|  |
| --- |
| トライテック |
| 仮想ボリューム |
| Virtual　Volume |

|  |
| --- |
| 渡辺典孝  2015年5月15日 |

修正履歴

|  |  |  |  |
| --- | --- | --- | --- |
| 日付 | 修正内容 | 修正箇所 | 備考 |
| 2015/05/10 | デレグR->W時の処理の説明  他のクライアントもW要求する場合があるのでデレグをいったん返却し、取得する。 | デレゲーション状態遷移 |  |
| 2015/05/10 | PFSDlg\_tにd\_OwnOldを追加 |  |  |
| 2015/05/10 | デレグ初期化、デレグ終了化関数の説明追加 |  |  |
| 2015/05/10 | DlgCcahe\_tにメンバd\*d\_pVoidの追加 |  |  |
| 2015/05/12 | スプリットブレインの節追加 |  |  |
| 2015/05/15 | 対象資源管理元での排他制御の必要性 |  |  |
|  |  |  |  |

内容

[対象資源管理元での排他制御の必要性 1](#_Toc419450378)

[はじめに 5](#_Toc419450379)

[1. 概要 5](#_Toc419450380)

[2. ソフトウェア構成 6](#_Toc419450381)

[2.1. 概要 6](#_Toc419450382)

[2.2. 計算式 7](#_Toc419450383)

[3. 構成情報 8](#_Toc419450384)

[3.1. システム情報 8](#_Toc419450385)

[3.1.1. システム共通情報 8](#_Toc419450386)

[3.1.2. ノード固有情報 8](#_Toc419450387)

[3.1.3. システム関連テーブル 8](#_Toc419450388)

[3.1.4. Virtual　Volume関連テーブル 9](#_Toc419450389)

[3.1.5. Logical　Volume関連テーブル 10](#_Toc419450390)

[3.1.6. COPY\_LOGテーブル 10](#_Toc419450391)

[4. ボリューム管理 11](#_Toc419450392)

[4.1. スナップショット 11](#_Toc419450393)

[4.1.1. 概要 11](#_Toc419450394)

[4.1.2. 世代の追加と削除 14](#_Toc419450395)

[4.2. ボリュームコピー 16](#_Toc419450396)

[4.2.1. 手順 16](#_Toc419450397)

[4.2.2. ゼロデータ 16](#_Toc419450398)

[5. プール管理 16](#_Toc419450399)

[5.1. 概要 16](#_Toc419450400)

[5.2. LVの移動 16](#_Toc419450401)

[5.3. LVデータの階層管理 17](#_Toc419450402)

[6. 複合IO処理 17](#_Toc419450403)

[6.1. LS単位の統合とB木 18](#_Toc419450404)

[6.2. 通信プロトコル 19](#_Toc419450405)

[6.3. 一括コピー 20](#_Toc419450406)

[6.4. 並列実行 20](#_Toc419450407)

[7. 制御 22](#_Toc419450408)

[7.1. 排他制御構造 22](#_Toc419450409)

[7.2. デレゲーション 22](#_Toc419450410)

[7.2.1. 状態遷移 22](#_Toc419450411)

[7.2.2. デレゲーションの使い方 24](#_Toc419450412)

[7.2.3. キャッシュ化デレゲーションの使い方 28](#_Toc419450413)

[7.2.4. 対象資源管理元での排他制御の必要性 31](#_Toc419450414)

[7.3. Write　throughキャッシュ（readキャッシュ） 31](#_Toc419450415)

[7.3.1. 概要 31](#_Toc419450416)

[7.3.2. デレゲーションの一括宣言/解除 32](#_Toc419450417)

[7.3.3. 64ビットチェックサム 32](#_Toc419450418)

[7.4. メッセージエンジン方式（スナップショット向け） 32](#_Toc419450419)

[7.4.1. メッセージエンジンとメッセージ 32](#_Toc419450420)

[7.4.2. PushFunc/PopFunc 33](#_Toc419450421)

[7.5. スプリットブレイン（ネットワークパーティション） 35](#_Toc419450422)

[8. コマンド 36](#_Toc419450423)

[8.1. 環境 36](#_Toc419450424)

[8.2. LVSの遠隔運転コマンド 36](#_Toc419450425)

[8.3. VVの遠隔運転コマンド 36](#_Toc419450426)

[8.4. 構成DB 37](#_Toc419450427)

[8.4.1. 初期設定 37](#_Toc419450428)

[8.4.2. 参照 37](#_Toc419450429)

[8.5. VVの管理コマンド 37](#_Toc419450430)

[8.5.1. LVの一覧 37](#_Toc419450431)

[8.5.2. LVの作成 37](#_Toc419450432)

[8.5.3. LVの削除 37](#_Toc419450433)

[8.5.4. LSの増量 38](#_Toc419450434)

[8.5.5. VVの一覧 38](#_Toc419450435)

[8.5.6. ボリュームの作成 38](#_Toc419450436)

[8.5.7. スナップショットの作成 38](#_Toc419450437)

[8.5.8. スナップショットの削除 38](#_Toc419450438)

[8.5.9. スナップショットのリストア 38](#_Toc419450439)

[8.5.10. 仮想ボリュームのコピー 38](#_Toc419450440)

[8.5.11. 仮想ボリュームのコピー一覧 38](#_Toc419450441)

[8.5.12. 仮想ボリュームのキャンセル 38](#_Toc419450442)

[8.5.13. 論理ボリュームの移動 39](#_Toc419450443)

[8.5.14. ボリュームのリサイズ 39](#_Toc419450444)

[8.5.15. ボリュームのリネーム 39](#_Toc419450445)

[9. iSCSIのセットアップ 39](#_Toc419450446)

[9.1. Target 39](#_Toc419450447)

[9.1.1. Driverの初期登録 39](#_Toc419450448)

[9.1.2. TgtdVVの起動 39](#_Toc419450449)

[9.1.3. targetの作成 40](#_Toc419450450)

[9.1.4. targetのACL作成 40](#_Toc419450451)

[9.1.5. LUNの作成 40](#_Toc419450452)

[9.1.6. 状態参照 40](#_Toc419450453)

[9.1.7. VVの遠隔操作コマンド 40](#_Toc419450454)

[9.1.8. 定義ファイル 41](#_Toc419450455)

[9.2. Initiator 41](#_Toc419450456)

[9.2.1. 起動 41](#_Toc419450457)

[9.2.2. 停止 41](#_Toc419450458)

[9.2.3. ログイン 41](#_Toc419450459)

[9.2.4. ログアウト 42](#_Toc419450460)

[9.2.5. 管理コマンド 42](#_Toc419450461)

[9.2.6. 接続試験 43](#_Toc419450462)

[10. 性能試験 43](#_Toc419450463)

[10.1. Fio 43](#_Toc419450464)

[10.1.1. スクリプト 43](#_Toc419450465)

[10.1.2. 結果 46](#_Toc419450466)

[10.1.3. 分析 55](#_Toc419450467)

# はじめに

レイヤーは、ReadCache→VS(Virtual　Segment)→Stripe→Mirror→LS（Logical　Segment）とする。

Linuxのdevice-mapperによる実装では、文献（「基幹システムの連続稼動のためのLinux　LVMミラー機能の拡張」IBM）によればmirrorのログの冗長化に問題があると指摘されており、また、snapshotの多数世代での性能低下及びsnapshotのターゲットの冗長化にも問題があるといわれている。

本方式では、mirrorはセルとして冗長化し、snapshotは差分（セルとして冗長化）方式の任意世代数対応とし、また、デレゲーションによるClient　Read　Cacheを導入する。

# 概要

　仮想ボリューム(Virtual　Volume)の仮想セグメント(Virtual　Segment)をストライプ数個の論理ボリューム(Logical　Volume)の論理セグメント(Logical　Segment：VS÷ストライプ数)にマッピングし、ペタバイト(PB)クラスの仮想ボリュームを実現する。ストライプへの割り当ては、仮想セグメントをユニットサイズ（Unit　Size）単位で順次に行う。

　実現にあたっては、ストライプによるストレージの負荷分散/並列処理による高速化とミラーでの冗長化による障害対応を考える。ミラーの実現では、当社のPaxos化キャッシュ技術を導入し、高速化を図る。

仮想ボリューム

A

ミラー

論理ボリューム

B

A

B

A

B

ストライプ

ミラー

仮想ｾｸﾞﾒﾝﾄ

ﾕﾆｯﾄｻｲｽﾞ

論理ｾｸﾞﾒﾝﾄ

論理ボリューム

Virtual　Volume

Logical　Volume

Logical　Segment

Virtual　Segment

　なお、スナップショットはスナップユニットサイズ（ユニットサイズより小さい）単位で管理する。

# ソフトウェア構成

## 概要

　仮想ボリュームの構成情報は、「VV　Manager」で管理する。「VV　Manager」もPaxos技術で構築され、超高可用性を実現している。「CSSセル」は、ネットワークロック、ネットワークメッセージキュー、ネットワークファイル、ネットワーク一時ファイル、イベント等の基本制御を装備している。「CMDBセル」は、メモリ上で動作する簡易RDBMSである。「RASセル」は、資源の死活監視を行う。「RASセル」の障害検知は、RASがCSSと同等なので「CSSセル」と相互監視を行うこともできる。死活の見える化は「監視端末」による。

　「クライアント」と「VV処理」はiSCSIで接続する。「クライアント」はinitiatorであり、「VV処理」はバッキングストアタイプがVVで組み込まれたtargetである。

「VV処理」は、「VV　Manager」から当該VVの構成情報を参照し、VS(Virtual　Segment)の情報を取得する。VS毎にストライプ数個の（LV、LS）が対応する。

「VV処理」では、VSをユニット単位で分割しストライプに分散することで負荷分散および並列高速化を行う。分割データを都度送信すると通信回数が増え著しく性能低下が生じるため、分割されたデータを一括したIO要求で該当「LVS(Logical　Volume　Server)」に転送する必要がある。

「VV処理」では、他の「VV処理」との構成情報の更新の競合を避けるためデレゲーション制御を行う。これにより権限の回収要求がない限りローカルだけで処理が行われる。

この結果、「VV処理」内にもキャッシュを設定できるがwrite　throughでなければならない。

　「LVS」では、キャッシュ管理を行う。時間遅れのWrite　Backで非同期にデータを永続化する。

Target

Initiator

Virtual　Volume

**Logical　Volume　Server**

RASｾﾙ

CMDBｾﾙ

CSSｾﾙ

ｸﾗｲｱﾝﾄ

Block処理

ｷｬｯｼｭ

WB

Block処理

ｷｬｯｼｭ

WB

・・・

VV　Manager

監視端末

**VV処理**

**Logical　Volume**

Virtual　Segment

Logical　Segment(Striped)

