**IBD Replication**

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **DATE** | **AUTHOR** | **REVISION** | **HISTORY LOG** |
| June/7/16 | Martin Wang |  | Created |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[1 Overview 3](#_Toc453511165)

[2 Used Modules 3](#_Toc453511166)

[2.1 DX 3](#_Toc453511167)

[2.2 MDS 4](#_Toc453511168)

[3 Work flow 4](#_Toc453511169)

[4 Commands 6](#_Toc453511170)

[4.1 Source Commands 7](#_Toc453511171)

[4.2 Target Commands 7](#_Toc453511172)

[5 Configuration 8](#_Toc453511173)

# Overview

Each ibdserver can be run as replication source and/or target. It is NOT allowed to configure a replication with same source volume and target volume.

IBD replication uses DX to transfer commands and data. IBD replication uses MDS APIs to read&write fp&data.

Replication only transfers data which does not exist on target. This could save a lot of IO and network, as well as the total time of a replication.

# Used Modules

## DX

IBD replication uses DX to transfer commands and data.

Replication source side:

In reps\_initialization(), set dxag from setup.

In reps\_start(), create dxa from dxag.

In \_\_reps\_send\_pkg(), use dxt\_send() to send control message and data message. Data message includes bitmap+fp or bitmap+fp+data.

In \_\_reps\_try\_recv\_pkg(), use dxt\_get\_request() to get a control message. Then use dxt\_get\_dbuf to get the corresponding data message.

Replication target side:

In rept\_initialization(), set dxpg from setup.

In rept\_start(), create dxp from dxpg.

In \_\_rept\_send\_pkg(), use dxt\_send() to send control message and data message. Data message includes bitmap or DATA\_TYPE\_DONE message.

In \_\_rept\_try\_recv\_pkg(), use dxt\_get\_request() to get a control message. Then use dxt\_get\_dbuf to get the corresponding data message.

## MDS

IBD replication uses the 5 MDS APIs. For more detail, please refer to ms\_intf.h.

MS\_Read\_Fp\_Async

It is used on replication source side, phase 1. It reads fp array of specified volume, offset, length. It returns fp found bitmap and fp array. Non-existing fp will be cleared to 0 in fp array.

MS\_Write\_Fp\_Async

It is used on replication target side, phase 1. It checks if corresponding fp is exist. It returns fp bitmap for missed fp.

MS\_Read\_Data\_Async

It is used on replication source side, phase 2. It reads data(4kB namespace) corresponding to fp. The number of fp is depends on count and iovec.iov\_len.

MS\_Write\_Data\_Async

It is used on replication target side, phase 2. It writes corresponding fp and data as the bitmap indicates.

MS\_Read\_Fp\_Snapdiff\_Async

It is used on replication source side, phase 1. It is similar to MS\_Read\_Fp\_Async. It reads fp changed between two snapshots.

# Work flow

For each namespace, it needs steps below to be replicated from source to target. Namespace won't be dealt separately. A call to MDS API will dealt a segment, that is namespace \* bitmap length.

1. Replication source side, phase 1.

Source IBD server reads fp array of specified volume, offset, length. It uses MS\_Read\_Fp\_Async to get fp found bitmap and fp array. Bitmap indicates if each namespace has data. The bitmap+fp will be keeped in a list on source for phase 2 usage. Non-existing fp will be purged from fp array before send bitmap+fp to target.

2. Replication target side, phase 1.

Target IBD server receives bitmap+fp with corresponding volume, offset, length. It uses MS\_Write\_Fp\_Async to get missed bitmap. Then tranfer the missed bitmap will be send to source side.

3. Replication source side, phase 2.

Source IBD server receives missed bitmap with corresponding volume, offset, length. Then source searches bitmap+fp list keeped in phase 1, for the one with the same offset. Fp for missed bitmap will be copied from fp keeped in phase 1. A missed bitmap+fp could be assembled for reading data, by using MS\_Read\_Data\_Async. Missed bitmap+fp+data will be send to target to write.

4. Replication target side, phase 2.

Target IBD server receives missed bitmap+fp+data with corresponding volume, offset, length. It uses MS\_Write\_Data\_Async to write bitmap+fp+data. Then send the data write done message to source side.

5. Replication source side, phase 3.

Source IBD server receives data write done message. Then decrease the data request waiting count. This count is for limiting data request send to target, to avoid resource exhausting on target.



# Commands

IBD replication has 3 kinds of commands:

Start: start reps and corresponding rept.

Display Setting: show the configuration of reps and rept.

Info: display reps and rept information, includes progress.

## Source Commands

Replication source side:

start: start reps and corresponding rept

sample:ibdmanager -r s -m reps start reps\_name

sample:ibdmanager -r s -m reps start reps\_name snap\_name

sample:ibdmanager -r s -m reps start reps\_name snap\_name snap\_name2

pause: pause reps and corresponding rept

sample:ibdmanager -r s -m reps pause reps\_name

continue: continue a paused reps and corresponding rept

sample:ibdmanager -r s -m reps continue reps\_name

info: information display command

progress: display reps progress

sample:ibdmanager -r s -m reps -i reps\_name info progress

display: show the configuration of reps

sample:ibdmanager -r s -m reps display setup

sample:ibdmanager -r s -m reps -i reps\_name display setup

sample:ibdmanager -r s -m reps display working

sample:ibdmanager -r s -m reps -i reps\_name display working

## Target Commands

Replication target side:

info: information display command

progress: display rept progress

sample:ibdmanager -r s -m rept -i rept\_name info progress

display: show the configuration of rept

sample:ibdmanager -r s -m rept display setup

sample:ibdmanager -r s -m rept -i rept\_name display setup

sample:ibdmanager -r s -m rept display working

sample:ibdmanager -r s -m rept -i rept\_name display working

# Configuration

On git, there are sample replication configuration files in

itest/reptest

ibdserver.conf.s is for source.

ibdserver.conf.t is for target.

Global support for replication source(reps, repsc) is required on source.

Global support for replication target(rept, reptc) is required on target.

Global support for mrw(mrw, mrwc) and dx(dxa, dxac, dxp, dxpc, dxt) are also required as precondition.

The value of rept\_name for reps on source should match the rept name on target.

The value of peer\_uuid for dxa and dxp on source and target should match the name of each other.

The value of voluuid for reps and rept should match the name of volume\_uuid for sac.

Sample configuration files are attached here.

