persinity

User Guide – Persinity NDT Migrate

NDT Migrate methods and algorithms are patented technology (U.S. Provisional Patent Application No.62/087,754) of Persinity Inc., US. For inquiries email at info@persinity.com

Ver. 1.0 Beta

July 2015, Persinity, Inc.

This document is intended to help database administrators (DBAs) and system administrators to install, configure and use the Persinity NDT Migrate product. Although NDT Migrate is designed in a way to maximize the reliability of your data during migration, it is not recommended to do migration without being acquainted with this guide. Latest version of this document along with other related documentation is to be found at http://persinity.com/

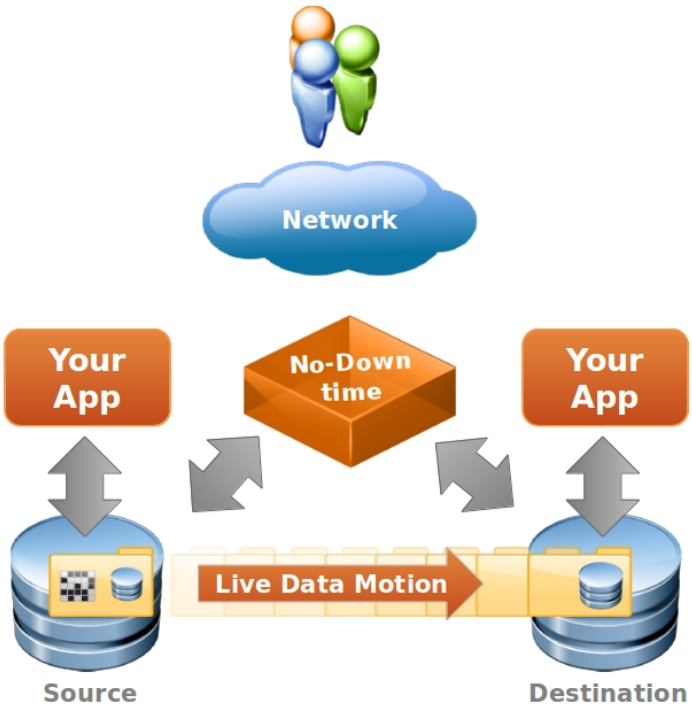
Table of Contents

Product Overview

The Persinity NDT Migrate product migrates database (DB) data from one IT stack to another, named source and destination.

NDT Migrate is designed to be used for migrating data of an application sensitive to *downtime*. *During migration the source can continue to serve user requests until the destination is ready to instantly take over.* Upon discretion of the NDT Migrate operator, the user traffic can be switched to the destination. Hence the no-down-time (NDT) feature.

The source and destination *IT stacks can be different*. For example the source application can run on 32 bit Windows Server 2008 operating system (OS) and store its data in MS SQL Server 8 DB, while the application on the migration destination can run on 64 bit RedHat Enterprise Linux 6 and store its data in Oracle 12c DB.



Use Cases

The NDT Migrate product characteristics make it an instrument that alone or in combination with third party administration tools can be used in vast range of use-cases:

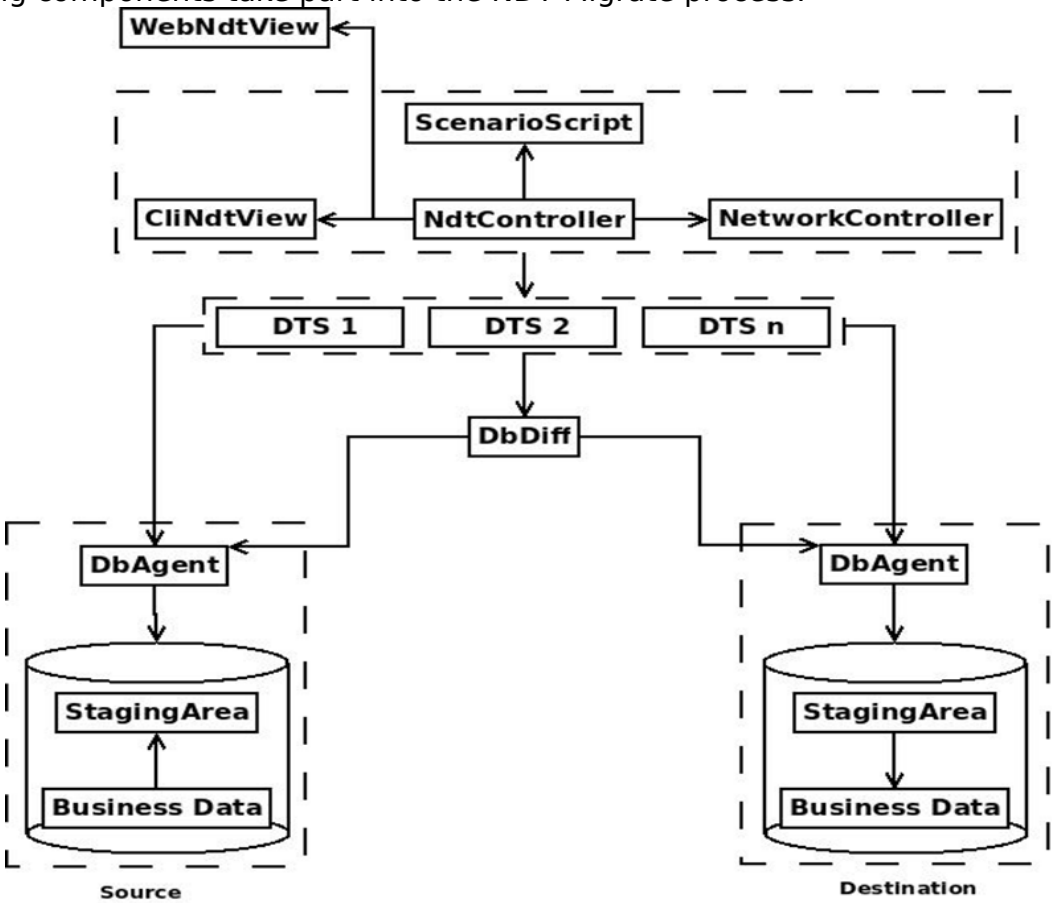
- Upgrade the hardware of application's host
- Move a running application to different hosting or cloud provider.
- Upgrade the version of the OS that hosts the data of a running application
- Change the vendor of the OS that hosts the data of an application
- Upgrade the version of a DB that hosts the data of an application
- Change the vendor of the DB that hosts the data of an application
- Anti-aging performance optimization of a running application – appropriate when the source application performance is hindered by aging IT stack and by junk artifacts that have accumulated with time, e.g. larger Windows OS Registry, log files, etc.
- Continuously maintain replica of the source application DB for fault-tolerance or reporting performance optimization, etc. In this case the source DB is the primary one, and the destination DB acts as the secondary one.
- Do consistent snapshot of the source application DB at given point in time and migrate it to a destination, which can be used for testing, development, etc.
- Record and replay application's data mutations – appropriate for regression testing or when evaluating a change that affects the DB layer of an application.

NDT Migrate Process Overview

User installs NDT Migrate and an empty destination application on a desired IT stack. Configures source and destination application DB connections and starts NDT Migrate. The latter starts tracking changes on the source application DB. These are written in the NDT source staging area and periodically migrated to the destination staging area. Meanwhile the user does initial export/import of the source data using its favorite DB export/import or backup/recovery tool. Once the initial migration is completed, the recorded and migrated changes are transformed from the destination staging area to the destination DB. Changes are periodically then migrated and transformed until the user retires the source application DB. Once retired, the last pending changes are migrated and transformed to the destination DB and the destination application is announced public. User requests are rerouted to it.

Technical Overview

The following components take part into the NDT Migrate process.



At the heart of the NDT Migrate product is the Persinity Data Transformation Server (DTS). It is efficient, robust and highly scalable data processing engine that extracts, transforms and loads data from/to various DBs.

