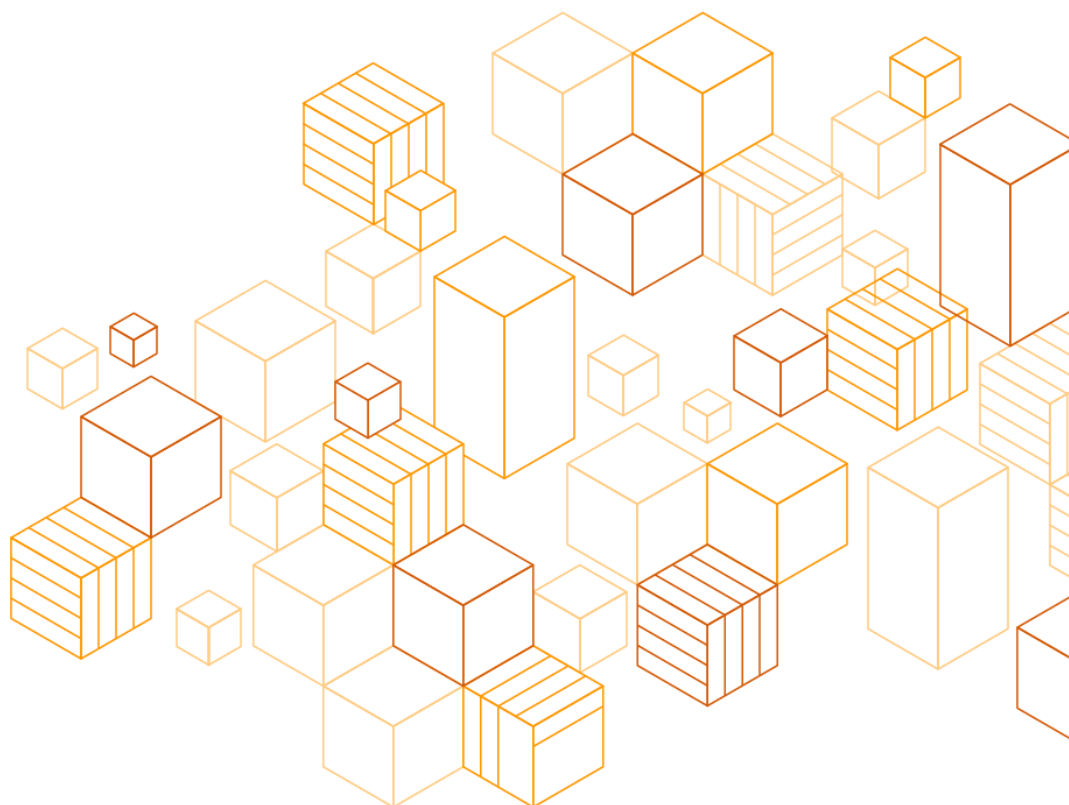


Oracle to PostgreSQL CDC Monitoring with AWS Database Migration Service

Migration Guide

June 2020



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.

© 2020 Amazon Web Services, Inc. or its affiliates. All rights reserved.

Contents

| | |
|------------------------------|----|
| Overview | 1 |
| Components of AWS DMS..... | 1 |
| Replication Instance..... | 1 |
| Source Endpoint..... | 8 |
| Target Endpoint..... | 10 |
| Task | 11 |
| When to Use DMS Metrics..... | 19 |
| Use case | 21 |
| Conclusion | 22 |
| Contributors | 22 |
| Additional Resources | 22 |
| Document Revisions..... | 23 |

About This Guide

This guide outlines the metrics to review when you move a database from Oracle to PostgreSQL using AWS Database Migration Service change data capture (CDC). It includes how to interpret the metric data at different steps of the process.

Understanding these metrics allows you to more quickly resolve issues when bottleneck points occur in data migration.

Overview

AWS Database Migration Service (AWS DMS) makes it easy to move projects of multiple sizes from on-premises databases to Amazon RDS, Amazon Redshift, and NoSQL database services. AWS DMS helps you change schemas, load full table records, and support on-going change data capture (CDC) processing during the migration project and production environment operations.

This guide covers CDC processing, including how to read metrics and how to handle abnormal situations indicated by metrics.

Components of AWS DMS

Before we can get into reading the metrics that AWS DMS provides, let's review the components that make it up. AWS DMS consists of four major components: replication instance (RI), source endpoint, target endpoint, and task. These components and their associated metrics are detailed in the following sections.

Replication Instance

The **replication instance** is an Amazon Elastic Compute Cloud (Amazon EC2) instance that contains the DMS engine. The replication instance has its own resources, such as network, CPU, storage, and memory. Each resource has its own limitations that can become bottlenecks in CDC task processing.

The **source database** is the source of all table records and CDC records. The source database also has physical resources, such as storage devices (SAN, DAS, NAS) and CPU/memory boards. Each resource has the potential to be a bottleneck point in CDC processing. In many cases, the network resource is the most vulnerable point in the pipeline. Often, the source database doesn't show the throttle power to push CDC records to the DMS replication instance through the source endpoint. Strictly speaking, the source database doesn't push the CDC records to the replication instance. Instead, the replication instance pulls the CDC records from the source database.

The [Source Endpoint](#) section, shows the related task settings and details how to improve pulling power from the replication instance and a task.

The **target database** is the transaction target for DMS. You can perform transactions with a range of commands, such as DML SQLs (INSERT, UPDATE, DELETE), or batch

operations (COPY). You can perform transactions with a range of commands, such as DML SQLs (INSERT, UPDATE, DELETE), or batch operations (COPY) on the source database. However, you can't choose which DML SQL commands are used in the DMS transactions on the target database. But sometimes, you must know which command can be used during full-loading operation. For example, the DMS PostgreSQL engine uses the COPY command with a .csv file format. If you set a table or database character, set it to UTF-8 but the source records have non-UTF8 format such as EUC-KR/EUC-JP/CP949, the COPY command can't handle the source records. So, you would see invalid character errors during full-loading. If you know that the COPY command will be used under bulk loading, you can choose the right solutions to overcome this error (for example, deleting invalid characters from the source database or choosing the right character set).