Stripe

相互監視

C

A

B

B

A

C

A

C

**Block　Device　I/F**

CSSでも機能的にOKだが、通信ﾌﾟﾛﾄｺﾙがLVSと異なる

Initiator作成時にsdxnを生成

ｷｬｯｼｭ

## 計算式

　VS単位のLSへの対応計算式を示す。

ユニットサイズ UNIT\_SIZE

ユニット数 UNITS

論理セグメントサイズ LS\_SIZE　=　UNITS\*UNIT\_SIZE

ストライプ数 STRIPES

仮想セグメントサイズ VS\_SIZE　=　STRIPES\*LS\_SIZE

とし、UNIT\_SIZEを基本とすると、

仮想ボリュームのoff\_tは、

ユニットインデックス(unit\_index) off\_t/UNIT\_SIZE

仮想セグメントインデックス(vs\_index) unit\_index/(UNITS\*STRIPES)

論理セグメントインデックス(ls\_index) unit\_index　%　STRIPES

論理セグメント内インデックス(lunit\_index) unit\_index　/　STRIPES

　論理セグメント内off\_t lunit\_index\*UNIT\_SIZE+off\_t　%　UNIT\_SIZE

と対応つけられる。

# 構成情報

## システム情報

　各システムに外部から与えられる情報である。

ただし、tgtd組み込みのため、外部情報を引数で与えるのが難しいため環境変数によることとする。

### システム共通情報

　・CSSセル名（環境変数名：CSS）

　・CMDBセル名（環境変数名：DB）

　・ボリューム情報キャッシュ最大数（10固定）

　・ミラーセル情報キャッシュ最大数(100固定)

　・クライアントキャッシュブロックサイズ（環境変数：CRC\_SIZE、128KiB）

　・クライアントキャッシュブロック数（環境変数：CRC\_MAX、2000）

### ノード固有情報

　・名前（環境変数名：VV\_NAME）

　・Id（環境変数名：VV\_ID）

### システム関連テーブル

#### LOCK\_LIST一覧テーブル

　本テーブルは、DBアクセスの排他制御に使用している。排他は、テーブルロックによる。

create table LOCK\_LIST( Name char(64), Cnt int );

#### SEQ\_LIST一覧テーブル

　本テーブルはシーケンサーであり、COPY\_LOGのシーケンサーに使用している。

create table SEQ\_LIST( Name char(64), Seq ulong );

insert into SEQ\_LIST values('COPY\_LOG', 0 );

### Virtual　Volume関連テーブル

#### VN\_LIST一覧テーブル

　ボリュームの属性の一覧管理テーブルである。

VN\_LIST(ボリューム名、ターゲット名、プール名、ストライプ数、最大サイズ、版、参照、参照数、時刻、タイプ、ステータス)

版はボリューム構成を変える毎に更新され、各種のテーブル名（write　table、スナップユニットCOW　table）の識別に用いる。

create table VN\_LIST( VV\_NAME char(64), Target char(64), Pool char(64),

Size ulong, Stripe int, Ver int, Ref int, Time uint,

Type char(16), Status int );

#### VV\_LIST一覧テーブル

　ボリュームの木構造管理テーブルである。

VV\_LIST(ボリューム名、WRテーブル名、COWテーブル名、下位ボリューム名)

　ボリューム名はVN\_LISTのボリューム名と同一である。COWテーブルに登録されているスナップユニット番号に対応するVSがWRテーブルに登録される。COWテーブルに登録されていないスナップユニット番号は下位ボリュームに格納されている。葉ではCOWテーブルが存在しないので、全てがWRテーブルに格納される。

スナップショット操作ではボリューム名がつけ変わる。

create table VV\_LIST(VV\_NAME char(64),

WR\_TBL char(64), COW\_TBL char(64), RD\_VV char(64) );

#### VTテーブル

　VV\_LISTのWRテーブルに相当し、テーブル名は、

VT\_ボリューム\_版(VS、ストライプ番号、LV、LS)

である。

VV毎に作成されるVSセグメント（VS、SI(Stripe　Index)）とLSセグメント(LV、LS)の対応テーブルである。

create table VT\_VOL\_2(VS ulong, SI int, LV char(64), LS uint );

#### COWテーブル

　スナップユニット番号のテーブルである。

VC\_ボリューム名\_版（スナップユニット番号）

create table VC\_VOL\_2(COW\_UNIT ulong);

### Logical　Volume関連テーブル

#### LV\_LIST一覧テーブル

　ボリュームの属性一覧テーブル。

LV\_LIST(LV名、フラグ、LS総数、使用可能LS数、所属プール名、所属LVSセル名)

割り付けは所属プールの使用可能数の多い順にストライプ個確保する。

フラグは割り付け不可フラグである。LV移動時の活性保守のために設けた。

LVはLVSセル名でIOが行われる。

create table LV\_LIST(LV char(64), Flag　uint,Total uint, Usable uint,

Pool char(64), Cell char(64));

#### LVテーブル

LV名（LS番号、フラグ、VT名、VS番号、ストライプ番号）

（LV、LS）と（VV、VS、SI）の対応表である。（VV、VS、SI）は、（LV、LS）をロジカルボリュームレベルで変更する場合に（VV、VS、SI）変更するための逆引き表である。

create table LV\_0(LS uint, Flag uint, WR\_TBL　 char(64), VS ulong, SI int);

### COPY\_LOGテーブル

　ボリュームコピーの度にログレコードを採り、進捗状況の記載、アボート指示を登録する。コピー開始時にシーケンサーから通番を採番しこれによりレコードを同定する。

COPY\_LOG(通番、コピー元ボリューム、コピー元スナップショット、コピー先ボリューム、コピー先スナップショット、コピー元LS数、コピー先LS数、コピー済LS数、開始時刻、終了時刻、ステージ、ステータス)

コピー時にはread　onlyのスナップショットを作成しスナップショット間でコピーする。これによりコピー元ではコピー中も読み書きが可能となる。また、コピー先ボリュームの子はコピー元スナップショットにリンクするのでコピー先でも読み書きが可能である。

コピーには複数段階のステージがあり、COPY\_STAGE\_COMPLETEで終了する。

#define COPY\_STAGE\_CREATE\_SNAP 1

#define COPY\_STAGE\_LINK 2

#define COPY\_STAGE\_COPY1 3

#define COPY\_STAGE\_COPY2 4

#define COPY\_STAGE\_UNLINK 5

#define COPY\_STAGE\_DELETE\_SNAP 6

#define COPY\_STAGE\_COMPLETE 7

#define COPY\_STATUS\_ABORT 0x1000

　COPY\_STAGE\_1/2では、並列転送毎にステータスを調べ、COPY\_STATUS\_ABORTビットが立っていれば、処理を中断する。

　なお、コピー先のLS数はコピー元のLS数以下である。

create table COPY\_LOG( Seq ulong, Fr char(64), FrSnap char(64), To Char(64), ToSnap char(64), FrLSs ulong,

ToLSs ulong, ToCopied ulong, Start uint, End uint, Stage int, Status uint);

# ボリューム管理

## スナップショット

### 概要

　本システムでは、更新前データのCOW方式ではなく、更新データのジャーナル方式を採用する。

また、スナップショットはスナップユニット単位で管理する。

t3

t1

t2

COW

COW

COW

更新

+原

　t1時点以降、最初の変更に対する原データはt1-cowに保存される。すなわち、t1ではt1時点のスナップショットが実現される。t2が指定される。t1からt2の間の変更でt1時点から2回目以降の変更は無視される。t2以降のt1時点から1回目の変更はt1に、2回目以上であればt2にt2時点のデータが現データとしてt2に保存される。t3の指定も同様に初めての保存はt1、2回目の保存はt2、3回目の保存はt3とし、世代という。

　復旧は、tn世代よりの前世代を反映する。tnのCOW、tn-1のCOW、・・・とし最後にデータを反映する。ただし、反映済みに上書きはしない。逆に、データからt1、t2、・・・と反映する場合には上書きをする。なお、存在管理は、ユニット単位のHashで管理する。

（最大容量を1PBとし、ユニット単位を64KBとすると、16M件となり、1.6GB程度の管理領域が必要になる。）

　以上がCOW方式であるが、スナップショット宣言後のwriteデータをジャーナルとして保持するのがシンプルである。

**性能的にも後者が優れている。前者は、更新時にスナップ指定時データを保存の有無をチェックし、データを読み出し、保存した後に更新データを書き込む。読み出し、保存、書き込みの3回のデータ操作がある。これに対し後者のジャーナル方式では1回の書き込みで済む。ただし、ブロックサイズ分のプリフェッチのCOW相当が発生する。**

**また、write　throughのread　cacheを最上位に設定すればネストのreadを抑止できる。**

この場合、スナップショット宣言時に従来データをreadonlyとし、writeを差分ジャーナルとして登録する。同一データの変更は次のスナップショット世代が登録されるまで更新される。次のスナップショットが宣言されるとreadonlyとする。

　これにより、あるスナップショットに対してwriteするような場合にはreadonlyを宣言しwrite差分の分岐をつくることが容易にできる。

t3

t1

t2

W

W

W

原

Read

Write

Read

Write

T4

W

　本方式は、ユーザランドに実装し、デレゲーションで構成を管理するので、COWについてもシンプルな方式を採用する。

　VVは、（更新テーブル、更新ハッシュ、参照テーブルorVV）で構成される。

スナップショットが宣言されていなければ、更新テーブルのみで構成される。

スナップショット宣言時には、COW用のハッシュテーブルと更新テーブルが作られ、参照VVにVV-1が設定される。また、VV-1はreadonlyのVVであり、参照テーブルに原テーブルが設定される。

さらに、スナップショットを宣言するとVV-2が作成される。

(VV、T-0、Null、Null)

↓

(VV、T-1、H-1、VV-1)

(VV-1、Null、Null、T-0)

↓

(VV、T-2、H-2、VV-2)

(VV-2、T-1、H-1、VV-1)

(VV-1、Null、Null、T-0)

　構成情報の変更は、Writeデレゲーション下で行い、Readデレゲーション獲得時に構成情報を更新する。

この方式によれば、スナップショットに対してクローンを定義することができる。

　なお、Hashは任意の単位で管理することができる。VVに対する更新時に、Hashを参照し、なければ参照VVを探索しコピーし、ハッシュ登録後、writeを行う。ただし、単位が小さいと通信回数及びコピー回数が増加するので性能向上が期待できない。逆に、単位が大きいとアクセスされない領域もコピーすることになる。Write時に領域がない場合はVS単位で確保される。

