第1版 　 2015年8月25日

第1.4版　2016年3月

クライアント（Libmemcached）とサーバー（memcached）の間のクライアントサイドにPaxosMemcacheを挿入し、さらなる高速化を図る。

PaxosMemache

内容

[1. はじめに 3](#_Toc447011146)

[2. Memcachedの問題点とPaxosMemcache 3](#_Toc447011147)

[2.1. Memcachedの問題点 3](#_Toc447011148)

[2.2. PaxosMemcache 4](#_Toc447011149)

[2.3. ロードマップ 5](#_Toc447011150)

[3. 構成 6](#_Toc447011151)

[4. 機能 7](#_Toc447011152)

[4.1. キーの唯一性 7](#_Toc447011153)

[4.2. キーとメタデータファイル名（MD5ハッシュ値） 7](#_Toc447011154)

[4.3. メタデータとレプリケーション 7](#_Toc447011155)

[4.4. WB（ライトバック） 9](#_Toc447011156)

[4.5. デレゲーション所有者 9](#_Toc447011157)

[4.6. 応答（Fake） 10](#_Toc447011158)

[4.7. 対 10](#_Toc447011159)

[4.8. 接続管理（コネクションプール） 10](#_Toc447011160)

[4.9. 「no　reply」と「quiet」 10](#_Toc447011161)

[4.10. Incrementとdecrement 11](#_Toc447011162)

[5. データ構造と処理フロー 11](#_Toc447011163)

[5.1. コネクションプール 11](#_Toc447011164)

[5.2. 属性 11](#_Toc447011165)

[5.3. Itemキャッシュ 12](#_Toc447011166)

[5.4. メッセージ 13](#_Toc447011167)

[5.4.1. テキスト 13](#_Toc447011168)

[5.4.2. バイナリ 14](#_Toc447011169)

[6. コマンド 14](#_Toc447011170)

[6.1. PaxosMemcache 14](#_Toc447011171)

[6.2. PaxosMemcacheAdm 15](#_Toc447011172)

[6.3. 運用 16](#_Toc447011173)

[6.3.1. 構成ファイル 16](#_Toc447011174)

[6.3.2. 遠隔操作コマンドシェル 16](#_Toc447011175)

[6.3.3. libmemcachedの取得 17](#_Toc447011176)

[6.3.4. memcachedの取得 17](#_Toc447011177)

[7. 試験 17](#_Toc447011178)

[7.1. 環境 17](#_Toc447011179)

[7.2. 性能 19](#_Toc447011180)

# はじめに

　「libmemcached」をAPIとするアプリケーションに対し、デレゲーションによるクライアント分散キャッシュを「memcached」との通信回路に「PaxosMemcach」を挿入することにより実現する。

memcached通信プロトコルはクライアントでスヌープされる。

スヌープ対象は、storageとretrieve、その他更新を伴うコマンドである。その他のコマンドは原則スルーとなる。

　データがクライアントにキャッシュされるので参照優勢のデータアクセスが高速化できる。複数クライアントに跨る分散キャッシュは、Paxosを使用した高可用性デレゲーションにより、初めて実現された[[1]](#footnote-1)。

さらに、類似商品を超えるため、自動レプリケーション機能、WB機能、コネクションプール機能を実装した。これにより、コヒーレンス、高可用性、高速化が実現される。

# Memcachedの問題点とPaxosMemcache

## Memcachedの問題点

　MemcachedはWebアプリケーション用にRDBMSアクセス等の高いcpuコストを軽減するためのキャッシュ機構として導入され広く利用されている。たとえば、プロファイル、認証等のユーザ情報、写真等のメディアのメタ情報、期間つきのセッション情報、静的ページやタグ等の一般的情報などである。

Memcachedは、単純な機能しかサポートしないので、複数のサーバーを前提としてクライアント側で分散格納によるスケールアウトやレプリケーションによる耐障害性を実現している。