AWS DMS CDC is similar to data pipelining from source to sink. Low CDC performance indicates some bottlenecks in the pipeline. Amazon CloudWatch Metrics and logs show if there is a bottleneck in the pipeline and also where the bottleneck is located in any component of AWS DMS replication instance, source database, target database, and network.

Replication Instance Metrics

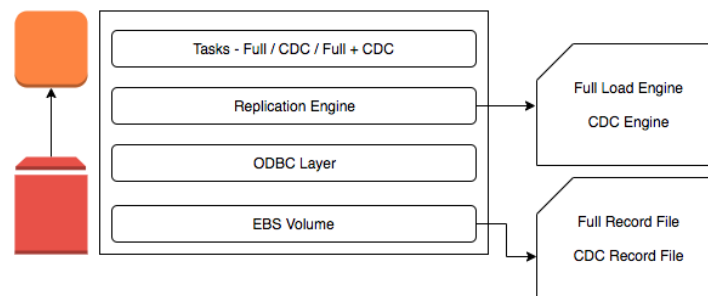


Figure 1:Replication instance diagram

A replication instance is based on an Amazon EC2 instance with an Amazon Elastic Block Store (EBS) volume size of your choosing. One Amazon EBS volume is a full record temporary file used with batch commands (load, copy, etc.) and the other is for CDC temporary records. The following sections detail the replication instance metrics.

CPUUtilization

AWS DMS currently supports the T2, C4, and R4 Amazon EC2 instance classes for replication instances:

- The T2 instance classes are low-cost standard instances designed to provide a baseline level of CPU performance with the ability to burst above the baseline. They are suitable for developing, configuring, and testing your database migration process. They also work well for periodic data migration tasks that can benefit from the CPU burst capability.
- The C4 instance classes are designed to deliver the highest level of processor performance for computer-intensive workloads. They achieve significantly higher packet per second (PPS) performance, lower network jitter, and lower network latency. AWS DMS can be CPU-intensive, especially when performing heterogeneous migrations and replications such as migrating from Oracle to PostgreSQL. C4 instances can be a good choice for these situations.
- The R4 instance classes are optimized for memory-intensive workloads. Ongoing migrations or replications of high-throughput transaction systems using DMS can, at times, consume large amounts of CPU and memory. R4 instances include more memory per vCPU.

If CPU utilization is high (reached 70% or 80%), change the replication instance to a larger size or replace the instance type with a compute-intensive instance type, such as C4.

If you see CPU utilization fluctuation, before you change the instance type, check the `FreeableMemory` and `SwapUsage` metrics. When the replication engine uses more real physical memory, the `FreeableMemory` metric would be approaching to zero. When this occurs, the replication engine uses swap memory (You can check it in the `SwapUsage` metric), which can degrade overall performance. If this situation occurs, change the RI type to a memory-intensive instance, such as R4 (Figure 2).



Figure 2: High CPU utilization with low freeable memory

FreeStorageSpace

Figure 1 shows that the local storage is used for full-loading temporary files and CDC records file. The total storage size should be larger than the sum of these local storage locations.

The size of CDC records is usually estimated by the source database transaction log size (REDO log, WAL log, binlog) during full-loading time. As the total records of target tables increases, the total storage of replication instance should also increase.

In the on-going CDC tasks, free storage is used only for CDC records and engine/task logs. However, disk storage is not a good choice for on-going CDC. All records should be processed in the memory not on the disk.

In the following figure, the `FreeStorageSpace` metric shows almost no usage of disk.

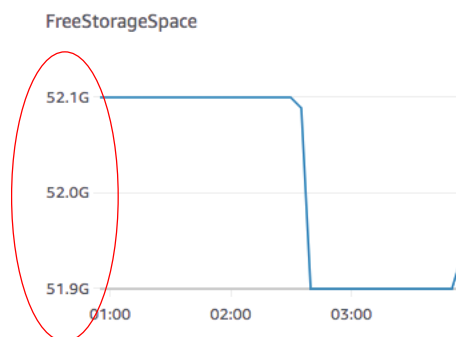


Figure 3: `FreeStorageSpace` metric with low usage

FreeableMemory

Memory is used for various purposes such as the OS kernel, AWS DMS engine, CDC records, and unloaded records from sources. Memory is faster than disk, so we should guide the replication engine to use memory instead of disk.

Memory usage that exceeds the physical memory size increases the `SwapUsage` value. The swap memory prevents out of memory (OOM) errors but it can compromise replication performance. Avoid this scenario, if possible. Every task allocates each memory page to replicate records, so you should estimate total memory by the sum of all tasks.

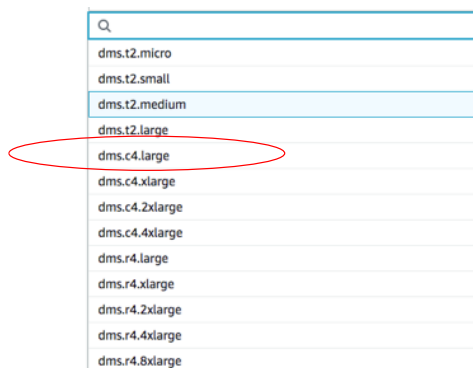
WriteIOPS/ReadIOPS/WriteThroughput/ReadThroughput

If you use **full-loading and on-going CDC**, the replication engine uses disk to store CDC records and unloaded records from source database.