A Database Agent acts as an adapter so that the DTS can work with various DBs, such as Oracle 8i, Oracle 9i, Oracle 10g, ..., Microsoft SQL Server 2008, 2012, ..., MySQL, Postgres, Hbase, etc. It is responsible for providing meta-information and SQL to the NdtController and DTS all DB related operations such as: meta-information for the involved tables and related data structures, data extraction queries, staging, transformation and loading SQL statements, etc. Upon demand, new agents can be plugged in to cover new DB vendor or version.

The DbAgent on the source node is responsible for providing SQL and meta-info to DTS and NDT Controller. This meta-information and SQLs are used for maintaining the change data capture (CDC) mechanism and the associated staging area.
The CDC mechanism is implemented with standard DB triggers. Triggers are mature DB technology that allows tracking of changes. Persinity Triggers are implemented in a fail-safe way that allows the source application to continue its normal operation even on complete failure of the CDC mechanism. Persinity CDC mechanism is also designed to be efficient. Even for very loaded DBs, the effect usually falls in 5%-10% of the time for which the core DB engine executes the application's mutation SQL. Since the DB engine execution time is only a small fraction of the total DB round-trip for the application's persistence request, the total effect of the CDC mechanism to the source application is negligible. However for special occasions, if required, Persinity can integrate other means for CDC, such as periodic difference comparisons of the source tables or mining transaction log DB structures at physical level.
The source DB staging area is well isolated structure from the business data that records its mutations caught by the CDC mechanism. The recording is done in real time in a form suitable for delayed extraction, so that the user can fine-tune and minimize the performance impact of the extraction.

The DbAgent on the destination node is responsible for providing SQL and meta-information to DTS and NDTController for maintaining the load staging area. The latter provides means for rapid recovery of a stalled migration step. It is also used for pre-load transformations, such ensuring the destination DB is in consistent state even after efficient, but inconsistent, initial import of source data.

The DbDiff module uses the DbAgents to generate mapping information so that the DTS is able to map data extracted from the source to the destination logical schema.

The NDTController orchestrates the DTS and the DbAgents according the predefined migration scenario. The scenario is comprised of various migration steps such as setup, migration, consistency checks and clean-up. The migration scenario is configurable and extensible, so that the user can add custom steps such as: record transformations, data checks at the destination or integration with third party systems.

The NetworkController is responsible for interacting with the network, so that the user requests are rerouted from the source to the destination application, once the migration has completed.

The NDT Views represent the UI. They display migration progress, messages and logs and accept operator's input. NDT Migrate provides common line and web interfaces for the operators.
All modules are implemented in pure Java, so that migrations are supported for various platforms. DbAgents are implemented in Java and may contain DB specific SQL or analog data manipulation language for efficient communication with the given DB.
Communications between modules happens through resilient non-blocking light-weight messaging for efficient, scalable and reliable migration. DB communication and data transportation relies on DB specific JDBC drivers.

Requirements

Software requirements

- JVM 1.7 or later for your source and destination OS. Can be installed from http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html by following the instructions at the download page.
- JDBC Driver(s) for the source and destination DBs, according the vendor. E.g. for Oracle you can download and install drivers from http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html by following the instructions at the download page.

Hardware requirements

For live migrations, it is recommended that NDT Migrate is installed on a dedicated host with 64 bit OS, at least 4GB RAM and 4 CPU cores.

The network connections between the NDT Migrate host and DB hosts should be at least 100Mbit.

The source DB host needs to have at least 20% spared CPU and memory capacity during the live migration and recommended value is 30%.

The recommended HDD I/O spared capacity on the source DB host is 50% in order to accommodate NDT Migrate activities during data migration.

All spared capacity requirements can be met more easily by choosing non-peak hours for the migration, when the source DB is not loaded to its maximum values.