しかし、次のような問題が指摘されている。

1. Cold　cache

memcachedノードが使用不能となったとき、新しいノードでキャッシュを再構築しなければならない。

1. スケールアウトに難がある

　個々のサーバーにRAMを追加する（scale-up）よりもサーバーの数を増やす(scale-out)のが普通である。サーバーの増減ではキーの再マップが必要となり、このときクライアントはデータを取得できなかったり古いデータを取得したりする。また、キーを再構築したりする。

1. 死活監視に難がある

　Memcachedはお互いに独立し相互監視ができないので監視しようとすると複雑になる。

1. 古いデータアクセス

　キーに関する厳密な最新情報が存在しないので、ネットワーク障害が発生すると、同じキーで複数のデータが分散され、クライアントが古いデータを参照する場合がある。

1. レプリケーションの整合性

Libmemcached等のクライアントでレプリケーションを実現することになるが、ネットワーク排他制御がないので、レプリケーション間でデータ不整合が発生する。

1. 接続数

　クライアントが1k、10kになると目一杯となる。

## PaxosMemcache

　PaxosMemcacheはクライアントキャッシュを導入し高速化を図ることを目的としている。このために、クライアント間の同期制御のためのデレゲーション機構を実装している。

　この結果、コヒーレンスが保持されることになり、前述の問題のいくつかが自動的に解決される。（①はクライアントキャッシュで軽減、②の古いデータ、③はデレゲーションとしての部分、④全部）

　キーの属性情報をCSSで共有管理し、exclusiveなwriteデレゲーションで更新管理する。この結果、全てのレプリケーションは最新となり、古いデータアクセス問題が発生しない。

　また、PaxosMemcacheでメタ情報管理ができるので、レプリケーションの自動認識、更新時のWBを実現できる。さらに、PaxosMemcacheはクライアントとmemcachedに挿入されるので接続のプール管理ができる。

## ロードマップ

ｸﾗｲｱﾝﾄ（APｻｰﾊﾞ）

DB

Memcached

Memcachedﾌﾟﾛﾄｺﾙ

現状

ｸﾗｲｱﾝﾄ（APｻｰﾊﾞ）

DB

PaxosMecache

Memcachedﾌﾟﾛﾄｺﾙ

ﾊｲﾌﾞﾘｯﾄﾞ(PaxosMemcache)

Memcached

Memcachedﾌﾟﾛﾄｺﾙ

CSS

Memcached

Memcached

ﾈｯﾄﾜｰｸ排他制御

Paxos化（PaxosKVS）

ｸﾗｲｱﾝﾄ（APｻｰﾊﾞ）

DB

PaxosMecache

Memcachedﾌﾟﾛﾄｺﾙ/

独自ﾌﾟﾛﾄｺﾙ

CSS

ﾈｯﾄﾜｰｸ排他制御

KVS(CSS)

ﾚﾌﾟﾘｹｰｼｮﾝ

　性能及び信頼性向上のための工程は上手のようである。

今回提供のハイブリッド型は、高速化のみならず、クライアントで実装されているレプリケーション機能を、データのレプリケーションを自動判定することにより対応している。

　PaxosKVSは、クライアントのレプリケーション機能をPaxosセルで実現し、クライアントの負荷を軽減する。デレゲーション用のCSSを用いることもできる。

# 構成

ｸﾗｲｱﾝﾄAP

Libmemcached

PaxosMemcache

Memcached

　PaxosMemcacheはクライアントサイド（できれば同一マシン）に搭載する。

PaxosMemcacheには、（PaxosMemcacheのlistenアドレス、Memcachedのlistenアドレス）の対を複数登録できる（ポートはTCPである）。

Memcachedとの接続はプール管理をおこなう。クライアントの接続要求時にプールから取り出す。プールに存在していないときにはmemcachedに接続する。存在する場合には参照カウンタをインクリメントする。