DMS uses gp2 EBS volume, so you can see when the replication engine hits the limit of EBS IOPS and bandwidth. Then, you should check the limit of two resources. First is the limit of GP2 EBS volume. The IOPS limit of GP2 depends on the size of EBS volume. You can see the more specific explanation in [Amazon EBS features](#). The second resource is the limit of EC2 itself, which you can find on the [Amazon EBS-optimized instances site](#).

If the DMS replication engine tried to use more IOPS than the limit of EC2, it could produce a bottleneck.

The following list is the current supported EC2 instance types in the replication instance.



| |
|----------------|
| Q |
| dms.t2.micro |
| dms.t2.small |
| dms.t2.medium |
| dms.t2.large |
| dms.c4.large |
| dms.c4.xlarge |
| dms.c4.2xlarge |
| dms.c4.4xlarge |
| dms.r4.large |
| dms.r4.xlarge |
| dms.r4.2xlarge |
| dms.r4.4xlarge |
| dms.r4.8xlarge |

Figure 4. DMS replication instance types

| | | | | |
|------------|-----|-------|-------|--------|
| c4.large | Yes | 500 | 62.5 | 4,000 |
| c4.xlarge | Yes | 750 | 93.75 | 6,000 |
| c4.2xlarge | Yes | 1,000 | 125 | 8,000 |
| c4.4xlarge | Yes | 2,000 | 250 | 16,000 |
| c4.8xlarge | Yes | 4,000 | 500 | 32,000 |

Figure 5. IOPS/Bandwidth limits (in EBS-optimized instances)

For example, dms.c4.large has 500 Mbps (62.5 MB) maximum bandwidth and 4000 IOPS. So, you can set at least 1.3 TB as DMS storage to gain full performance in IOPS ($1300 \text{ GiB} * 3 \text{ IOPS/GiB} = 3900 \text{ IOPS}$). If you see the maximum IOPS in this metric indicate the limit of EC2 instance or EBS, you can choose the bigger size of EC2 instance and size up the EBS volume to avoid EBS bottleneck.

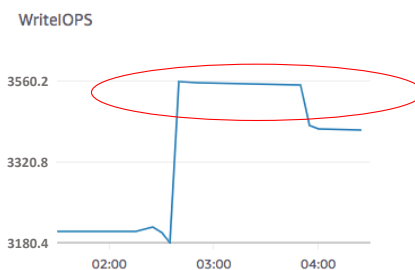


Figure 6: Near maximum IOPS limit

For more information, see [Amazon EBS Volume Performance on Linux Instances](#).

SwapUsage

The swapfile is enabled in the replication instance. It provides more stability for replication tasks. However, the overuse of a swapfile is the one of the major causes of performance bottlenecks. If a combination of low freeable memory and high swap memory usage is present, increase RI memory by increasing the RI EC2 instance and changing the EC2 type to a more memory intensive type such as R4.

NetworkReceiveThroughput

`NetworkReceiveThroughput` is a key factor to check network bandwidth between replication instances and the source database. This metric indicates the sum of all incoming traffic, including both source database and target database traffic. The largest volume of incoming traffic consists of unloaded and CDC records from the source database.

For security reasons, on-premises original source databases can't support exposure outside of their private subnets. DMS needs direct connections between all endpoints and replication instance. In these cases, the network engineer and security manager of

on-premises IDCs create a temporary network path between the source database and AWS DMS replication instance. The bandwidth for a temporarily created network channel tends to have some deficits. For example, you could see that the network bandwidth of DX is 100 Mbps(12 MB/s), but the maximum of network receive throughput is less than 12 MB/s.

Figure 7 shows the metric graph sample of above case.

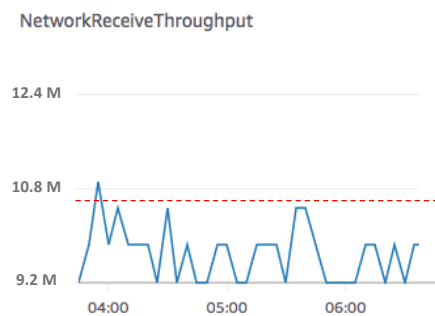


Figure 7: Low bandwidth between the source DB and RI

If you see this, notify all stakeholders about the low bandwidth of the network to find obstacles in the network.

There are many factors that limit the maximum network throughput. These factors include source database performance, and low-bandwidth network channel.

If you want to see the full bandwidth between the source database and RI, add more tasks with binary reader option. For example, if you see that one task uses an average of 40 Mbps, you can add two more tasks to check the maximum bandwidth at 120 Mbps. For help with solving network troubles, see [Networking Issues](#).

NetworkTransmitThroughput

This metric is the sum of outbound traffic from the replication instance. It includes outbound traffic from RI to source database, from RI to target database. In many cases, the DMS replication instance and the target database exist in the same VPC. It is rare to find a bottleneck in the network channel between the replication instance and the target database.

Source Endpoint

A source endpoint itself has no metric information. But it can hold configuration and initial parameters used when connected. As an Oracle source endpoint, it can configure the parameters listed in this section in the connection string for performance.

Note

Some parameters are used for binary reader mode. Some are for LogMiner mode.

There aren't many parameters related with performance in an Oracle source endpoint. But the choice between LogMiner mode and Binary Reader mode is critical to task performance.

The major difference between LogMiner and Binary Reader is where to analyze the online and archived redo log files. In LogMiner mode, the source database itself analyzes redo log files and CDC tasks pull the records matched with filter options. In Binary Reader mode, all online and archived redo log files, CDC task pull all the files from the source database by opening files and streaming it to the RI's local disk. And then the task in the replication instance analyzes them and finds matched records.

LogMiner mode could make source database busier, but it really depends on other conditions. For example, the following figure shows a majority of targeted transactions. When most transactions are targeted, Binary Reader is a better solution.