Supported Databases

NDT Migrate supports migration from/to combination of the following DBs:

- Oracle 8i, 9i, 10g, 11g, 12c – all versions
- MS SQL Server 6, 7, 8, 9, 10, 11 and 12 - all versions
- MySql 4, 5 – all versions with the Inno DB engine
- IBM DB2 8.1, 9, 10 – all versions
- Postgres 7.4, 8, 9 – all versions
- SAP Hana
- Hbase
- Gemfire 7, 8 – all versions
- Others on demand

Installation and Startup

As prerequisite, you need to have an empty instance of your application installed as destination for the migration. Then:

1. Make sure you have fulfilled the hardware and software requirements for the machine that will host the NDT Migrate product.
2. In the NDT Migrate intallation package select the ndt-migrate-<version>.zip and unzip it into a directory at the migration host ($NDT_HOME) by using utility for manipulating archives such as 7-zip, unzip, etc.
3. Copy the "$NDT_HOME/config/ndt-db.properties.template" file to ndt-db.properties in the same directory and fill in:
    - the connection details of the source and destination application DBs – edit the "src.app.*" and "dst.app.*" properties
    - the staging area DB URLs which is recommended to be the same as the application DB URLs – edit the "src.ndt.*" and "dst.ndt.*" properties.

For the URLs follow the syntax of the JDBC driver for your application's DB vendor.

Follows example for an Oracle DB using thin JDBC driver.

```
# Source application DB
src.app.db.url = jdbc:oracle:thin:@my-source-ora-server:1521:orcl
src.app.db.user = testapp_user
src.app.db.pass = testapp_pass

# Destination application DB
dst.app.db.url = jdbc:oracle:thin:@my-dest-ora-server:1521:orcl
dst.app.db.user = testapp1_user
dst.app.db.pass = testapp1_pass

# Source NDT DB
src.ndt.db.url = jdbc:oracle:thin:@my-source-ora-server:1521:orcl
src.ndt.db.user = ndt
src.ndt.db.pass = ndt

# Destination NDT DB
dst.ndt.db.url = jdbc:oracle:thin:@my-dest-ora-server:1521:orcl
dst.ndt.db.user = ndt1
dst.ndt.db.pass = ndt1
```

4. Create DB users for the NDT Migrate staging areas.

It is recommended to have the NDT staging areas co-located with their application schema counterparts, e.g. the source staging area to use the source application DB server.

- Log in your source application DB as DBA and execute the script $NDT_HOME/config/admin.sql.template. The script will ask for user name and password for the source NDT staging area DB user*
- Log in your destination application DB as DBA and execute the script $NDT_HOME/config/admin.sql.template. The script will ask for user name and password for the destination NDT staging area DB user*

* If your DB tool does not support parameters collection, the script will fail in collecting the needed user name and password. Copy the template script and fill in the credentials manually before invoking the scripts.

5. Start the NDT Migrate server by using the provided bash or bat scripts. Follows an example with the bash script.

```
$NDT_HOME/bin/ndt-migrate.sh
```

On success you should see the welcome messages.

```
$NDT_HOME/bin/ndt-migrate.sh
Persinity NDT Migrate <ver>
Loading
....

Confirm to setup NDT.
Enter "Y" to continue:
```

Migrate Guide

This section gives you information about the live data migration process with Persinity NDT Migrate. You may want to review this guide as preparation for your migration and use it as companion during the migration process.

Preparation

Install empty instance of your application along with its empty DB schema on machine designated as migration destination. The destination IT stack can differ from the source one, e.g. different OS, DB vendor, application server, etc.

Given that the source application and DB is live, you may cancel the migration process without losing any source application data or availability of your source application. Still, as good practice for any activity on production server, *it is recommended to back up the source DB*, before proceeding with the migration.

Install, configure and start up the NDT Migrate server by following the “Installation and Startup” section of this guide.

Once the NDT Migrate server is started, follow the instructions in the Web UI/CLI, which will guide you through the No-Down Time Migration scenario. To cancel the migration process, press Ctrl-C (in the CLI).