### 世代の追加と削除

* + - 1. 世代の追加

追加

V

S0

S0

V

S1

V

　世代の追加は、現在（V）状態をSnのread　only状態とし、新たに状態を確保し現在状態とする。

* + - 1. 世代の削除統合

反映

Rename

削除

S0

Sn

Sn+1

S’n+1

Sn+1

　世代Snの削除は、Sn+1の状態をSnの状態に反映し、Snの状態をSn+1とし、Sn+1の状態を削除する。

　反映は上書きをするだけでよい。統合は、LS毎に行う。比較単位はスナップユニット単位だが、連続性の判断はLSで行いセグメント化する

　ただし、Sn+1の状態が複数存在する場合は削除統合を行わない。

* + - 1. 世代のrestore

S0

Sn

Sn+1

V

　VへのrestoreはVの状態を初期化し対象Snに繋ぐ。

## ボリュームコピー

### 手順

A

B

F

T

A

B

ﾌﾟｰﾙA

ﾌﾟｰﾙB

コピー

A

B

　ボリュームのコピーについては、コピー元のボリュームについては以下のようにする。

1. コピー元ボリューム、コピー先ボリュームが定義されている。
2. コピー元には、スナップショットと同様に最新世代を定義すると直近世代はread　onlyのボリューム(F)となる。Aには書き込みができる。
3. コピー先も同様にするが、最新世代はFにリンクする。Bについても書き込みができる。
4. FからTにコピーを実行する。
5. 最後に、直近世代ボリュームを削除統合する。

### ゼロデータ

　ゼロデータが展開されないように、コピー元に存在するVSのみについてコピー行う。

# プール管理

## 概要

(未)

## LVの移動

　Logical　Volumeレベルのコピーである。

LS単位のコピーであり、終了後コピー元のLSは未使用にする。

1. コピー元のLVを割り付け禁止にする。

コピー元のLS単位毎にループする。

1. 対応するVVをLockWとする。これによりVVのread/writeが保留される。
2. コピー先のLSを取得する。
3. データをコピーする。
4. コピー先のLSにVS情報をコピー元からコピーする。
5. VSをコピー先のLSで更新する。
6. コピー元は未使用に変更し、LV\_LISTの未使用をインクリメントする。
7. DBをコミットする
8. ロックを解除する。

　以上の手順で、順次に移動ができる。

## LVデータの階層管理

　参入時のインクリメンタルキャッチアップを考慮すると、実ファイルのサイズを一定サイズ以下にする必要がある。

たとえば、40MB/sとすれば10sで400MBとなる。参入時にコピー元を長い時間凍結することはできないので100MB程度が最大と考えらる。

　最大サイズを64ビットとすると、27ビット（100MiB）、10ビット（1Ki）、10ビット（1Ki）、17ビット（128Ki）と分割することができる。すなわち、3階層である。

#define DEFULT\_FILE\_SIZE 　(1024\*128)

#define FILE\_NUMBER (1024)

#define FILE\_PATH(offset,dir0,dir1,base,off)　\

({off\_t　off1;　off=offset%FILE\_SIZE;　off1=offset/FILE\_SIZE;　\

base=off1%FILE\_NUMBER;　off1　/=　FILE\_NUMBER;　\

dir1=off1%FILE\_NUMBER;dir0=off1/FILE\_NUMBER;　})

オフセット

ディレクトリ名

ディレクトリ名

ファイル名

27ビット

10ビット

10ビット

17ビット

　LVSでもファイルを同様な階層管理を行っているのに留意する。

# 複合IO処理

　LS単位内の複数セグメントのIOは一括IO要求とし、通信回数を減らす。

性能は、メモリ内コピー回数と通信回数によるので、通信回数を減らすことにより速度向上ができる。

## LS単位の統合とB木

　VS内の連続IOは、ストライプのユニットサイズで分割され対応するLSに割り付けられる。

LS上は連続になるとしてもVS上では不連続であるためIO自体は分割して行わなければならない。

　また、IOの内部処理単位はユニットサイズより小さいスナップユニットサイズ単位より小さい単位で行うので連続領域の再構築にB木を使う。

破線はコピーしない

LS内のIOを一括処理

B木で連続領域を統合

VS上が不連続

VS

LS

VV\_IO\_t

LS\_IO\_t

LV\_IO\_t

LV\_IO\_t

LS\_IO\_t

LV\_IO\_t

LV\_IO\_t

LS\_IO\_t

LV\_IO\_t

LV\_IO\_t

LV\_IO\_t

VV\_IO\_init()

LS\_IO、LV\_IOの構築

VVKickReqIO()で一括実行

VV\_IO\_destroy()

複合IO

VVKickReqIOCopy()では、複数個のLS\_IO\_t処理ごとに中断チェック

　上記手法により、Read　Cacheのキャッシュにないデータの不連続なIO、スナップショットの複数の小切片IOは自動的に統合される。

typedef struct VV\_IO {

Mtx\_t io\_Mtx;

Cnd\_t io\_Cnd;

HashList\_t io\_LS\_IO;

DHash\_t io\_DHashCOW; // for admin

} VV\_IO\_t;

並列実行制御を行う。

typedef struct LS\_IO {

list\_t l\_Lnk;

VS\_Id\_t l\_VS;

LS\_t l\_LS;

LS\_t l\_LSTo;

BTree\_t l\_ReqLV\_IO;

} LS\_IO\_t;

Read/writeはl\_LSのみ、copyはl\_LSToも設定する。

typedef struct LV\_IO {

list\_t io\_Lnk;

char \*io\_pBuf;

off\_t io\_Off;

size\_t io\_Len;

#define LV\_IO\_FREE 0x1

int io\_Flag;

} LV\_IO\_t;

LS内のio\_Offからio\_Lenが対象データ、io\_pBufがメモリ上のアドレス。Copyのときは使用しない。

## 通信プロトコル

　IO要求は、LS単位に対応するセルに対し、複数のLV\_IO\_tのread/writeを行う。

Read要求1

Read要求2

Read応答1

データ1

Read応答2

データ2

Write要求1

データ1

Write要求2

データ2

Write応答1

Write応答2

Read時は読み込まれたデータをMsg\_tで参照し（コピーをしない）、write時は順次にデータを書き込む。なお、全体の長さは先頭に設定されている。

## 一括コピー

　read/writeの一括IOはバッファ対する操作である。送信時（write）は指定バッファから直接通信路に書き出すことができるが、受信時は受信バッファに読み込みさらに指定バッファに読み込むことになる。

　ストレージ間のコピーは指定バッファなしで、一括readで取得した受信データをメモリコピーせずに一括writeで転送することができる。

## 並列実行

　LS毎に格納されるセルが異なるので、複数のセルに対して並列実行を行うことができる。

そこで、複数のワーカーを用意し、LS\_IO\_tを投入し並列実行を行う。VV\_IO\_tは並列完了同期のために用意した。

ﾜｰｶｰｽﾚｯﾄﾞ

ﾜｰｶｰｽﾚｯﾄﾞ

セル

セル

VV\_IO\_t

LS\_IO\_t/LV\_IO\_t

LS\_IO\_t/LV\_IO\_t

IO実行ｽﾚｯﾄﾞ

投入

完了待ち

完了通知

　IO実行スレッドは、全ての並列実行ワーカースレッドからのIO完了通知を待つ。

ちなみに、IOスレッドも複数である。

# 制御

## 排他制御構造

　VV処理の排他制御構造は以下の通りである。

Volume\_XXX

VL\_open/close

VL\_LockR/VV\_getR ボリュームの排他

VSDLockR セグメント群の排他

VV\_getR VVの排他

DBの排他

VV\_getR

VSDLockR

・・・

　VL\_open/closeはボリュームの初期化open/終了closeである。

IO要求の開始時にVL\_LockRでデレグ排他宣言し、終了時にVL\_Unlockで解除する。

管理コマンドVVAdminはVL\_LockWでIO要求とデレゲーションによる排他制御を行う。

　ボリュームは、連続するセグメント群の塊単位でデレグ排他を行う。これにより同一ボリュームについても、例えば、始めの部分、終わりの部分を別々にデレグ排他制御を行うことができる。

　VV\_getRは、世代管理されたVVの排他に用いる。

## デレゲーション

デレゲーションは、RwLockのネットワークへの拡張であり、マルチプロセス/マルチスレッドでHoldW時に更新、HoldR時に参照を行う。

なお、本デレゲーションは、CSSで管理されるので高可用性が担保されている。

### 状態遷移

　クライアントは、デレゲーションを持っていない場合には、サーバにデレゲーション要求を行い獲得直後に初期処理を行う。Recallによる返却、陽に返却の場合には返却直前に終了処理を行う。

　また、複数クライアントからの更新が競合することが考えられ、ロックにはバージョン（エポック）が設けられ、R->Wで期待されるバージョンでなければ初期処理が呼ばれる。

|  |  |  |  |
| --- | --- | --- | --- |
| Delegation | Return/Recall | HoldR | HoldW |
| 0 | ― | 1.LockR  2.初期処理  ->R | 1.LockW  2.初期処理  ->W |
| R | 1.終了処理  2.Unlock  ->0 | ->R | 1.終了処理  2.LockR->LockW  3.初期処理  ->W |
| W | 1.終了処理  2.Unlock  ->0 | ->W | 1.他のスレッドのRelease待ち  ->W |

0->R,W:HoldR/W

　初めてのデレゲーション獲得であり、それぞれR,Wとし初期処理を行う。

R->W：HoldW

　サーバはPFSLockR中にPFSLockWが発行されると、HoldListからはずし、一人であればWロックとし、HoldListにつなぐ。一人でなければWaitListにつなぎEventRaiseで他のホルダーをRecallする。

　クライアントでは他のスレッドがReleaseするまで待ち、PFSLockWを発行し、RからWに切り替える。なお、PFSLockW発行前に終了処理を呼び出す。また、切り替え後に初期処理を呼び出す。

　Rデレグをいったん返却することになるが、

W->W:HoldR

　Wロックは一人なので何もしない。デレゲーションもWのままである。他人の要求時にはrelease時に自動回収され、

W->W:HoldW

　プロセスがWロックを持っている場合の複数スレッドの制御である。他のスレッドがHold中であればReleaseまで待つ。

W、R->0:Return

　Recall/Return契機で遷移する。RecallはHold中であればReleaseまで待つ。

なお、ロックにはバージョンが設定され、サーバで、Wの発行時と最初のR発行時に、つまり、R/Wの切り替え時にインクリメントしている。したがって、初期処理、終了処理、hold中にデレゲーションのバージョンを確認して処理を行うことができる。たとえば、デレゲーション獲得時に旧版の終了処理を行い、新版の初期化処理を行うこともできる。