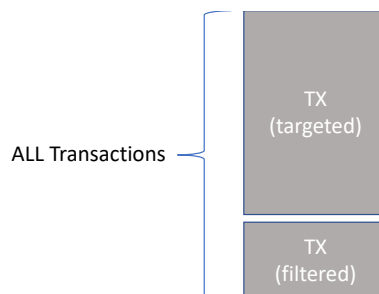


Figure 8: Transaction ratio (targeted > filtered)

The following figure shows a majority of filtered transactions. In this case, there is no reason to pull all of the records from source database. You can skip the majority of these records. Only a few records should be transferred to the target database.

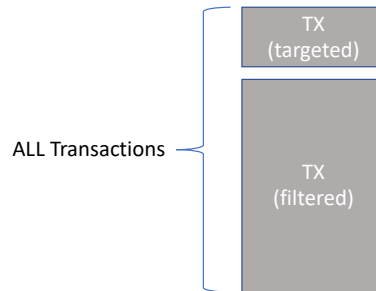


Figure 9: Transaction ratio (targeted < filtered)

In LogMiner mode, the CDC task pulls the full list of all transaction metadata in the source database. You can find it in the task detail debug logs. But for the only matched (targeted) records, the CDC task pulls the detail information from the source database through LogMiner. It relieves the burden of network bandwidth and source database utilizations.

Common Parameter

The following parameter is common to both LogMiner mode and Binary Reader mode.

retryInterval

The CDC records are pulled from the replication instance, not pushed. Because the CDC latency requirement is short, you should lower this parameter value. The default value is 5 (seconds). You can set to this value to 1 and above. We recommend that you set it to 1. Note, however, that this can put a burden on the source database.

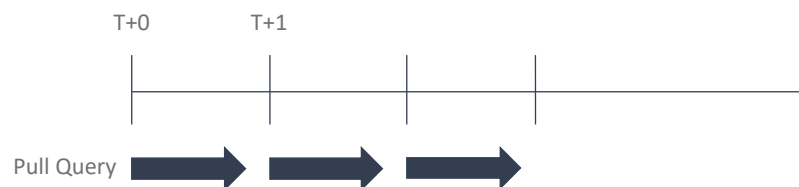


Figure 10: RetryInterval

LogMiner Mode

addSupplementalLogging

Set to **Y**. CDC in Oracle (source) depends on the supplemental logging feature. LogMiner can retrieve executed transaction logs under this option.

useLogminerReader

Set to **Y**.

Binary Reader Mode

useLogminerReader

Set to **N**.

useBfile

Set to **Y**.

archivedLogD

DMS tasks in on-going CDC option use the file input stream from online/archive redo log files. Sometimes, DMS tasks can't handle redo log files directly. This can happen in RDS, where file access is prohibited for managed services, and also in ASM, where ASM file access can be only for user ASM indirectly. In these circumstances, set this value to **False**.

archivedLogsOnly

A CDC task can handle all online and archive redo logs. If this option is set, the CDC task only handles archive redo log files. This means CDC latency can be delayed with various checkpoint options on the source database.

Target Endpoint

The target endpoint itself also has no metric information. It supports initial parameters in the connection string.

maxFileSize

DMS uses the **COPY** DML command in the full load. Each file size has a limit under the `maxFileSize` parameter. The **COPY** DML command is the best option to load the initial parameter instead of using **INSERT**.

In the DMS reference manual, DMS recommends that you disable triggers temporarily. However, sometimes triggers are required for business purposes. In these instances, you should allow CLOB/BLOB transferring through DMS.

BLOB/CLOB Operations

If the binary size of BLOB/CLOB columns has limited size under specific size (for example 8 MB), DMS task tries to unload all binary content from its source column and then transfers and upload it at one time.

But what if the size of binary of the BLOB/CLOB columns in the records are bigger than the limit size? In the PostgreSQL target, DMS concatenates the binary contents through one insert and multiple updates. So, if you set the chunk size of LOB in DMS to 1 MB, it results in the combination of one insert command and eight update commands.

If the target table has been triggered by the update DML, all transactions will be multiplied by the size of BLOB/CLOB columns of the record. To avoid this situation, remove the trigger and make other solutions (Batch, ETL, Application, etc.).

Task

A task is the main role in the pipeline between source and target databases. It pulls from the source database, filtering, transforming, rearranging, and uploading to the target database. Each task has an independent pipeline channel. This means if you use Binary Reader mode and, for example, the size of all the redo log files was 10 GB, each channel pulls 10 GB from the source database. If you use 10 multiple tasks, the total pulled stream size is almost 100 GB.

The task itself, is the source of power for pulling data from the source database, and this gives us the concurrency. If a record is waiting for target database process, all other records in the same task would also be waiting.

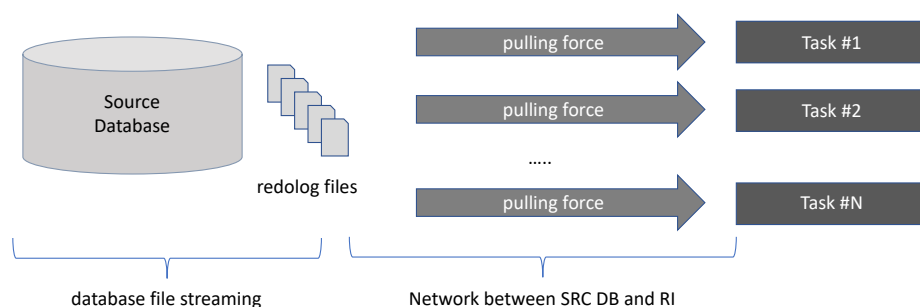


Figure 8: Pulling force in multiple tasks

The preceding diagram shows the trade-off relationship between tasks and various source endpoint resources. If you increase concurrency and throughput from the source database, you can complete more tasks. However, this action also causes more resource use, such as source database CPU, volume I/O (especially in Binary Reader mode), and network bandwidth from the source to replication instance.