Setup Step

Acknowledge the UI prompt to enter this step.

```
Confirm to setup NDT.
Enter "Y" to continue: Y
```

During the setup step the NDT Migrate:

- Deploys DB agents to the pre-configured source and destination DBs.
- Initializes the migration (DTS) engine according the pre-supplied configuration (refer to the “Fine Tuning” section)
- Disables constraints on the destination to optimize the initial data transfer.
- Starts recording changes of the source DB data in the source staging area.

```
Installing NDT at Source DB "testapp_user", Source Staging "ndt", Destination Staging "ndt1" and Destination DB “testapp1_user”.
```

Migrate Changes Step

Acknowledge the UI prompt to enter this step.

```
Confirm to start NDT Migrate.
Enter "Y" to continue: Y
```

All recorded changes are periodically migrated from the source to the destination. Each data movement is called a window. A migration window moves set of changes from the source staging area to the destination staging area. You may preset the window size and frequency to adjust the migration speed-vs-load balance on the source DB, migration and destination machine(s). (Refer to the “Fine Tuning” section). During the migration you will see continuous status progress:

```
Migrated rows to Staging: 10479, to Destination: 0
```

The number of migrated changes indicates how many mutations have been recorded and migrated to the destination staging area. The number of transferred records indicates how many of them are applied to the tables of the destination application. The “Migrated rows to

Destination" counter will not be incremented during this step.

Initial Transfer Step

This is placeholder step that is automatically started after the previous step is started. It requires the operator to use third party data backup/recovery or export/import tool. Examples of such tools to use are the Oracle "exp/imp" tool, the "data pump" or "RMAN". Refer to the corresponding guide if unsure how to use them during this step.

```
You may start with initial transfer using your favorite backup/recovery or export/import tool.
Confirm when initial transfer is completed.
Enter "Y" to continue:
```

You use an export/import or backup/recovery tool to bulk transfer the source data to the empty DB of the destination application. Some DB vendors do not have data export/backup tools, which provide consistent snapshot view of the exported DB. Generally a consistent snapshot export tool (if available), is considered as serious load to the source DB. Since NDT Migrate already records source changes, the export may or may not be consistent. NDT Migrate will take care of merging the recorded changes with the destination data imported from inconsistent export to achieve consistent destination DB.
Once the data is successfully imported in the destination, you may proceed to the next step by acknowledging the UI prompt.

Consistent Destination Step

NDT Migrate enters automatically this step right after the previous step was completed.

```
Rolling Destination DB "testapp1_user" to consistent state...
```

During this step the NDT Migrate applies to the tables of the destination application all staged changes that were migrated during and potentially not caught by the initial export tool. The "Migrated rows to Destination" status counter starts to increase, as the staged mutations are applied to the destination.

```
Migrated rows to Staging: 23419, to Destination: 6734
```

Through combination of the initially transferred data and the recorded changes, the destination DB is brought to consistent state. The destination constraints are then enabled and optionally custom supplied checks ran to validate the destination application state.

Delta Migration Step

NDT Migrate enters this step automatically after the destination is validated as consistent.
NDT Migrate continues to periodically apply source application changes to the destination application tables to keep the source and destination DBs in sync. The operator can ask users to validate destination data, and use the destination system (in read-only mode for the Beta) to confirm that it meets expectations.

```
Now you can stop Source Application.
Confirm when Source Application is stopped.
Enter "Y" to continue:
```

In order to minimize the subsequent downtime period, you may want to stop the source application once the "Migrated rows to Destination" status counter has caught up with the "Migrated rows to Staging" counter.

```
Migrated rows to Staging: 28475, to Destination:  28224
```

Retire Source Step

Important: *Post this step the destination DB becomes the primary one and its data may*

deviate from the retired source DB data. Hence it may be impossible to revert users back to the source application and its DB without loosing data.

