If the full load happens very slowly, taking several hours, verify whether a parallel-load feature is being used, where table partitions are loaded in parallel. For non-partitioned tables, you can create logical segmentations to parallelize the data load. Ensure that you are using AWS DMS Batch Apply settings with Amazon Redshift to speed up the CDC replication phase.

From the Amazon Redshift end, ensure that a dedicated queue is configured for the data from AWS DMS. This will prevent AWS DMS batches (such as `COPY` commands) from queuing behind other queries. You can use CloudWatch metrics such as `WLMQueueWaitTime` and `WLMQueueLength` to get an idea on how many queries are waiting to enter the Amazon Redshift WLM queues. For more metrics, refer to Monitoring Amazon Redshift using CloudWatch metrics.

You can also view the current state of queries being tracked by WLM by querying the STV_WLM_QUERY_STATE table. Check for swap and disk usage to identify disk spills by querying the `SVL_QUERY_SUMMARY` table, and verify whether the `is_diskbased` field value is true. Verify the memory consumption and WLM memory allocation strategy by referring to How do I use and manage Amazon Redshift WLM memory allocation?.

If the Amazon Redshift cluster is being used by multiple workloads and performance is being affected, isolate the workloads by splitting up the Amazon Redshift clusters into multiple clusters using data sharing functionality.

Ensure that you enable notifications for each of the components in the data migration pipeline. You can use Amazon EventBridge to send out notifications when events occur. For example, you can configure Amazon EventBridge event rules to notify appropriate teams when AWS DMS replication instances are created or deleted.

Conclusion

Data migration is supported from several databases to Amazon Redshift data warehouse within AWS using the AWS DMS. Once the data is in Amazon Redshift, it can be used for creating analytical sales forecasts, business intelligence (BI) dashboards, predicting customer churn, reporting, and for several different use-cases impacting business outcomes. If the migration best practices are not followed, this can lead to severe source and target latency issues which can impact or stall the data migration efforts, specifically if the data migration is happening on an ongoing basis. It is crucial to understand the different components of the data migration process and work on optimizing each of them to deliver optimal performance.

Contributors

Contributors to this document include:

- Sindhura Palakodety, Solutions Architect, Amazon Web Services
- Eli Doe, DB Migration Specialist SA, Amazon Web Services
- Rajiv Gupta, Analytics Specialist SA Manager, Amazon Web Services

Further reading

For additional information, refer to:

- Best practices for AWS Database Migration Service
- Using an Amazon Redshift database as a target for AWS Database Migration Service
- Sharing data across clusters in Amazon Redshift
- Change processing tuning settings

- Migrate from Oracle to Amazon Redshift (Oracle as Source and Amazon Redshift as Target)

Document revisions

Date	Description
July 15, 2022	First Publication

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.