### デレゲーションの使い方

　一般に、デレゲーションは分散システムのマスタの状態と分散されたレプリケーションの状態の同期に用いられる。

　本デレゲーション管理では、PFSDlgHoldR/W－PFSDlgRelease間で排他が行われる。デレゲーションを有していなければサーバに対してそれぞれPFSLockR/Wを発行しロック（デレゲーション）を獲得する。デレゲーションは、PFSDlgReturnで返却する。

Rの場合は、複数スレッドが共有できる。Wの場合は１スレッドのみであり、他のスレッドのReleaseを待つ。

回収は、イベントセッションでPFSDlgReturnが呼ばれる。ホルド中であればPFSDlgReleaseまで待たされる。

また、Holdの重複およびホールド中のR/Wの切り替えが可能としている。ただし、PFSDlgReleaseは１回である。

#### PFSDlg\_t　デレゲーション構造体

Typedef　struct　PFSDlg　｛

Int d\_Own; //　0　なし、１(PFS\_DLG\_R）read、2(PFS\_DLG\_W)　write

Int d\_Ver; //　バージョン（エポック）

Int d\_OwnOld; //直前のデレゲーション

・・・

}　PFSDlg\_t;

#### PFSDlgInit　デレゲーション構造体の初期化

int PFSDlgInit( struct Session \*pS, struct Session \*pE,

PFSDlg\_t\*pD, char \*pName,

void(\*fInit)(struct PFSDlg\*,void\*),

void(\*fTerm)(struct PFSDlg\*,void\*) )

pS 本セッション

pE イベントセッション

pD デレゲーション構造体

pName デレゲーションパス名

fInit 初期化関数

fTerm 終了関数

　初期化関数はデレゲーション取得直後、終了化関数はデレゲーション返却直前に呼び出される。

#### PFSDestroy　デレゲーション構造体の破壊

int PFSDlgDestroy( PFSDlg\_t \*pDlg )

pDlg デレゲーション構造体

#### PFSDlgHoldW　Wデレゲーションの宣言

int PFSDlgHoldW( PFSDlg\_t\*pD, void\*pVoid )

pD デレゲーション構造体

pVoid 初期/終了関数への引数

他のクライアントのスレッドを含め自スレッド以外に排他を宣言する。

#### PFSDlgHoldR　Rデレゲーションの宣言

int PFSDlgHoldR( PFSDlg\_t\*pD, void\*pVoid )

pD デレゲーション構造体

pVoid 初期/終了関数への引数

共有排他を宣言する。

#### PFSDlgRelease　デレゲーションの解除

int PFSDlgRelease( PFSDlg\_t\*pD )

pD デレゲーション構造体

デレゲーションを解除する。

本関数はデレゲーションの所有を放棄するだけでデレゲーションをサーバに返却しない。

#### PFSDlgReturn　デレゲーションの返却

int PFSDlgReturn( PFSDlg\_t\*pD, void\*pVoid )

pD デレゲーション構造体

pVoid 初期/終了関数への引数

デレゲーションをサーバに返却する。

#### PFSDlgEventSet/PFSDlgEventCancel　回収イベントの設定/解除

Int　PFSDlgEventSet/Cancel(PDlg\_t　\*pD)

pD デレゲーション構造体

回収イベントの設定解除を行う。

#### PFSDlgRecall　デレゲーションの回収

int PFSDlgRecall( PFSEventLock\_t\*pEv,PFSDlg\_t\*pD, void\*pVoid )

pEv ロックイベント構造体

pD デレゲーション構造体

pVoid 初期/終了関数への引数

#### イベントセッションの設定方法

1. イベントコールバック関数

int

PFSEventCallback( struct Session \*pSession, PFSHead\_t \*pF )

{

switch( (int)pF->h\_Head.h\_Cmd ) {

case PFS\_EVENT\_DIR:

PFSEventDir(pSession,(PFSEventDir\_t\*)pF);

break;

case PFS\_EVENT\_LOCK:

PFSEventLock(pSession,(PFSEventLock\_t\*)pF );

break;

case PFS\_EVENT\_QUEUE:

PFSEventQueue(pSession,(PFSEventQueue\_t\*)pF );

break;

default: Printf("Cmd=%d\n", pF->h\_Head.h\_Cmd ); break;

}

return( 0 );

}

イベント通知はコールバックで呼ばれる。

デレゲーションは、PFS\_EVENT\_LOCKコマンドである。

1. ロックコールバック関数

int

PFSEventLock( struct Session\* pSession, PFSEventLock\_t\* pEvent )

{

\_EventLockGetCount\_ ++;

Printf("[%u:%u]PFSEventLock->Recall:LockName[%s] el\_Ver=%d\n",

(uint32\_t)pthread\_self(), (uint32\_t)time(0),

pEvent->el\_Name, pEvent->el\_Ver );

// find Dlg.

PFSDlgRecall(pEvent, DlgGet(pEvent->el\_Name), NULL );

Printf("[%u:%u]<-Recalled:LockName[%s]\n",

(uint32\_t)pthread\_self(), (uint32\_t)time(0), pEvent->el\_Name );

return( 0 );

}

イベントは、デレゲーション名で通知されるので該当デレゲーション構造体を取得しPFSDlgRecallを呼び出す。PFSDlgRecallは待ち、終了処理後にデレゲーションを返却する。

1. イベントセッションの確保/接続

PaxosSessionAlloc(),PaxosSessionOpen()でイベントセッションを確保接続する

1. イベントスレッドの起動

void\*

ThreadEvent( void \*pArg )

{

struct Session \*pE = (struct Session\*)pArg;

int32\_t count, omitted;

int Ret;

while( 1 ) {

Ret = PFSEventGetByCallback( pE, &count, &omitted,

PFSEventCallback );

Printf("EventGet:Total Count[%d] Ret[%d] Master is %d.\n",

\_EventLockGetCount\_, Ret, PaxosSessionGetMaster( pE ) );

}

pthread\_exit( 0 );

return( NULL );

}

引き渡されたイベントセッションに対しイベント通知を要求する。

#### イベントの設定・解除

Ret = PFSDlgEventSet( &Dlg )

Ret = PFSDlgEventCancel( &Dlg )

PFSDlgEventSet()でイベント受付が開始する。

PFSDlgEventCancel()で解除される。

### キャッシュ化デレゲーションの使い方

　前述のデレゲーションをキャッシュ化し、数多くのデレゲーションに対応し、使い勝手を良くした。

　このため、デレゲーション管理構造体、リコール管理構造体を設け、キャッシュ化デレゲーション構造体を管理することにした。

なお、本デレゲーションは再帰的に使用できる。

ﾘｺｰﾙ管理構造体

ﾃﾞﾚｸﾞ管理構造体

ﾃﾞﾚｸﾞ

ﾃﾞﾚｸﾞ

ﾃﾞﾚｸﾞ

ﾃﾞﾚｸﾞ管理構造体

ﾃﾞﾚｸﾞ

ﾃﾞﾚｸﾞ

#### デレゲーション管理構造体

typedef struct DlgCacheCtrl {

list\_t dc\_Lnk;

GElmCtrl\_t dc\_Ctrl;

struct Session \*dc\_pSession;

struct Session \*dc\_pEvent;

struct DlgCache \*(\*dc\_fAlloc)(struct DlgCacheCtrl\*);

int (\*dc\_fFree)(struct DlgCacheCtrl\*,struct DlgCache\* );

int (\*dc\_fSet)(struct DlgCacheCtrl\*,struct DlgCache\* );

int (\*dc\_fUnset)(struct DlgCacheCtrl\*,struct DlgCache\*);

int (\*dc\_fInit)(struct DlgCache\*,void \*pArg);

int (\*dc\_fTerm)(struct DlgCache\*,void \*pArg);

struct Log \*dc\_pLog;

} DlgCacheCtrl\_t;

　要素保護のためのキャッシュ用関数とデレゲーション初期/終了関数を保持する。

また、操作用のセッションとイベント用のセッションを保持する。

本管理構造体でキャッシュ化デレゲーション構造体を管理する。

#### リコール管理構造体

typedef struct DlgCacheRecall {

list\_t rc\_LnkCtrl;

struct Session \*rc\_pEvent;

uint64\_t rc\_EventCount;

pthread\_t rc\_Thread;

} DlgCacheRecall\_t;

　本管理構造体は、イベントスレッドを管理し、イベント発生時にイベント名で順次デレゲーション管理構造体に問い合わせリコールを実行する。つまり、異なるタイプのデレゲーションのリコールが制御できる。ただし、イベント名はユニークでなければならない。

#### デレゲーション構造体

typedef struct DlgCache {

GElm\_t d\_GE;

DlgCacheCtrl\_t \*d\_pCtrl;

timespec\_t d\_Timespec;

PFSDlg\_t d\_Dlg;

　　Void　　　　　　\*d\_pVoid;

} DlgCache\_t;

　デレゲーションをキャッシュ化で包んだ構造体となっている。

#### リコールの開始と終了

int DlgCacheRecallStart(DlgCacheRecall\_t \*pRC, struct Session \*pEvent);

int DlgCacheRecallStop( DlgCacheRecall\_t \*pRC );

　リコールスレッドを起動し、イベントセッションでイベントを待つ。

#### デレゲーション管理構造体の初期化/終了

int DlgCacheInit( struct Session \*pSession, DlgCacheRecall\_t \*pRC,

DlgCacheCtrl\_t \*pDC, int MaxD,

DlgCache\_t \*(\*fAlloc)(DlgCacheCtrl\_t\*),

int (\*fFree)(DlgCacheCtrl\_t\*,DlgCache\_t\*),

int (\*fSet)(DlgCacheCtrl\_t\*,DlgCache\_t\*),

int (\*fUnset)(DlgCacheCtrl\_t\*,DlgCache\_t\*),

int (\*fInit)(DlgCache\_t\*,void \*pArg),

int (\*fTerm)(DlgCache\_t\*,void \*pArg),

struct Log \*pLog );

int DlgCacheDestroy( DlgCacheCtrl\_t \*pDC );

　デレゲーション管理構造体を初期化し、リコール管理構造体に通知する。

なお、MaxDはキャッシュ最大数である。

#### デレゲーションの取得/解放

DlgCache\_t\* DlgCacheLockR(DlgCacheCtrl\_t\*, char \*pName,

bool\_t bCreate, void \*pTag, void \*pArg );

DlgCache\_t\* DlgCacheLockW(DlgCacheCtrl\_t\*, char \*pName,

bool\_t bCreate, void \*pTag, void \*pArg );

int DlgCacheUnlock( DlgCache\_t \*pD );

それぞれReadデレゲーション、Writeデレゲーションの取得である。

pTagは初めての領域取得時にデレゲーションに設定される。

pArgはfInit、fTermの引数である。

注：pTag、pArgは重複している？

#### デレゲーションの返却

int DlgCacheReturn( DlgCache\_t \*pD, void \*pTerm )