ソフトウェア構成の詳細は次図である。

AF\_INET

ｸﾗｲｱﾝﾄAP

Libmemcached

Memcached

Memcached

Memcached

ｸﾗｲｱﾝﾄAP

Libmemcached

CSS

AF\_INET

元データ

PaxosKVS

マシン

PaxosMemcache

　挿入されたread　cacheを「PaxosMemcache」（斜線部）と称する。このキャッシュはCSSのデレゲーションで保護される。すなわち、更新時にはキャシュ破棄が行われデレゲーションが回収される。参照時にデレゲーションが取得されキャッシュ化される。

# 機能

## キーの唯一性

　個々のPaxosMemcache内ではキーはユニークであるとする。

PaxosMemcacheは名前空間（space）付で起動され、キーは名前空間内でユニークである。したがって、異なるクライアントノードでPaxosMemcacheを同じ名前空間で起動することにより同じキーをアクセスする。

　同じクライアントノードで異なる名前空間にアクセスする場合は、複数のPaxosMemcacheを起動する。

## キーとメタデータファイル名（MD5ハッシュ値）

　キーには、”/”を含んだパス等（ディフォルト：最大1024バイト）を設定することができる。このキーに対応したファイル名（ディフォルト：最大256バイト、含\0終端）にキーのMD5ハッシュ値（33バイト、含\0終端）を採用する。

　MD5では、hash(a)=hash(b)となる異なるa、ｂはまれであるとされる。

実際、MD5は128ビットであり2\*\*128個識別できる。

　2\*\*128=(2\*\*5)\*\*123=32\*\*123

から、アスキー文字数を約32とすると128文字桁数に相当する。

一様に対応づけられれば128文字桁数は区別できる。さらに、キーがURL等のパス名であれば、basenameのみが異なるのが一般的であり、個数も2\*\*128は極めて大きいので充分に使用できる。

## メタデータとレプリケーション

　更新のレプリケーション対象には、set、add、replace、append、prepend、cas、incr、decr、delete等がある。

APPサーバー（クライアント）

Memcached

Memcached

Memcached

PaxosMemcache

統合属性

格納ｻｰﾊﾞｰﾘｽﾄ（α、β、γ）

ﾃﾞｰﾀ長

ﾁｪｯｸｻﾑ

Expire時刻

ﾒﾀCAS

ｴﾎﾟｯｸ

個別属性(Memcached)

　ﾌﾗｸﾞ

ﾃﾞｰﾀ長

ﾁｪｯｸｻﾑ

Expire時刻

ﾒﾀCAS

CAS(Memcached)

ｷｬｯｼｭﾃﾞｰﾀ

差分

　データについては、PaxosMemcacheはメタデータである属性データと全レプリケーションのキャッシュデータを持つことができるとする。このキャッシュデータの管理にメタCASを用いる。

クライアントからの要求は、対象memcachedに対して行われる。

* 1. キーに属性が登録されていなければ、新規である。
  2. 対象memcachedがデータを有し、個別メタCASが統合属性の統合メタCASと一致していれば更新と判断する。更新時には、統合メタCASをインクリメントし個別メタCASにも設定する。
  3. 個別メタCASと統合メタCASが一致していなければレプリケーションかを判断する。とりあえず、個別メタCASが一世代前であれば、レプリケーションとする。それ以外はエラーとする。
  4. 対象memcachedがデータを有していなければ、そして他のmemcachedがデータを有していれば、新規に登録されるレプリケーションの可能性があると判断する。データ内容を調べ一致していればレプリケーションとする。そうでなければ、エラーとする。
  5. Append、prependの場合は、続くAppend、prependがレプリケーションとして判断される。この時、本体データがなければ取得し、全体データとしてレプリケーションチェックを行う。
  6. Casコマンドは、個別casに変換する。
  7. Incr、decrのvalueは差分として統合属性に記録する。

## WB（ライトバック）

　Memcachedはキーがあれば最新のデータが取り出されるReadCacheである。キーがなければ最新のデータを登録することになる。また、キーも任意に破棄される。