| # of Tasks | Mode | Source Resource | | Concurrency | Read Throughput |
|-------------|---------------|--------------------------|------------------|-------------|-----------------|
| | | Database | Network | | |
| Few | Binary Reader | CPU very low I/O low | Bandwidth low | low | Medium |
| | LogMiner | CPU low I/O low | Bandwidth low | | |
| Many | Binary Reader | CPU low I/O high | Bandwidth high | high | high |
| | LogMiner | CPU medium I/O medium | Bandwidth medium | | |

If the ratio between matched records and all transaction is high, the Binary Reader is the better solution. Otherwise, we recommend that you use LogMiner.

Note

More tasks don't always equal more throughput. For example, in Binary Reader mode, one task needs more network bandwidth. More tasks can cause high I/O throughput in the source database and the network utilization between the source database and your replication instance.

Transaction Multiplication

Multiple tasks can also produce transaction multiplication. Transactions on source database will be guaranteed only when the all tables in a transaction should be in one task. If part of the tables in one transaction is separated into multiple tasks, all tasks related with this transaction would create another transaction in the target database.

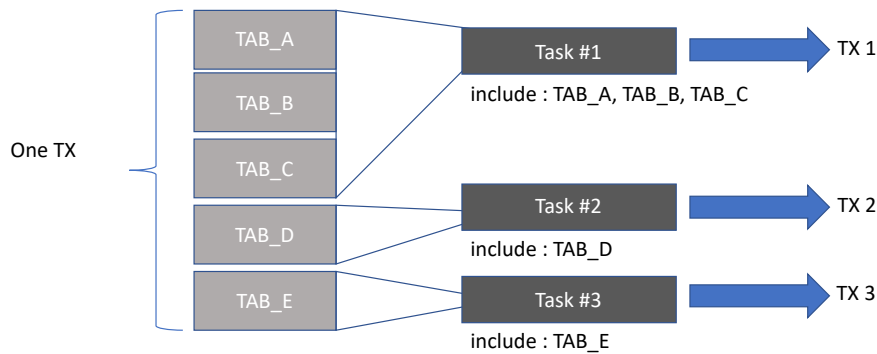


Figure 9. Transaction multiplication

If possible, review the original transaction, and make sure that the filter options in one task have all tables in the original transactions. Configurations (filter options in Tasks) that are mismatched with original transactions cause unnecessary transactions to the target database.

Batch Apply

DMS supports batch apply mode. **Apply** refers to its operation in the target database. Batch apply is the well-known parameter for performance. If batch apply is enabled, all transactions are maintained for certain periods (`BatchApplyTimeoutMin`) and records with same primary key are serialized, removing unnecessary change records from the serialized transactions.

The following parameters are used in the task settings.

Batch Apply Enabled Mode

- `BatchApplyPreserveTransaction` – If set to true, transactional integrity is preserved and a batch is guaranteed to contain all the changes within a transaction from the source. This option is valid for oracle database as target. If this is not applicable to your task, this parameter should be set to false.
- `BatchApplyTimeoutMin` / `BatchApplyTimeoutMax` — The default value is 1. `BatchApplyTimeoutMin` is the waiting time before transactions are applied to the target database. In some cases, delayed `BatchApplyTimeoutMin` value is used for batch apply. For example, if you set `BatchApplyTimeoutMin` to 3600 (1 hour), the CDC records will be applied in each hour. (In this case, you can use Amazon Redshift database as a target.)

- **BatchApplyMemoryLimit** — Batch apply rearranges all the CDC records in the task memory. If the task meets a shortage of memory, the task tries to apply the records in the memory to release allocated memory for the batch apply. Increase this parameter value to the appropriate value under `MemoryLimitTotal` size.

Transaction Mode (Batch Apply disabled)

- **MinTransactionSize** — Minimum number of changes to include in each transaction. The default value is 1000. This means that there are over 1,000 records in one transaction, the batch of these records could be applied to the target database. If there are many heavy transactions such like 10,000 CDC records per transaction, increase this parameter value to increase bulk operation.
- **CommitTimeout** — If the number of records from the source database is small, it isn't possible that more than `MinTransactionSize` CDC records exist in one transaction. In this case, `CommitTimeout` value is used. The default value is 1. So, after one second, whether there are sufficient CDC records in a transaction, the bulk of the records would be applied into the target database.

Common Parameters

MemoryLimitTotal

This indicates how much memory a task can use. If a task feels the shortage of memory, it tries to use disk space to store CDC records to relieve memory shortage. In the CDC on-going task, this operation is useless. Increasing this parameter value will prevent disk usage.

MemoryKeepTime

The CDC records in the transaction should be kept in memory for the certain amount of time. `MemoryKeepTime` indicates the residence time of CDC records in memory. The default value is **60** seconds. If the transaction of the source database is over 60 seconds, a task tries to use disk space to store CDC records. By checking the longest transaction time on the source database and how much time is needed to fetch these CDC records from source database, you increase this value to avoid disk usage.

Task CloudWatch Metrics

Task has several CloudWatch metrics. The important metrics are `CDCLatencySource`, and `CDCLatencyTarget`. There are other CDC metrics related to long transaction and uncommitted transactions.

Note

Some metrics are only meaningful in the full-load CDC tasks (FullLoadThroughputBandwidthSource, FullLoadThroughputBandwidthTarget, FullLoadThroughputRowsSource, FullLoadThroughputRowsTarget).