When the users are comfortable with the functional and performance aspect of the destination application, the operator can stop the source/legacy application. Once the application has been stopped and no data changes happen on the source DB, the operators confirms the NDT Migration prompt.
The last pending changes of the retired source application are migrated.

```
Waiting migration to complete...
```

Since the source application has been stopped, it will not generate new data changes and the “Migrated rows to Destination” counter will catch up with the “Migrated rows to Staging” one. This is the downtime period of your application, as the destination DB is not yet fully ready to take over and the source one has been stopped. If the source application has been retired while the status counter difference was within a couple of thousand of records, the downtime should be within a minute.
When the “Migrated rows to Destination” counter closes the gap with the “Migrated rows to Staging” one, you are approaching the end of the migration.

```
Migrated rows to Staging: 28475, to Destination:  28473
```

Note: In case you are using a backup/recovery or export/import tool that is not integrated with the Persinity NDT Migrate product or when such tools are not capable of doing consistent snapshot of the source DB data, it is possible that there is desynchronization between the mutated source data and the initially exported data. NDT Migrate will recognize such scenario and auto-correct its migration so that the end result of the migration is target DB that is consistent to the source one. In this case however the two status counters may differ slightly. These counters are for estimated progress reporting only. NDT Migrate tracks not the informative counters, but the actual data states. It ends the migration process only when all the source data has been migrated to a consistent destination DB schema.

Once this step is completed, the destination application is the primary one.
Optionally the user network traffic is rerouted automatically to the destination server.

Clean Up Step
In this step, the NDT Agents are uninstalled from the DBs along with all their artifacts. The system announces the successful end of the process with an UI message.

```
Uninstalling NDT
...
Migration completed.
```

Note that this step is carried out even if you choose to cancel the migration process by hitting “Ctrl-C” at one of the previous steps.

Fine Tuning

Skipping tables during migration

You may want to instruct NDT Migrate to not migrate data for some tables. This might be useful when:

- your source application DB schema differs* from the destination one, you need to instruct the DbDiff module to exclude the different tables from the migration process.
- there are tables with data you don't want migrated, e.g. junk or sensitive or denormilized data.

List the tables you need to be skipped during Migration in the "$NDT_HOME/config/ndt-db.properties" file in the "src.app.db.tables.skiplist" and " dst.app.db.tables.skiplist" properties respectively.
For example if there are is a table called "my_skiptable1" in the source DB that you want to be skipped during migrate, you need to add the following entry:

```
src.app.db.tables.skiplist = my_skiptable1
```

Note that:

- if the "my_skiptable1" is present in the destination you need to list it under dst.app. db.tables.skiplist property too.
- If the "my_skiptable1" is parent table to a child "my_skiptable2", you need to add the child tables too.

Follows an extract from the "$NDT_HOME/config/ndt-db.properties" for the above example:

```
src.app.db.tables.skiplist = my_skiptable1, my_skiptable2
dst.app.db.tables.skiplist = my_skiptable1, my_skiptable2
```

*If you need to do more complex transformation of the source data to match different destination DB schema (e.g. for reporting, integration with TP systems, etc purposes), you may want to take a look at the Persinity NDT Transform product.

Adjusting the Performance-vs-load balance:

NDT Migrate can be configured with the desired performance-vs-resources balance:

- *economy* processing that is easy on the source, destination and NDT related resources;
- *multi-thread* processing within single node that provides balanced migration (default setting);
- *multi-thread-multi-node* processing with linear scaling for aggressive migration of very large data sets;

In order to fine-tune the migration performance, you edit the "$NDT_HOME/config/ndt-controller.properties" file. Follows brief description of its configuration parameters:

Parameter	Default Value	Description
etl.window.size	80	Sets how many of the recorded transactions should be moved in a window. Increase this value if you expect high load that results in high number of transactions to be migrated. This should increase the transfer speed and minimize down time. Decrease this value to decrease the load on the source DB due to excessive data reads. Decrease this value to decrease the load on the NDT Migrate host.
etl.instruction.size	100	Sets how many changes a single CPU core on the NDT Migrate host should handle per transfer window. Set this

		value roughly to the etl.window.size * average_statements_per_transaction / cpu_cores / 2. E.g. for migrate host with 4 available CPU cores, etl.windows.size = 80 transactions and on average 10 insert, update or delete SQL statements per transaction, the etl.instructtion.size value is calculated as 80 * 10 / 4 / 2 = 100. Decrease value to offload the NDT Migrate host CPU and memory utilization and to decrease the load on the source DB due to frequent reads. Increase this value to speed up the transfer and minimize down time during live migrations.
etl.window.check.interval.seconds	5	Sets how often the source DB will be checked for recorded changes. Increase this value to decrease the load on the source DB and the NDT Migrate host. Decrease this value to minimize the destination data latency and the downtime period during live migrations. Destination data latency shows how much time is needed for a change in the source DB to be migrated in the destination. It is key factor during replication use-cases. E.g. when NDT Migrate is used to maintain fault-tolerance replica, the latency should be minimal; when NDT Migrate is used to maintain replica for test and development purposes or reporting the latency may be even hours or days.
etl.metrics.reporting.interval.seconds	1	Determines how often the status counters of the migrated records are updated in the UI. Higher values increase the vagueness of progress reporting while preserving NDT Migrate host resources to some extend and vice-a-versa.
dbagent.clog.gc.interval.seconds	600	Sets how often processed records in the staging areas are cleared. Decrease this value to force frequent clearing - if you expect intensive mutations on the source application data, when you have increased etl. window values or want to free space more aggressively. Increase this value to conserve source and destination DB resources.
haka.enable	true	Turns on/off the multi-thread processing mode of the DTS. Haka is the generic Persinity framework for parallel processing used by DTS. Set this value to false to switch from parallel to serial processing. Suitable when you want to aggressively decrease the load on the source DB and NDT Migrate host(s) on the price of much slower migration process and increased downtime in case of live migration use-case.

There are additional Haka framework properties that can be edited in the "$NDT_HOME/config/haka-ndt.conf" file. Follows brief description of its configuration parameters:

Parameter	**Default Value**	**Description**
haka.workers	10	Sets how many parallel DTS/Haka window sub/task to work. Increase this value if the machine that hosts NDT Migrate product has free resources and you want to improve its migration throughput.
haka.watchdog-period	120 seconds	Period on which a check is performed for stalled DTS/Haka window sub/task.
haka.status-update-timeout	300 seconds	The timeout after which a stalled DTS/Haka window sub/task will be restarted. Restarts are handled automatically and

		internally by DTS so that the successfully moved data is not lost or repeated. This setting is not used when haka.enabled mode is set to false. Increase this value if you experience frequent timeouts or when no data is being migrated due to massive time-outs (warning messages in the log).

You can change the location of the NDT staging areas. For example if there are security or performance restrictions in the source or the destination application DBs, you can instruct NDT Migrate to install the source/destination staging area in dedicated server. Note that in order CDC to have minimal impact on the source DB, the source staging area should be accessible from the source app DB by fast channel (ideally IPC or analog local machine communication). To alter the staging areas location edit the “src.ndt.*” and “dst.ndt.*” properties in the “$NDT_HOME/config/ndt-db.properties file”.

Trouble-shooting
If you have support agreement for the NDT Migrate product, you may file your questions at support@persinity.com at any time.

Recovery of your system to Pre-NDT state.
If hardware prerequisites are followed, NDT Migrate is designed to not interfere significantly with your source application. NDT Migrate installs and uninstalls automatically its artifacts, including the fail-safe triggers on the source application DB. However in order to preserve disk space on the DBs that host the NDT staging areas, it is recommended to uninstall NDT Migrate when not needed anymore.

This will clean up its artifacts and remove the Change Data Capture triggers from your source DB schema, even in the case of interrupted migration, e.g. in an event of power outage.