　そこで、WB(WriteBack)を実装することを考える。

WBはあるクライアントのW要求がサーバーにデータを書き終わる前に応答を返し、その後実際にサーバーにデータを書き込む。サーバーへの書き込みが失敗した場合、他のクライアントがデータを読み込めないようにしなければならない。一般に、読み込めないこのクライアントはDBから最新データを取得し、登録に来ることになる。

　Wデレゲーションに関しては、hold後、FakeReplyを返し、WBでreleaseすることになる。FakeReplyを返すスレッドとWBのスレッドは異なるので制御が難しくなる。このために、デレゲーションに所有者を指定できるようにする。

　また、複数クライアントの同期制御ができなければならない。これはWデレゲーションを用いる。Wデレゲーションを獲得した後に、応答をクライアントに返し、サーバーに書き終わった時点でWデレゲーションを解放する。Wデレゲーション獲得中の異常終了ではキーが存在しないようにしなければならない。このため、デレゲーション名のファイルを想定し、異常終了時には強制削除するようにする。サーバーの異常はPaxosMemcacheが検知しファイルを削除する。PaxosMemcacheの異常切断はCSSが検知し自動的にファイルを削除する。

　なお、通信回数を減らすために、データにepochを導入する。Epochは、差分では更新されず、set/replaceでインクリメントされる。メッセージにもepochを記録し、WB時にepochが異なれば省略、破棄する。これにより、不必要なW要求通信を抑止する。

## デレゲーション所有者

　デレゲーションの所有者はスレッドとしていたが、スレッド以外も指定できるようにした。これにより、異なるスレッドでのhold/releaseが可能になり、要求スレッドでhold、WBでreleaseが可能となる。

　所有者は、クライアントの接続構造体とし、要求はリストでシリアライズされる。要求は順次処理されるが、WBが発生するとWBと同期の２つのシリアラズ列が発生する。このため、WB依頼を行わない要求をWB列の終了まで待ち合わせる。

　Rデレゲーション取得時も実体がWデレゲーションであることもあるのでWB列の終了を待つ必要がある。

ﾃﾞﾚｹﾞｰｼｮﾝ

CSS

Prologue

WB

同期処理

応答

応答

Epilogue

所有者

所有者

所有者

WB

Yes

No

所有者がおなじであれば重複ロック

## 応答（Fake）

　Write時は、キャッシュを更新し応答（Fake）を返し、WB依頼を行う。

　Read時には、クライアントキャッシュにデータがあれば、指示されているサーバーの如何に拘わらず、応答が返却される。クライアントはデータが指示したサーバーからデータが得られたと判断する。

## 対

　登録対は、（libmemcachedに対するPaxosMemcacheのlistenアドレス、memcachedのlistenアドレス）で構成される。

登録対はPaxosMemcacheのlistenアドレスでユニークとする。したがって、memcachedのlistenアドレスは一つで固定される。ただし、一つのmemcachedのlistenアドレスを複数の対に登録することができる。

## 接続管理（コネクションプール）

　Memcachedに対する接続管理は、コネクションプールによる。

## 「no　reply」と「quiet」

　「no　reply」はテキストプロトコルで応答を全く必要としない、「quiet」はバイナリプロトコルで正常であれば応答を必要としないが異常時には応答する、の意である。

　更新系の場合、クライアントは、memcachedまで問い合わせ、正常を確認した後、キャッシュ化することになる。「no　reply」では正常か異常かが不明である。「quiet」では異常の場合にエラーが返ってくる。

　参照系の場合でも、クライアントはキャッシュにない場合には、memcahedに問い合わせ、正常に取得した後、キャッシュ化することになる。

　両者とも、memcacheでは正常の確認が必要なため、「no　reply」、「quiet」なしのプロトコルに変換してmemcachedに問いあわせ、応答を解析し、バイナリの場合には「quiet」に変換して応答をクライアントに返すようにする。