デレゲーションを陽に返却する。デレゲーションを返却する前に終了処理が呼ばれる。

pTermは終了処理の引数である。

### 対象資源管理元での排他制御の必要性

　デレゲーション制御により対象資源を排他制御することができるが、デレゲーションを持っているクライアントAに障害が発生したとき、デレゲーション管理元および対象資源管理元と切断することになる。この時、先にデレゲーション管理元が切断されると、デレゲーション解除を待っていたクライアントBがデレゲーションを獲得し、クライアントAの対象資源管理元との切断の前に対象資源にアクセスする可能性がある。例えば、rollback前にアクセスしてしまう。

　この事態を避けるために、通信回数等が増すが、対象資源管理元での排他制御が必要である。

## Write　throughキャッシュ（readキャッシュ）

### 概要

　VV処理にwrite　throughのread　onlyキャッシュを設け、参照を高速化する。

本キャッシュにはキャッシュ化デレゲーションを用いる。

　ボリュームは、ブロックサイズで分割され、デレゲーションが割り当てられる。

　Read時には、HoldRを行い、キャッシュにあればキャッシュデータを読み込み、なければIO要求を発行してクライアントキャッシュ化する。

　Write時にはHoldWを行い、IO要求を発行して、成功完了後キャッシュにも書き込みをする（Write　through）。Write時の両端のブロックはCOW(CopyOnWrite)となるので、端数があれば読み込みIO要求を前もって行っておく。

　デレゲーションの回収時、返却時にはキャッシュアウトとする。

【検討】太線のキャッシュアウトを回避したいが？

自プロセス

他プロセス

自プロセス

Nバージョン

N+1バージョン

N+2バージョン

他プロセスの回収要求の原因（R）でキャッシュアウトなしを判断すると(W-R-R/W)、他プロセスが故障した時には払い出されるデレゲーションがW-W-R/Wとなる場合がある。この時、回収時にキャッシュアウトしておかないとデータの不整合が発生する可能性がある。

また、回収時にキャッシュアウトをしないで、デレゲーションを再取得した時にW-W-R/Wを判断してキャッシュアウトを行うと同様にデータの不整合が発生する可能性がある。

**結論的には、回収時には必ずキャッシュアウトを行う。省略可能なのは、Wデレゲーション保有時のHoldRのみである。**

### デレゲーションの一括宣言/解除

　デレゲーションはブロック単位毎に定義され、データは不連続にキャッシュ化されているのでIOは不連続的になる。これについて複合IO処理を行うので、複数ブロックのデレゲーションの一括宣言解除を行う必要がある。

　BCReadIO 一括宣言とreadIO作成

　VVRead キャッシュにread

　BCReadDone キャッシュから読み込み、一括解除

　BCWriteIO 一括宣言とCOW両端のreadIO作成

　VVRead キャッシュにread

　VVWrite 書き込み

　BCWriteDone キャッシュに書き込み、一括解除

### 64ビットチェックサム

　ブロック単位毎にチェックサムを計算する。

　Write時にチェックサムを計算し、要チェクサムチェック時刻を設定する。

　Read時刻が要チェックサム時刻を超えていれば、チェクサム計算を行い、チェックし、次の要チェックサムチェック時刻を設定する。

## メッセージエンジン方式（スナップショット向け）

　スナップショット世代数が多いと通常のプログラム方式では、関数のネストが深くなりスタックオーバーフローが発生する。スタックオーバーフローを回避するため、スタックを消費しない関数列によるメッセージエンジン方式を採用する。

### メッセージエンジンとメッセージ

　メッセージエンジンは、メッセージを引数として指定関数を呼び出す。指定関数は処理後次の関数を返却する。メッセージエンジンは返却関数を呼び出す。返却関数がnullであればループを停止する。

int

MsgEngine( Msg\_t \*pM, MsgFunc\_t fNext )

{

MsgFunc\_t fFunc;

pM->m\_Err = 0;

do {

fFunc = fNext;

} while( (fNext = (fFunc)( pM )) != NULL );

if( pM->m\_Err ) return( -1 );

else return( 0 );

}

Main()　{　MsgEngine(pM,Factor);}

typedef struct fact {

int f\_n;

int f\_nn;

} fact\_t;

void\*

Factor( Msg\_t \*pM )

{

fact\_t \*pF = (fact\_t\*)MsgFirst( pM );

if( pF->f\_n ) {

pF->f\_nn \*= pF->f\_n--;

return( Factor );

} else return( NULL );

}

### PushFunc/PopFunc

　メッセージエンジン方式で関数のPush/Popを用意することでネストを回避する。

スタックの縦型展開を横型展開したものとなる。

#define MSG\_FUNC\_ARG( pMsg ) ((pMsg)->m\_pTag2)

typedef struct MsgFuncList {

list\_t fl\_Lnk;

MsgFunc\_t fl\_fFunc;

void \*fl\_pArg;

} MsgFuncList\_t;

int

MsgPushFunc( Msg\_t \*pMsg, MsgFunc\_t fFunc, void \*pArg )

{

MsgFuncList\_t \*pFL;

pFL = (\*pMsg->m\_fMalloc)( sizeof(\*pFL) );

if( !pFL ) goto err;

list\_init( &pFL->fl\_Lnk );

pFL->fl\_fFunc = fFunc;

pFL->fl\_pArg = pArg;

list\_add( &pFL->fl\_Lnk, &pMsg->m\_Lnk );

return( 0 );

err:

return( -1 );

}

MsgFunc\_t

MsgPopFunc( Msg\_t \*pMsg )

{

MsgFuncList\_t \*pFL;

MsgFunc\_t fFunc = NULL;

pFL = list\_first\_entry( &pMsg->m\_Lnk, MsgFuncList\_t, fl\_Lnk );

if( pFL ) {

list\_del( &pFL->fl\_Lnk );

fFunc = pFL->fl\_fFunc;

MSG\_FUNC\_ARG( pMsg ) = pFL->fl\_pArg;

(\*pMsg->m\_fFree)( pFL );

}

return( fFunc );

}

## スプリットブレイン（ネットワークパーティション）

　ネットワーク障害より、サーバが多数派と少数派に別れ、クライアントが少数派に繋がっている場合がある。

クライアント

多数派(Majority)

少数派(Minority)

　サーバでの状態は多数派に引き継がれ、クライアントと同期しているが、非接続状態が長く続くとサーバは強制切り離し処理を行うと、クライアントとマジョリティでズレが生じスプリットブレインとなる。

　サーバが切り離しを行わなければよいのだが、そうするとゴミが溜りいずれシステムを逼迫することになる。

　そこで、クライアントはマスタを検知できない状態が一定程度続いたら状態の破棄を行う。この判断はヘルスチェック時に行う。破棄判断では、セッションを使用不能とする。アプリケーションは使用不能を検知したら再度セッションを取得し、マスタ接続を待ち、要求を送る。このため、破棄判断処理は、最初の接続まで行わない。

# コマンド

## 環境

「VV　Manager」としてCSS、CMDBが起動されていなければならない。

これらは、環境変数CSS、DBにそれぞれ指定する。

　また、ミラーを構成するLVSサーバセルも起動されていなければならない。このセル名はLVを作成するときに（LV名、セル名、プール名）でDBに登録される。

　また、管理コマンド用には、環境変数VV\_NAME（VVの名前）、VV\_ID（Id）を設定する。

## LVSの遠隔運転コマンド

「lvs\_deploy.sh　サーバセル名」で配備をする。

「lvs\_start.sh　サーバセル名」で起動する。

「lvs\_restart.sh　サーバセル名」で再起動する。

「lvs\_stop.sh　サーバセル名」で停止する。

「lvs\_log.sh　サーバセル名　id」でログを取得する。

「lvs\_adm.sh　。。。」は情報取得、制御、停止のためのコマンドである。

　特に、「lvs\_adm.sh shutdown service id comment」は、個々のサーバの強制正常終了であり、終了通番が書き込まれたSHUTDOWNファイルが作成される。サーバ起動時には削除される。したがって、障害でダウンした場合には作成されない。

「lvs\_adm.sh restart service id」は個々のサーバの再起動、「lvs\_restart.sh　service」はセル全体の再起動である。SHUTDOWNファイルの有無で障害と正常終了の判断を行って再起動する。