To force clean up, you start NDT Migrate: press “Ctrl-C” on the prompt for starting NDT Migrate:

```
Persinity NDT Migrate v1.0-beta

Loading
....

Confirm to setup NDT.
Enter "Y" to continue: Y


Installing NDT at Source DB "testapp_user", Source Staging "ndt",
Destination Staging "ndt1" and Destination DB "testapp1_user"
..


Confirm to start NDT Migrate.
Enter "Y" to continue:
^C

Aborting on interrupt...

Uninstalling NDT
..............

Migration completed
```

Debug Information
You may use the following tools to solve uncommon issues:

- Messages for migration errors are displayed on the NDT Web UI/CLI.
- Migration logs can be found at the “$NDT_HOME/log” directory and used for further diagnosis or monitoring the detailed progress.
- DB Agent error logs are found in the “ndt_log” table of the source and destination staging areas. You may want to consult them if you suspect there is a problem in the NDT CDC or change log mechanism.

If required by the Persinity support team, you may enable detailed logs by editing the “log4j.rootLogger” option of the “$NDT_HOME/config/log4j.properties” file:

```
log4j.rootLogger=DEBUG, fall
```

Beware that the “DEBUG” log mode may consume disk space.

In case you are not able to resolve the issues by following the error instruction, following the support engineer requests and mining the log you may be asked to zip the contents of the “$NDT_HOME/log” and “$NDT_HOME/config” directories and send it to support@persinity.com

DB Issues

On loaded DB systems, it is necessary to increase the values of the parameters that control their concurrency. Refer to the errors in the NDT Migrate log file and follow your DB vendor documentation. Follows some common settings:

- Set the OPEN_CURSOR Oracle parameter to at least 5000, to avoid “ORA-01000: maximum open cursors exceeded” during intensive migrations.
- Set larger space quota for the staging area DB users. In order to provide fail-safety, the default quota for recorded mutations is set to 1GB. Thus even if change log clearing fails or your source application experiences unusually high rate of data mutation (e.g. DoS attack), the staging area won't consume more than the preset amount of space. To increase this setting issue ALTER USER statement following your DB vendor documentation:

```
ALTER USER <ndtusername> QUOTA 2048M ON TABLESPACE users;
```

Release Notes

The NDT Migrate v1.0 Beta is intended for early preview of the product to be released later. Users are encouraged to experiment with it and provide feedback, which will be highly appreciated.

While an effort has been put by Persinity to deliver stable Beta Product, it should be still considered a beta quality product. Enterprise performance is not a goal at this stage, especially under MS Windows.

The following features are not provided by the NDT Migrate v1.0 Beta:

- Move a running application to different hosting or cloud provider
- Record and replay application's data mutations – appropriate for regression testing or when evaluating a change that affects the DB layer of an application.
- From the list of supported vendors the Beta works only with Oracle 10g, 11g and 12c versions.
- The NDT Migrate scenario and its steps are fixed and are not configurable and extensible. Custom supplied checks are not supported. Users can rely on enabled DB constraints and manual checks in the destination DB/application.
- The Network Controller is not available; the migration operator has to configure manually the network so that user requests are automatically rerouted to the destination application. Alternatively the application users can be provided with new end point to access the migrated application.
- The NDT Migrate Web UI is not provided.
- Multi-thread-multi-node processing with linear scaling is not supported in Beta. Users can choose from economy and multi-thread processing mode.
- Migrating data for table without primary key or table with primary key that can mutate (e.g. updated by the user) is not NDT Migrate in beta. If data in such tables is not crucial for your migrate, you can instruct NDT Migrate to skip them by using the “skip.tables” setting for the source and destination DBs (refer to the “Skipping tables during migration” section). If you need NDT Migrate to migrate data for such tables, you can workaround the missing feature by replacing the mutating primary key with surrogate primary key (adding surrogate primary key in the case of table without primary key). The surrogate primary key is an artificial ID column, usually of large numeric type, which does not take into the business logic, but is solely used to uniquely identify records and changes on them. The surrogate primary key is usually created on a column with auto-increment option, so that the application database populates the needed unique values transparently for the application.