## Incrementとdecrement

　本コマンドは、データをuint64\_tとみなし、値の増減を行う。

　したがって、クライアントキャッシュをパージアウトし、サーバーからの応答をクライアントに返却する。

また、クライアントキャッシュ化は行わない。次のget/setでキャッシュ化する。

# データ構造と処理フロー

## コネクションプール

typedef struct MemCachedCon {

GElm\_t m\_Elm;

int m\_Fd;

int m\_FdBin;

struct sockaddr\_in m\_Addr;

Mtx\_t m\_Mtx;

Cnd\_t m\_Cnd;

} MemCachedCon\_t;

　Memcachedとの接続には、アスキー(m\_Fd)とバイナリ(m\_FdBin)の回路を開設する。

## 属性

typedef struct MemItemAttr {

struct sockaddr a\_MemCached;

#define MEM\_ITEM\_CAS 0x1

#define MEM\_ITEM\_ACIVE 0x2

int a\_Status;

uint32\_t a\_Flags;

uint64\_t a\_Bytes;

uint64\_t a\_MetaCAS;

uint64\_t a\_CAS;

timespec\_t a\_ExpTime;

timespec\_t a\_CksumTime;

uint64\_t a\_Cksum64;

Msg\_t \*a\_pData;

Msg\_t \*a\_pDiff;

} MemItemAttr\_t;

typedef struct MemItemAttrs {

uint64\_t a\_MetaCAS;

uint64\_t a\_Epoch;

int a\_I; // current

int a\_N; // array size

MemItemAttr\_t a\_aAttr[1];

} MemItemAttrs\_t;

　個別属性、統合属性がある。

統合属性にはクライアント向けのメタCAS、WB背魚用の更新エポックがある。なお、エポックは要求メッセージに付与される。

統合属性の配列a\_aAttr[0]をカレントとする。要求の度に配列要素を入れ替える。

## Itemキャッシュ

typedef struct MemItem {

DlgCache\_t i\_Dlg;

Mtx\_t i\_Mtx;

char \*i\_pKey;

MemItemAttrs\_t \*i\_pAttrs;

#define i\_pData i\_pAttrs->a\_aAttr[0].a\_pData

#define i\_Status i\_pAttrs->a\_aAttr[0].a\_Status

#define i\_Flags i\_pAttrs->a\_aAttr[0].a\_Flags

#define i\_Bytes i\_pAttrs->a\_aAttr[0].a\_Bytes

#define i\_CAS i\_pAttrs->a\_aAttr[0].a\_CAS

#define i\_MetaCAS i\_pAttrs->a\_aAttr[0].a\_MetaCAS

#define i\_ExpTime i\_pAttrs->a\_aAttr[0].a\_ExpTime

#define i\_CksumTime i\_pAttrs->a\_aAttr[0].a\_CksumTime

#define i\_Cksum64 i\_pAttrs->a\_aAttr[0].a\_Cksum64

#define WB\_WORKER 0x1

int i\_FlagWB;

list\_t i\_MsgList;

} MemItem\_t;\* Item attruiute（to　be　done） \*/

　データはi\_pDataに設定され、DlgCache\_tで保護される。テキストとバイナリで操作に必要な共通データ（i\_Flags、i\_CAS、i\_Bytes）も保持される。統合属性との比較のためのi\_MetaCASも保持する。

　エクスパイア時刻（i\_ExpTime）は、参照時にこの時刻を過ぎていれば、memcachedに問い合わせ、存在していなければキャッシュを破棄し、存在していればエキスパイア時刻を更新する。

　チェックサム時刻(i\_CksumTime)も同様に、参照時にこの時刻を過ぎていればチェックサム計算を行いメモリ障害の有無を調べる。障害時にはアボートする。

## メッセージ

### テキスト

#define TOKEN\_BUF\_SIZE 1024

typedef struct CmdToken {

char \*c\_pCmd;

char \*c\_pKey;

char \*c\_pFlags;

char \*c\_pExptime;

char \*c\_pBytes;

char \*c\_pNoreply;

char \*c\_pCAS;

char \*c\_pValue;

char \*c\_pSave;

uint64\_t c\_MetaCAS;

uint64\_t c\_Epoch;

char c\_Buf[TOKEN\_BUF\_SIZE];