## VVの遠隔運転コマンド

　セルと同様の定義ファイルに対して実行する。

「vv\_deploy.sh　セル名」で配備をする。

「vv\_start.sh　セル名　id　CSSセル　DBセル」で起動する。

「vv\_stop.sh　セル名　id」で停止する。

「vv\_log.sh　セル名　id」でログを取得する。

「vv\_adm.sh　セル名　id　add　target　volume」はtarget　を追加する。

「vv\_adm.sh　セル名　id　del」はtargetを削除する。

「vv\_adm.sh　セル名　id　show」はtgtdの情報を取得する。

「vv\_adm.sh　セル名　id　ras　ras\_cell　path」はras\_cellのpathにidでras登録を行う。

## 構成DB

### 初期設定

「DropAllDB.sh　DBセル名」でテーブルドロップを行い、「InitDB.sh　DBセル名」でテーブルを作成する。

　その他、必要なテーブルは管理コマンドで自動作成される。

例）「VV-1.sh　lvs-0　pool-1　ras　xjPing」

本スクリプトは、lvs-0のLVSセルにLV-A,B,C,DのLVを宣言し、仮想ボリュームVV-1を宣言する。RasはCSSセル、xjPingはDBセルである。

### 参照

「db\_adm.sh　…」で構成情報を参照する。

## VVの管理コマンド

VVAdminは、メタデータの更新でありtgtdVVとは排他しながら独立に構成情報を変更する。環境変数を設定する。

### LVの一覧

「VVAdmin　ListLV」とする。

### LVの作成

「VVAdmin　CreateLV　LV\_name Cell　Pool　Segments」とする。

LV\_nameは、LV名、

Cellは、LVの属するセル、

Poolは、プール名

SegmentsはLogical　Segment（標準４MiB）数である。

### LVの削除

「VVAdmin　DeleteLV　LV\_name」とする。

使用中であれば削除されない。

### LSの増量

「VVAdmin　IncLV　LV\_name」とする。

LV\_nameのセグメント数を追加する。

なお、使用中の減量は、当面サポートしない。

### VVの一覧

「VVAdmin　ListVV」とする。

### ボリュームの作成

「VVAdmin　CreateVolume　VolName　Target　Size　Stripes　Pool」とする。

ストライプ数（6以下）、プール名、サイズを指定し、ボリュームを作成する。

### スナップショットの作成

「VVAdmin　CreateSnap　VolName　SnapName」とする。

ボリュームのシャップショットを作成する。

### スナップショットの削除

「VVAdmin　DeleteSnap　SnapName」とする。

指定スナップショットを削除する。データは次世代にマージされる。

### スナップショットのリストア

「VVAdmin　RestoreSnap　VolName　SnapName」とする。

指定されたスナップショットをリストアする。

### 仮想ボリュームのコピー

「VVAdmin　CopyVolume　FromVolume　ToVolume」とする。

コマンド投入時点のFromVolumeをToVolumeにコピーする。

### 仮想ボリュームのコピー一覧

「VVAdmin　ListCopy」とする。

COPY\_LOGの一覧が表示される。

次のCancelVolumeCopyの通番はこのコマンドで取得する。

### 仮想ボリュームのキャンセル

「VVAdmin　CancelCopyVolume　Seq」とする。

コピー中であれば、コピーを中断させコピー前の状態に戻す。

### 論理ボリュームの移動

「VVAdmin　MoveLV　FromLV　ToLV」とする。

コマンド投入時点のLVをコピーし、コピー元を開放する。

FromLVを使用しているVVの構成情報がこれに伴いToLVに変更される。

LV単位の活性保守に使用できる。

### ボリュームのリサイズ

「VVAdmin　Resize　Volume　Size」とする。

VN\_LISTのSizeアイテムを変更する。

### ボリュームのリネーム

「VVAdmin　Rename　From　To」とする。

ボリューム名をFromからToに変更する。

# iSCSIのセットアップ

## Target

### Driverの初期登録

　GNUのmain前に実行される、ディレクチブ

\_\_attribute\_\_((constructor))static　void　bs\_VV\_constructor(void)

で初期登録する。

### TgtdVVの起動

　「tgtdVV　–f　–d　1」はforeground、debug起動となる。

　Target、lunはtgtadmで設定する。

つぎの環境変数を設定する。

CSS ・・・Paxos　Core

DB ・・・Paxos　メモリDB

VV\_NAME ・・・本デーモンのクラスタ名

VV\_ID ・・・クラスタ内Id

CRC\_SIZE ・・・Readキャッシュのバッファサイズ（ディフォルト128KB）

CRC\_MAX ・・・Readキャッシュのバッファ数（ディフォルト2000）

　なお、ビルド時にはデバッグ用の「XXXDbg」も生成する。tgtdVVがrootユーザ向けに対して、tgtdDbgは一般ユーザ向けとなる。

### targetの作成

「tgtadm　–L　iscsi　–o　new　–m　target　–t　1　–T　iqn.yyyy-mm.組織:ｼﾘｱﾙ番号」

でSCSIのホストアダプターを作成する。

### targetのACL作成

「tgtadm　–L　iscsi　–o　bind　–m　target　–t　1　–I　ALL」

で全てにアクセス許可。

### LUNの作成

「tgtadm　–L　iscsi　–t　1　–o　new　–m　logicalunit　–l　1　–E　VV」

でtargetにLun1を作成する。

### 状態参照

「tgtadm --lld iscsi --op show --mode target」で参照する。

### VVの遠隔操作コマンド

「vv\_deploy.sh　構成ファイル名　id」で配備する。

「vv\_start.sh　構成ファイル名　id　CSSセル　DBセル」でtgtdVVを起動する。

「vv\_adm.sh　構成ファイル名　id　add　ターゲット名　ボリューム名」でターゲット、ACLを登録する。

「vv\_adm.sh　構成ファイル名　id　del」でターゲットを削除する。

「vv\_stop.sh　構成ファイル名　idl」でtgtdVVを停止する。

「vv\_adm.sh　構成ファイル名　id　show」でターゲット情報を参照する。

「vv\_log.sh　構成ファイル名　id」でPaxos関連ログを参照する。ターゲットの出力はOUTtgtである。

構成ファイル例） \_vv.conf

# id host\_name bin root

0 paxos03 ~/vv /pss/vv/0

1 paxos04 ~/vv /pss/vv/1

2 paxos05 ~/vv /pss/vv/2

その他、「service」を使用することができる。

ただし、環境変数CSS、DBの設定、

Paxos関連ログ

に留意する必要がある。

### 定義ファイル

　/etc/tgt/targets.confで

**<target iqn.2001-03.jp.tritech:tgtd-css>**

**<backing-store vol001>**

**bs-type css**

**lun 1**

**</backing-store>**

**incominguser iscsi\_css iscsi\_css000**

**</target>**

のように指定する。ただし、定義ファイルは、tgt-adminが処理し、tgtadmをパラメータで起動する。

## Initiator

### 起動

「iscsi-start.sh」とする。

―――

#!/bin/sh

/etc/init.d/open-iscsi start

―――

### 停止

「iscsi-stop.sh」とする。

―――

#!/bin/sh

/etc/init.d/open-iscsi stop

―――

### ログイン

「iscsi-login.sh　ホスト名」とする。

―――

#!/bin/sh -x

if [ $# -lt 1 ]

then

echo "USAGE:iscsi-login.sh host"

exit 1

fi

host=$1

TARGET=$(iscsiadm -m discovery -t sendtargets -p $host)

TARGET=$(echo $TARGET | cut -d " " -f 2)

iscsiadm -m node -T $TARGET -l

exit 0

―――

### ログアウト

「iscsi-logout.sh」とする。

―――

#!/bin/sh -x

TARGET=$(iscsiadm -m session)

echo $!

TARGET=$(echo $TARGET | cut -d " " -f 4 )

echo "iscsiadm -m node -T $TARGET -u"

iscsiadm -m node -T $TARGET -u

―――

### 管理コマンド

「iscsi-adm.sh　{show}」

Showコマンドでブロックデバイス名（例えば、sdb－＞/dev/sdb）を確認する。

―――

#!/bin/sh

if [ $# -lt 1 ]

then

echo "USAGE:iscsi-adm.sh {scsi}"

exit 1

fi

iscsiadm -m session -P 3

exit 0

―――

### 接続試験

#### Fdisk

「fdisk　/dev/sdb」とし、p、n、wでパーティションを作成する。

#### Mkfs

Fdiskで作成したパーティション/dev/sdb1、/dev/sdb2に対してファイルシステムを作成する。

「mkfs　–t　ext4　/dev/sdb1」

#### Mount

「mount　/dev/sdb1　mntディレクトリ」

でマウントする。

　アンマウントは「umount」による。

# 性能試験

## Fio

### スクリプト

root@paxos-nw-000:~/vv# cat fio.sh

fio - <<\_EOF\_

[READ\_REMOTE]

rw=read

size=100m

directory=/home/nw/vv/remote/

\_EOF\_

echo

echo

fio - <<\_EOF\_

[READ\_LOCAL]

rw=read

size=100m

directory=/home/nw/vv/local/

\_EOF\_

echo

echo

fio - <<\_EOF\_

[RANDOM\_READ\_REMOTE]

rw=randread

size=100m

directory=/home/nw/vv/remote/

\_EOF\_

echo

echo

fio - <<\_EOF\_

[RANDOM\_READ\_LOCAL]

rw=randread

size=100m

directory=/home/nw/vv/local/

\_EOF\_

echo

echo

fio - <<\_EOF\_

[WRITE\_REMOTE]

rw=write

size=100m

directory=/home/nw/vv/remote/

\_EOF\_

echo

echo

fio - <<\_EOF\_

[WRITE\_LOCAL]

rw=write

size=100m

directory=/home/nw/vv/local/

\_EOF\_

echo

echo

fio - <<\_EOF\_

[RANDOM\_WRITE\_REMOTE]

rw=randwrite

size=100m

directory=/home/nw/vv/remote/

\_EOF\_

echo

echo

fio - <<\_EOF\_

[RANDOM\_WRITE\_LOCAL]

rw=randwrite

size=100m

directory=/home/nw/vv/local/

\_EOF\_

fio - <<\_EOF\_

[RW\_REMOTE]

rw=rw

size=100m

directory=/home/nw/vv/remote/

\_EOF\_

echo

echo

fio - <<\_EOF\_

[RW\_LOCAL]

rw=rw

size=100m

directory=/home/nw/vv/local/

\_EOF\_

fio - <<\_EOF\_

[RANDOM\_RW\_REMOTE]

rw=randrw

size=100m

directory=/home/nw/vv/remote/

\_EOF\_

echo

echo

fio - <<\_EOF\_

[RANDOM\_RW\_LOCAL]

rw=randrw

size=100m

directory=/home/nw/vv/local/

\_EOF\_

### 結果

root@paxos-nw-000:~/vv# ./fio.sh

READ\_REMOTE: (g=0): rw=read, bs=4K-4K/4K-4K, ioengine=sync, iodepth=1

fio 1.59

Starting 1 process

READ\_REMOTE: Laying out IO file(s) (1 file(s) / 100MB)

Jobs: 1 (f=1)

READ\_REMOTE: (groupid=0, jobs=1): err= 0: pid=15341

read : io=102400KB, bw=68085KB/s, iops=17021 , runt= 1504msec

clat (usec): min=0 , max=206446 , avg=51.91, stdev=1332.81

lat (usec): min=1 , max=206446 , avg=52.72, stdev=1332.80

bw (KB/s) : min=47068, max=76064, per=90.43%, avg=61566.00, stdev=20503.27

cpu : usr=1.60%, sys=35.93%, ctx=497, majf=0, minf=26

IO depths : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%

submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

issued r/w/d: total=25600/0/0, short=0/0/0

lat (usec): 2=38.48%, 4=24.13%, 10=0.50%, 20=0.41%, 50=32.54%

lat (usec): 100=2.27%, 250=0.17%, 500=0.40%, 750=0.08%, 1000=0.10%

lat (msec): 2=0.12%, 4=0.72%, 10=0.07%, 20=0.01%, 250=0.01%

Run status group 0 (all jobs):

READ: io=102400KB, aggrb=68085KB/s, minb=69719KB/s, maxb=69719KB/s, mint=1504msec, maxt=1504msec

Disk stats (read/write):

sdb: ios=375/4, merge=1/0, ticks=2052/1072, in\_queue=3128, util=92.57%

READ\_LOCAL: (g=0): rw=read, bs=4K-4K/4K-4K, ioengine=sync, iodepth=1

fio 1.59

Starting 1 process

READ\_LOCAL: Laying out IO file(s) (1 file(s) / 100MB)

Jobs: 1 (f=1)