CDCLatencySource

This is the gap, in seconds, between the last event captured from the source endpoint and current system timestamp of the AWS DMSInstance. In short, this is the time gap between the oldest records of source database that are not loaded on the replication instance and the current timestamp of the replication instance itself. If no changes have been captured from the source due to task scoping, AWS DMS sets this value to **0**.

For example, some batch programs execute 100,000 records inserted and committed (T+0), and the replication instance is trying to get the committed records from the source database but the process of loading records needs time. At this point, the `CDCLatencySource` metric would be raised.

The network bottleneck between source database and replication instance could cause the increase of `CDCLatencySource` metric. Sometimes the source database table locks could also have a big impact.

Heavy transactions on the source database could cause an increase in the `CDCLatencySource` metric. If you see this metric value raised, try to raise the “pulling force” (source database → replication instance) with various parameters.

1. Create another task in the replication instance. The tasks in the replication instance raise both pulling force and the applying force. But the increasing tasks need to split tables in the source database for each task. It's not easy to design transaction boundary, so considering raising the network usage.

2. Raise the “stream buffer” counts. Stream buffers are used to store loaded recordset into the memory. Lots of LOB records can produce the following message in the CloudWatch task logs.

```
"Outgoing stream is full. Forwarding events to target is postponed"
```

When you see it, you should raise the count of stream buffer and the stream buffer size.

CDCLatencyTarget

This is the gap, in seconds, between the first event timestamp waiting to commit on the target and the current timestamp of the AWS DMS instance. The value occurs if there are transactions that are not handled by the target. Otherwise, if all transactions are applied, the target latency is the same as the source latency. The target latency should never be less than the source latency.

This metric shows that there are some issues in the target database. The issues include target table locking, network bandwidth shortage, and target database performance.

The following image shows the relationship between CDCLatencySource and CDCLatencyTarget.

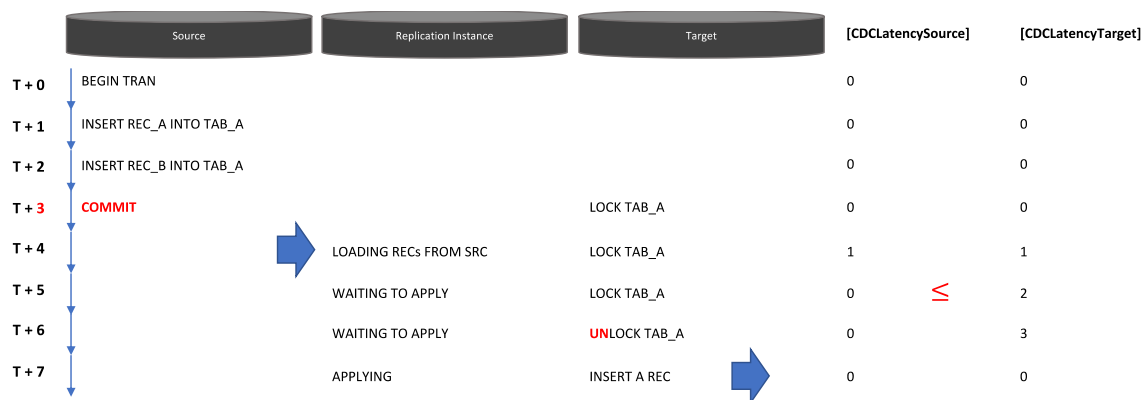


Figure 10. Delayed target latency

The following image shows a case of a target database with low CUD performance.



Figure 11. Target latency graph

There are many factors to increase target apply performance. Try upgrading the instance size. You can also try `BatchApply` mode to increase target performance.

With the previous situation, the customer suffered from cross-AZ DMS performance. We recommended that this customer use the `BatchApply` option to the task, which helped in this case. The below graph is the applied version.

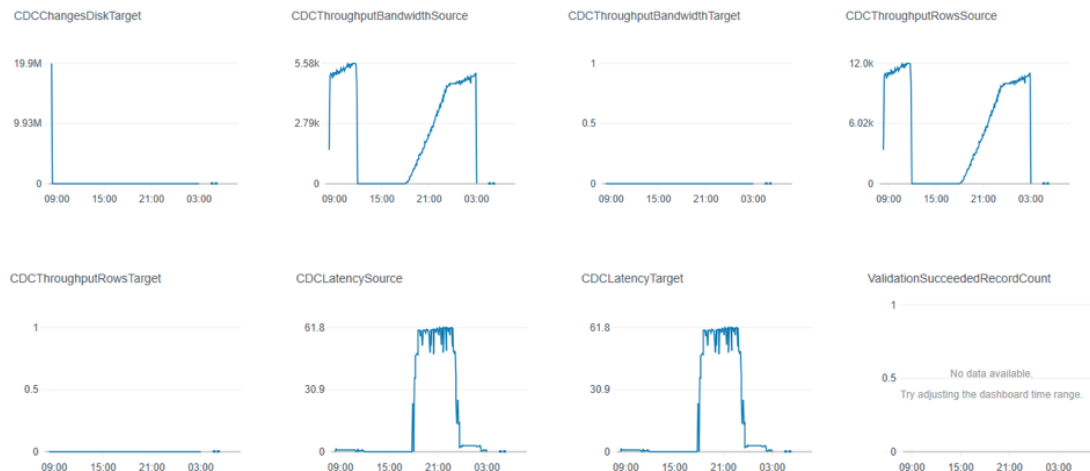


Figure 12. BatchApply mode enabled

The same value for shows for both `CDCLatencySource` and `CDCLatencyTarget`. This means that there is no target apply performance issue.

CDCThroughputRows

This metric indicates how many records are changed (updated, inserted, or deleted) in the source database. If a transaction was started and not committed, this metric would be raised, but the `CDCLatencySource` metric wouldn't. So, if you see the gap between

start time of `CDCIncomingChanges` and `CDCLatencySource` metrics, you can see that there are uncommitted transactions or long transactions on the source database.

CDCChangesMemorySource

This metric indicates how much memory is used for the uncommitted records of the source database on the replication instance. So `CDCIncomingChanges` and `CDCChangesMemorySource` metrics are raised together.

CDCChangesDiskSource

This metric indicates how much disk space is used for uncommitted records of the source database on the replication instance. Why disk used? Because the accumulated uncommitted records in the memory can't be applied to the target database, these records could impact the memory usage of the replication instance. So, the records stay in the memory for longer than the `MemoryKeepTime` specifies when they should be released to the disk.

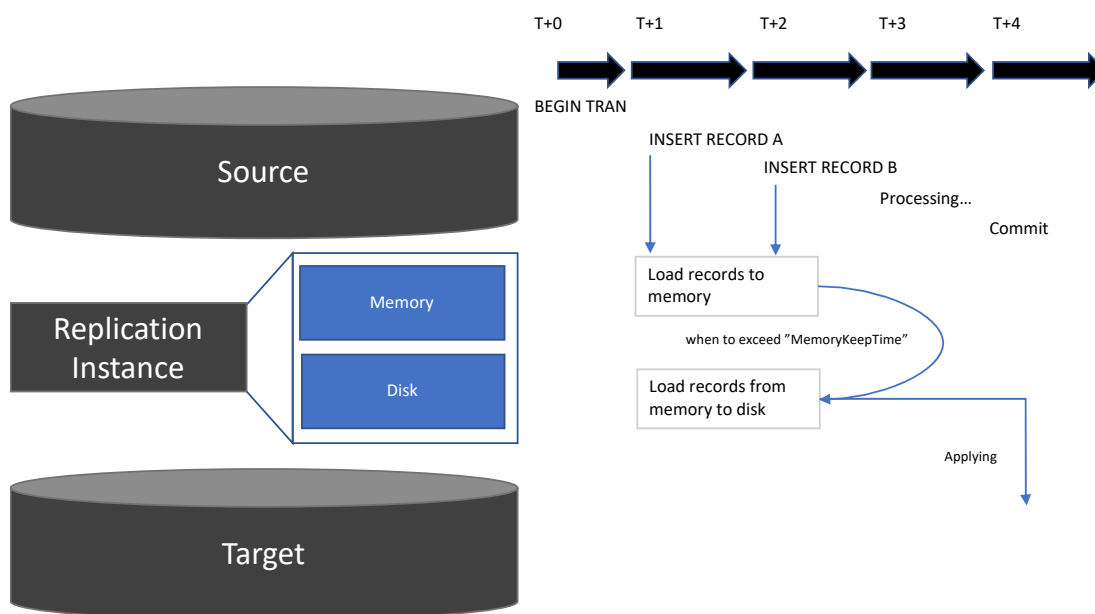


Figure 13. Uncommitted transactions processing

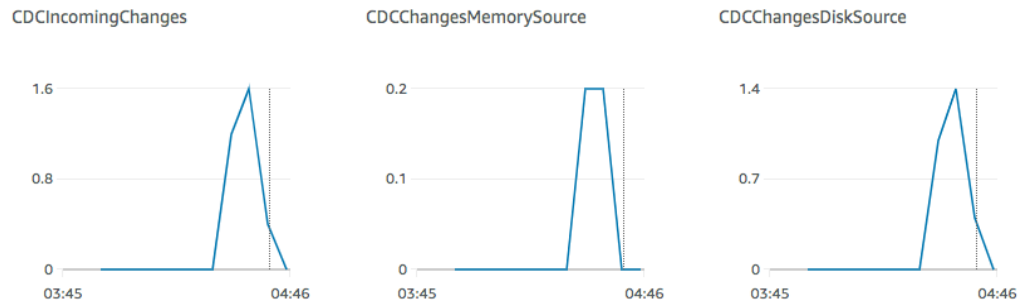


Figure 14. Uncommitted transaction and task metrics

When to Use DMS Metrics

The following diagram displays well-known workflows for when to adopt DMS as migration service.