READ\_LOCAL: (groupid=0, jobs=1): err= 0: pid=15344

read : io=102400KB, bw=74042KB/s, iops=18510 , runt= 1383msec

clat (usec): min=1 , max=10947 , avg=53.40, stdev=418.98

lat (usec): min=1 , max=10948 , avg=53.50, stdev=418.98

bw (KB/s) : min=72246, max=74240, per=98.92%, avg=73243.00, stdev=1409.97

cpu : usr=1.16%, sys=6.08%, ctx=418, majf=0, minf=26

IO depths : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%

submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

issued r/w/d: total=25600/0/0, short=0/0/0

lat (usec): 2=50.22%, 4=46.97%, 10=1.07%, 20=0.11%, 50=0.05%

lat (usec): 100=0.01%, 500=0.01%, 750=0.01%

lat (msec): 2=0.01%, 4=1.20%, 10=0.36%, 20=0.01%

Run status group 0 (all jobs):

READ: io=102400KB, aggrb=74041KB/s, minb=75818KB/s, maxb=75818KB/s, mint=1383msec, maxt=1383msec

Disk stats (read/write):

sda: ios=365/0, merge=0/0, ticks=2420/0, in\_queue=2424, util=92.69%

RANDOM\_READ\_REMOTE: (g=0): rw=randread, bs=4K-4K/4K-4K, ioengine=sync, iodepth=1

fio 1.59

Starting 1 process

RANDOM\_READ\_REMOTE: Laying out IO file(s) (1 file(s) / 100MB)

Jobs: 1 (f=1): [r] [100.0% done] [6972K/0K /s] [1702 /0 iops] [eta 00m:00s]

RANDOM\_READ\_REMOTE: (groupid=0, jobs=1): err= 0: pid=15347

read : io=102400KB, bw=9923.5KB/s, iops=2480 , runt= 10319msec

clat (usec): min=258 , max=384468 , avg=400.05, stdev=3970.52

lat (usec): min=259 , max=384468 , avg=400.31, stdev=3970.51

bw (KB/s) : min= 2928, max=13096, per=101.27%, avg=10048.58, stdev=2638.77

cpu : usr=1.05%, sys=5.27%, ctx=25720, majf=0, minf=24

IO depths : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%

submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

issued r/w/d: total=25600/0/0, short=0/0/0

lat (usec): 500=98.71%, 750=0.97%, 1000=0.12%

lat (msec): 2=0.07%, 4=0.05%, 10=0.03%, 20=0.01%, 50=0.01%

lat (msec): 100=0.01%, 250=0.02%, 500=0.01%

Run status group 0 (all jobs):

READ: io=102400KB, aggrb=9923KB/s, minb=10161KB/s, maxb=10161KB/s, mint=10319msec, maxt=10319msec

Disk stats (read/write):

sdb: ios=25430/18, merge=0/2, ticks=9168/4552, in\_queue=13700, util=88.76%

RANDOM\_READ\_LOCAL: (g=0): rw=randread, bs=4K-4K/4K-4K, ioengine=sync, iodepth=1

fio 1.59

Starting 1 process

RANDOM\_READ\_LOCAL: Laying out IO file(s) (1 file(s) / 100MB)

Jobs: 1 (f=1): [r] [100.0% done] [880K/0K /s] [215 /0 iops] [eta 00m:00s]

RANDOM\_READ\_LOCAL: (groupid=0, jobs=1): err= 0: pid=15350

read : io=102400KB, bw=710686 B/s, iops=173 , runt=147544msec

clat (usec): min=135 , max=29930 , avg=5760.51, stdev=2735.92

lat (usec): min=135 , max=29930 , avg=5760.74, stdev=2735.92

bw (KB/s) : min= 565, max= 988, per=99.99%, avg=693.93, stdev=40.53

cpu : usr=0.06%, sys=1.08%, ctx=25871, majf=0, minf=24

IO depths : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%

submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

issued r/w/d: total=25600/0/0, short=0/0/0

lat (usec): 250=4.03%, 500=1.70%, 750=0.03%, 1000=0.01%

lat (msec): 2=1.58%, 4=21.94%, 10=67.69%, 20=3.02%, 50=0.01%

Run status group 0 (all jobs):

READ: io=102400KB, aggrb=694KB/s, minb=710KB/s, maxb=710KB/s, mint=147544msec, maxt=147544msec

Disk stats (read/write):

sda: ios=25544/14, merge=0/2, ticks=146752/3280, in\_queue=150004, util=99.56%

WRITE\_REMOTE: (g=0): rw=write, bs=4K-4K/4K-4K, ioengine=sync, iodepth=1

fio 1.59

Starting 1 process

WRITE\_REMOTE: Laying out IO file(s) (1 file(s) / 100MB)

WRITE\_REMOTE: (groupid=0, jobs=1): err= 0: pid=15353

write: io=102400KB, bw=538947KB/s, iops=134736 , runt= 190msec

clat (usec): min=4 , max=136 , avg= 6.24, stdev= 3.58

lat (usec): min=4 , max=136 , avg= 6.42, stdev= 3.63

cpu : usr=14.81%, sys=84.66%, ctx=21, majf=0, minf=26

IO depths : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%

submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

issued r/w/d: total=0/25600/0, short=0/0/0

lat (usec): 10=95.83%, 20=2.73%, 50=1.39%, 100=0.05%, 250=0.01%

Run status group 0 (all jobs):

WRITE: io=102400KB, aggrb=538947KB/s, minb=551882KB/s, maxb=551882KB/s, mint=190msec, maxt=190msec

Disk stats (read/write):

sdb: ios=0/0, merge=0/0, ticks=0/0, in\_queue=0, util=0.00%

WRITE\_LOCAL: (g=0): rw=write, bs=4K-4K/4K-4K, ioengine=sync, iodepth=1

fio 1.59

Starting 1 process

WRITE\_LOCAL: Laying out IO file(s) (1 file(s) / 100MB)

Jobs: 1 (f=1)

WRITE\_LOCAL: (groupid=0, jobs=1): err= 0: pid=15356

write: io=102400KB, bw=87223KB/s, iops=21805 , runt= 1174msec

clat (usec): min=3 , max=19550 , avg=44.65, stdev=574.04

lat (usec): min=3 , max=19550 , avg=44.86, stdev=574.04

bw (KB/s) : min=57455, max=114586, per=98.62%, avg=86020.50, stdev=40397.72

cpu : usr=2.73%, sys=11.94%, ctx=165, majf=0, minf=26

IO depths : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%

submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

issued r/w/d: total=0/25600/0, short=0/0/0

lat (usec): 4=0.70%, 10=96.44%, 20=2.03%, 50=0.21%, 100=0.01%

lat (usec): 250=0.01%, 1000=0.01%

lat (msec): 2=0.01%, 4=0.27%, 10=0.24%, 20=0.09%

Run status group 0 (all jobs):

WRITE: io=102400KB, aggrb=87223KB/s, minb=89316KB/s, maxb=89316KB/s, mint=1174msec, maxt=1174msec

Disk stats (read/write):

sda: ios=0/89, merge=0/0, ticks=0/23196, in\_queue=38840, util=46.07%

RANDOM\_WRITE\_REMOTE: (g=0): rw=randwrite, bs=4K-4K/4K-4K, ioengine=sync, iodepth=1

fio 1.59

Starting 1 process

RANDOM\_WRITE\_REMOTE: Laying out IO file(s) (1 file(s) / 100MB)

Jobs: 1 (f=1): [w] [80.6% done] [0K/897K /s] [0 /219 iops] [eta 00m:07s]

RANDOM\_WRITE\_REMOTE: (groupid=0, jobs=1): err= 0: pid=15359

write: io=102400KB, bw=3597.7KB/s, iops=899 , runt= 28463msec

clat (usec): min=4 , max=3671.7K, avg=1110.09, stdev=31052.30

lat (usec): min=5 , max=3671.7K, avg=1110.29, stdev=31052.30

bw (KB/s) : min= 7, max=138581, per=113.65%, avg=4088.13, stdev=22104.51

cpu : usr=0.21%, sys=0.63%, ctx=2106, majf=0, minf=23

IO depths : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%

submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

issued r/w/d: total=0/25600/0, short=0/0/0

lat (usec): 10=85.35%, 20=6.36%, 50=0.48%, 100=0.04%, 250=0.01%

lat (usec): 500=0.01%

lat (msec): 2=0.01%, 4=2.41%, 10=3.34%, 20=1.61%, 50=0.31%

lat (msec): 100=0.03%, 250=0.02%, 500=0.01%, 750=0.01%, 1000=0.01%

lat (msec): 2000=0.01%, >=2000=0.01%

Run status group 0 (all jobs):

WRITE: io=102400KB, aggrb=3597KB/s, minb=3683KB/s, maxb=3683KB/s, mint=28463msec, maxt=28463msec

Disk stats (read/write):

sdb: ios=0/3138, merge=0/1837, ticks=0/688304, in\_queue=508620, util=99.83%

RANDOM\_WRITE\_LOCAL: (g=0): rw=randwrite, bs=4K-4K/4K-4K, ioengine=sync, iodepth=1

fio 1.59

Starting 1 process

RANDOM\_WRITE\_LOCAL: Laying out IO file(s) (1 file(s) / 100MB)

Jobs: 1 (f=1): [w] [61.5% done] [0K/4603K /s] [0 /1124 iops] [eta 00m:05s]

RANDOM\_WRITE\_LOCAL: (groupid=0, jobs=1): err= 0: pid=15362

write: io=102400KB, bw=11573KB/s, iops=2893 , runt= 8848msec

clat (usec): min=4 , max=761920 , avg=343.84, stdev=6034.55

lat (usec): min=4 , max=761921 , avg=344.05, stdev=6034.56

bw (KB/s) : min= 13, max=73168, per=68.27%, avg=7900.62, stdev=17563.24

cpu : usr=0.50%, sys=2.17%, ctx=5662, majf=0, minf=23

IO depths : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%

submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

issued r/w/d: total=0/25600/0, short=0/0/0

lat (usec): 10=75.88%, 20=2.32%, 50=0.19%, 100=0.06%, 250=1.64%

lat (usec): 500=13.06%, 750=1.71%, 1000=1.45%

lat (msec): 2=1.41%, 4=0.45%, 10=1.33%, 20=0.46%, 50=0.01%

lat (msec): 250=0.02%, 500=0.01%, 1000=0.01%

Run status group 0 (all jobs):

WRITE: io=102400KB, aggrb=11573KB/s, minb=11850KB/s, maxb=11850KB/s, mint=8848msec, maxt=8848msec

Disk stats (read/write):

sda: ios=0/7764, merge=0/5234, ticks=0/180932, in\_queue=180928, util=96.63%

RW\_REMOTE: (g=0): rw=rw, bs=4K-4K/4K-4K, ioengine=sync, iodepth=1

fio 1.59

Starting 1 process

RW\_REMOTE: Laying out IO file(s) (1 file(s) / 100MB)

Jobs: 1 (f=1)