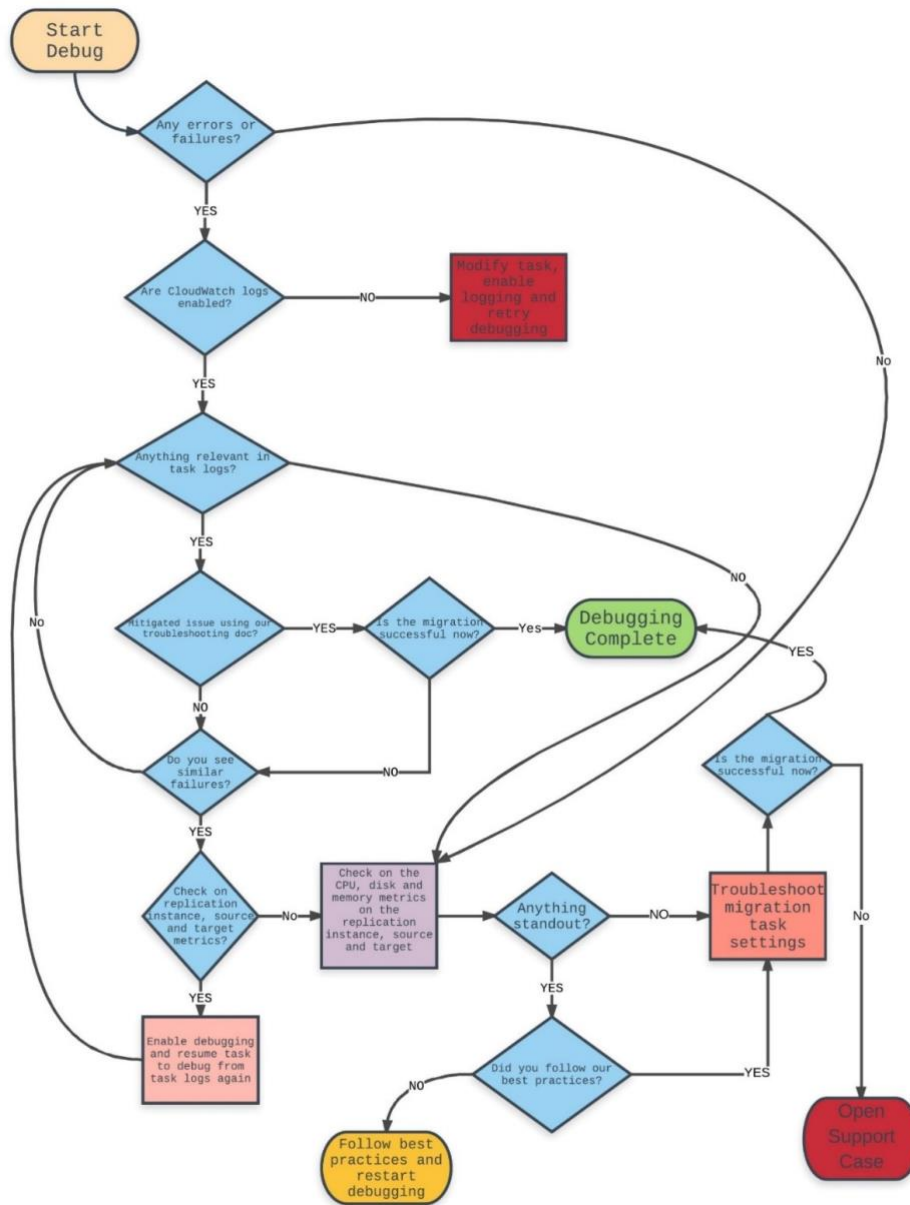


Figure 15. DMS trouble shoot flow

You can use this document as a reference where the **Check on replication instance, source and target metrics?** process appears in this diagram (located on the left bottom).

Metrics show helpful information, such as performance bottlenecks during DMS operation. But they don't show information that was improperly set. If you get a failure case on starting DMS tasks, check the RI, endpoints, and task settings first.

Use case

This section demonstrates a use case where an ecommerce company wants to use DMS to CDC replication from an on-premises, user-interfacing Oracle database to Aurora PostgreSQL to analyze user events in near-real-time real time.

Their Oracle database is heavily used, with almost 95% CPU utilization. There is a shortage of storage volume, and the database is only able to maintain four hours of archive log files. Their CDC requirement is latency delay below 5 seconds.

The network channel between on-premises IDC and AWS is 100M DX. The number of records for real-time replication is a few thousands per second at peak time.

In this use case, we took the following actions:

1. Before adopting CDC replication of DMS, we enabled Oracle databases to use the supplement logging feature. This action can cause performance impact to the source database. In this instance, we advised upgrading or increasing CPU capacity of the source database before enabling supplement logging. They added CPU resources (Maximum CPU Utilization is under 80%) and volume resources.
2. In CDC replication mode, we maintained 24-hour archive logs and enlarged the source database storage volume.
3. Because the number of all tables of the source database is over 3000 and the number of replication target tables is only 150, we used LogMiner mode.
4. A DMS task extracts the committed records from the source database. As the number of tasks increases, the pulling force also increases. Many tasks can cause the performance impacts such as higher CPU, I/O, network bandwidth utilization. So, it is critical to create an appropriate number of tasks to match customer requirements.

In this case, one task increases a 4% CPU utilization on the source database. So, we made four tasks to increase loading performance and to match the transformation operation for target database.

5. We noticed that at the dawn, the bulk operations (bulk insert, bulk update) have been executed and this type of workload made CDC replication delay impacts through CDCLatencySource and CDCLatencyTarget metrics. We analyzed the percentage of target transactions out of whole transactions on the source

database. It was just under 20%. The whole transaction log size was over 60% usage for the full network bandwidth, so we decided to use LogMiner mode instead of Binary Reader mode.

6. At the bulk operations, some LOB tables created issues that can be found on the CDCLatencyTarget and CDCChangesDiskSource metrics. On CDC mode, we recommended minimizing the disk usage to decrease latency and increase throughput.
7. In the target database, some interesting transactions were found. When one LOB record was inserted, the trigger bind to the table was fired repeatedly. This symptom is based on the DMS LOB handling. The process of handling a large LOB record in DMS has two parts. One is to insert a record in the target database. The other is to update the same record with a binary stream that has the target buffer sized repeatedly. If the target table has triggers, this update process would make for a low performance in loading and CDC.

Conclusion

There are many metrics on AWS DMS metrics. Each one can help you understand the bottlenecks in the DMS tasks. We hope this guide helps you overcome any performance barriers in your production environment.

Contributors

- KyungPyo Park, Solutions Architect, Geo Solutions Architect Team
- Yong Dae Kim, Professional Service, Ent Seg/Country Team
- Sungsoo Khim, Solutions Architect, Geo Solutions Architect Team
- Ebrahim Khiyami, Migration Solutions Architect, Geo Solutions Architect Team

Additional Resources

For additional information, see the following resources.

- [AWS Database Migration Service Documentation](#)
- [Debugging Your AWS DMS Migrations: What to Do When Things Go Wrong \(Part 1\)](#)

- [Debugging Your AWS DMS Migrations: What to Do When Things Go Wrong \(Part 2\)](#)
- [Debugging Your AWS DMS Migrations: What to Do When Things Go Wrong? \(Part 3\)](#)

Document Revisions

| Date | Description |
|-----------|-------------------|
| June 2020 | First publication |