RW\_REMOTE: (groupid=0, jobs=1): err= 0: pid=15365

read : io=51008KB, bw=20701KB/s, iops=5175 , runt= 2464msec

clat (usec): min=1 , max=267442 , avg=119.29, stdev=2538.60

lat (usec): min=1 , max=267443 , avg=119.52, stdev=2538.60

bw (KB/s) : min= 3413, max=33832, per=107.80%, avg=22316.33, stdev=16500.37

write: io=51392KB, bw=20857KB/s, iops=5214 , runt= 2464msec

clat (usec): min=3 , max=633338 , avg=69.32, stdev=5592.76

lat (usec): min=3 , max=633339 , avg=69.74, stdev=5592.76

bw (KB/s) : min= 3310, max=33344, per=106.72%, avg=22258.00, stdev=16488.44

cpu : usr=1.14%, sys=8.28%, ctx=311, majf=0, minf=24

IO depths : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%

submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

issued r/w/d: total=12752/12848/0, short=0/0/0

lat (usec): 2=14.97%, 4=36.83%, 10=41.27%, 20=0.99%, 50=2.52%

lat (usec): 100=1.76%, 250=0.76%, 500=0.03%

lat (msec): 4=0.28%, 10=0.53%, 20=0.05%, 100=0.01%, 500=0.01%

lat (msec): 750=0.01%

Run status group 0 (all jobs):

READ: io=51008KB, aggrb=20701KB/s, minb=21198KB/s, maxb=21198KB/s, mint=2464msec, maxt=2464msec

WRITE: io=51392KB, aggrb=20857KB/s, minb=21357KB/s, maxb=21357KB/s, mint=2464msec, maxt=2464msec

Disk stats (read/write):

sdb: ios=233/24, merge=2/0, ticks=2320/4520, in\_queue=6768, util=100.00%

RW\_LOCAL: (g=0): rw=rw, bs=4K-4K/4K-4K, ioengine=sync, iodepth=1

fio 1.59

Starting 1 process

RW\_LOCAL: Laying out IO file(s) (1 file(s) / 100MB)

Jobs: 1 (f=1)

RW\_LOCAL: (groupid=0, jobs=1): err= 0: pid=15368

read : io=51276KB, bw=36212KB/s, iops=9052 , runt= 1416msec

clat (usec): min=1 , max=24029 , avg=87.64, stdev=747.25

lat (usec): min=1 , max=24029 , avg=87.74, stdev=747.25

bw (KB/s) : min=34159, max=37402, per=98.81%, avg=35780.50, stdev=2293.15

write: io=51124KB, bw=36105KB/s, iops=9026 , runt= 1416msec

clat (usec): min=2 , max=5179 , avg=21.35, stdev=241.68

lat (usec): min=2 , max=5179 , avg=21.52, stdev=241.68

bw (KB/s) : min=33729, max=37061, per=98.04%, avg=35395.00, stdev=2356.08

cpu : usr=1.41%, sys=7.07%, ctx=302, majf=0, minf=24

IO depths : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%

submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

issued r/w/d: total=12819/12781/0, short=0/0/0

lat (usec): 2=9.71%, 4=77.54%, 10=11.15%, 20=0.21%, 50=0.05%

lat (usec): 100=0.10%, 250=0.19%

lat (msec): 4=0.52%, 10=0.50%, 20=0.01%, 50=0.01%

Run status group 0 (all jobs):

READ: io=51276KB, aggrb=36211KB/s, minb=37080KB/s, maxb=37080KB/s, mint=1416msec, maxt=1416msec

WRITE: io=51124KB, aggrb=36104KB/s, minb=36971KB/s, maxb=36971KB/s, mint=1416msec, maxt=1416msec

Disk stats (read/write):

sda: ios=235/231, merge=0/0, ticks=1328/2300, in\_queue=3952, util=99.85%

RANDOM\_RW\_REMOTE: (g=0): rw=randrw, bs=4K-4K/4K-4K, ioengine=sync, iodepth=1

fio 1.59

Starting 1 process

RANDOM\_RW\_REMOTE: Laying out IO file(s) (1 file(s) / 100MB)

Jobs: 1 (f=1): [m] [37.5% done] [7979K/8159K /s] [1948 /1992 iops] [eta 00m:05sJobs: 1 (f=1): [m] [50.0% done] [7143K/7180K /s] [1744 /1753 iops] [eta 00m:04sJobs: 1 (f=1): [m] [62.5% done] [7671K/7389K /s] [1873 /1804 iops] [eta 00m:03sJobs: 1 (f=1): [m] [75.0% done] [4718K/5115K /s] [1152 /1249 iops] [eta 00m:02sJobs: 1 (f=1): [m] [77.8% done] [2342K/2416K /s] [572 /590 iops] [eta 00m:02s] Jobs: 1 (f=1): [m] [100.0% done] [9953K/10162K /s] [2430 /2481 iops] [eta 00m:00s]

RANDOM\_RW\_REMOTE: (groupid=0, jobs=1): err= 0: pid=15371

read : io=50740KB, bw=5965.3KB/s, iops=1491 , runt= 8506msec

clat (usec): min=250 , max=1307.5K, avg=643.72, stdev=13777.96

lat (usec): min=250 , max=1307.5K, avg=643.94, stdev=13777.96

bw (KB/s) : min= 1386, max=10872, per=117.39%, avg=7002.38, stdev=2832.57

write: io=51660KB, bw=6073.4KB/s, iops=1518 , runt= 8506msec

clat (usec): min=4 , max=138703 , avg=20.84, stdev=1220.42

lat (usec): min=5 , max=138704 , avg=21.16, stdev=1220.43

bw (KB/s) : min= 1561, max=11240, per=116.58%, avg=7079.69, stdev=2923.12

cpu : usr=0.94%, sys=4.42%, ctx=12775, majf=0, minf=22

IO depths : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%

submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

issued r/w/d: total=12685/12915/0, short=0/0/0

lat (usec): 10=31.02%, 20=17.77%, 50=1.66%, 500=48.92%, 750=0.37%

lat (usec): 1000=0.02%

lat (msec): 2=0.02%, 4=0.02%, 10=0.03%, 20=0.06%, 50=0.05%

lat (msec): 100=0.04%, 250=0.01%, 500=0.01%, 750=0.01%, 2000=0.01%

Run status group 0 (all jobs):

READ: io=50740KB, aggrb=5965KB/s, minb=6108KB/s, maxb=6108KB/s, mint=8506msec, maxt=8506msec

WRITE: io=51660KB, aggrb=6073KB/s, minb=6219KB/s, maxb=6219KB/s, mint=8506msec, maxt=8506msec

Disk stats (read/write):

sdb: ios=12486/487, merge=0/1, ticks=7680/2814864, in\_queue=1248332, util=100.00%

RANDOM\_RW\_LOCAL: (g=0): rw=randrw, bs=4K-4K/4K-4K, ioengine=sync, iodepth=1

fio 1.59

Starting 1 process

RANDOM\_RW\_LOCAL: Laying out IO file(s) (1 file(s) / 100MB)

Jobs: 1 (f=1): [m] [100.0% done] [811K/684K /s] [198 /167 iops] [eta 00m:00s]

RANDOM\_RW\_LOCAL: (groupid=0, jobs=1): err= 0: pid=15374

read : io=51332KB, bw=672225 B/s, iops=164 , runt= 78194msec

clat (usec): min=123 , max=218678 , avg=6073.54, stdev=5135.17

lat (usec): min=124 , max=218679 , avg=6073.76, stdev=5135.17

bw (KB/s) : min= 357, max= 779, per=100.13%, avg=656.86, stdev=65.07

write: io=51068KB, bw=668767 B/s, iops=163 , runt= 78194msec

clat (usec): min=3 , max=13363 , avg=14.07, stdev=244.58

lat (usec): min=4 , max=13363 , avg=14.41, stdev=244.58

bw (KB/s) : min= 302, max= 943, per=99.90%, avg=652.36, stdev=121.29

cpu : usr=0.08%, sys=1.21%, ctx=13007, majf=0, minf=22

IO depths : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%

submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

issued r/w/d: total=12833/12767/0, short=0/0/0

lat (usec): 4=0.01%, 10=40.04%, 20=9.37%, 50=0.41%, 100=0.01%

lat (usec): 250=1.79%, 500=0.92%, 750=0.06%, 1000=0.05%

lat (msec): 2=0.80%, 4=10.72%, 10=33.86%, 20=1.66%, 50=0.23%

lat (msec): 100=0.06%, 250=0.03%

Run status group 0 (all jobs):

READ: io=51332KB, aggrb=656KB/s, minb=672KB/s, maxb=672KB/s, mint=78194msec, maxt=78194msec

WRITE: io=51068KB, aggrb=653KB/s, minb=668KB/s, maxb=668KB/s, mint=78194msec, maxt=78194msec

Disk stats (read/write):

sda: ios=12800/1455, merge=0/106, ticks=77376/6431316, in\_queue=7120020, util=99.71%

root@paxos-nw-000:~/vv#

### 分析

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Bw(KB/s) | Iops(1/s) | Run(ms) | Lat(平均us) | Lat(標準偏差) |
| READ\_REMOTE | 68085 | 17021 | 1504 | 51.91 | 1332.81 |
| READ\_LOCAL | 74042 | 18510 | 1383 | 53.40 | 418.98 |
| RANDOM\_READ\_REMOTE | 9923.5 | 2480 | 10319 | 400.05 | 3970.52 |
| RANDOM\_READ\_LOCAL | 711 | 173 | 147544 | 5760.51 | 2735.92 |
| WRITE\_REMOTE | 538947 | 134736 | 190 | 6.24 | 3.58 |
| WRITE\_LOCAL | 87223 | 21805 | 1174 | 44.65 | 574.04 |
| RANDOM\_WRITE\_REMOTE | 35978 | 899 | 28463 | 1110.09 | 31052.30 |
| RANDOM\_WRITE\_LOCAL | 11573 | 2893 | 8848 | 343.84 | 6034.55 |
| RW\_REMOTE　　　　read  　　　　　　　　　　　write | 20701  20857 | 5175  5214 | 2464  2464 | 119.29  69.32 | 2538.60  5592.76 |
| RW\_LOCAL | 36212  36105 | 9052  9026 | 1416  1416 | 87.64  21.35 | 747.25  241.68 |
| RANDOM\_RW\_REMOTE | 5965  6073 | 1491  1518 | 8506  8506 | 643.72  20.84 | 13777.96  1220.42 |
| RANDOM\_RW\_LOCAL | 672  668 | 164  163 | 78194  78194 | 6073.54  14.07 | 5153.17  244.58 |

　リモートはローカルと比べて遜色なく、むしろ優勢である。