} CmdToken\_t;

　テキスト入力のコマンド行は、トークン分けされ、上記構造体に格納され、この構造体のポインタはMsg\_tのm\_pTag1に登録される。

　ボディデータはMsg\_tにVec\_tで格納される。

### バイナリ

typedef struct {

uint8\_t h\_Magic;

uint8\_t h\_Opcode;

uint16\_t h\_KeyLen;

uint8\_t h\_ExtLen;

uint8\_t h\_DataType;

union {

uint16\_t req\_VbucketId;

uint16\_t rpl\_Status;

} h\_Vbucket\_Status;

#define h\_VbucketId h\_Vbucket\_Status.req\_VbucketId

#define h\_Status h\_Vbucket\_Status.rpl\_Status

uint32\_t h\_BodyLen;

uint32\_t h\_Opaque;

uint64\_t h\_CAS;

} BinHead\_t;

typedef struct {

BinHead\_t t\_Head;

char \*t\_pExt;

char \*t\_pKey;

uint64\_t t\_MetaCAS;

uint64\_t t\_Epoch;

// follow Ext and Key

} BinHeadTag\_t;

　バイナリデータは、extra　data及びkey　dataまで、BinHeadTag\_tに格納され、ポインタがMsg\_tのm\_pTag2に設定される。同時に、extra　dataおよびkey　dataのポインタがそれぞれt\_pExt、t\_pKeyに設定される。

　ボディデータはMsg\_tにVec\_tで格納される。

# コマンド

## PaxosMemcache

コマンド

「PaxosMemcache　[オプション]　id　css　space」

Id 数字

Css デレゲーション管理のcssセル

Space キーの名前空間

オプション

-I　ddd キャッシュ化アイテム最大数（ディフォルト：2000）

-s　{1|0} チェックサムの有無（ディフォルト：有）

-C　ddd チェックサムインターバル（ディフォルト：5s）

-E　ddd エクスパイアチェックインターバル（300s）

-W　d ワーカー数（ディフォルト：5）

説明

　Id（管理用ポート）とcssセル名お呼び名前空間で起動する。起動後、cssセルに接続し、管理用ポートを開設し、コマンドを待つ。キーは名前空間でユニークである。

　対の登録指示で対を登録するとともに、クライト受け付け用のlistenポートを開設する。

クライアントからの本ポートへの接続で接続構造体を払い出し、対応するmemcachedに接続する。

## PaxosMemcacheAdm

本コマンドは、PaxosMemcacheの管理ポートを介して、対の登録削除、情報取得を行う。

コマンド

「PaxosMemcacheAdm Id add client[:port] server[:port]」

「PaxosMemcacheAdm Id del client[:port] server[:port]」

「PaxosMemcacheAdm Id event」

「PaxosMemcacheAdm Id item」

「PaxosMemcacheAdm Id pair」

「PaxosMemcacheAdm Id stop」

「PaxosMemcacheAdm Id log」

「PaxosMemcacheAdm Id ras　cell」

「PaxosMemcacheAdm Id unras　cell」

説明

Add/del 対の登録削除

Event イベント監視情報の取得

Item アイテムキャッシュ情報の取得

Pair 対情報の取得

Stop PaxosMemcacheプロセスの停止

Log バッファリングされているログを出力する

Ras/unras RASセル(cell)に監視対象の登録削除を行う。

## 運用

### 構成ファイル

例えば、ホームディレクトリ下に「~/\_Memcache.conf」の構成ファイルを配置する。

サービス名（Mecache）が名前空間となる。セル名は、shell引数入力で行う。

#

#　PaxoxMemCache稼動ホスト

#　id　host　bin　root

0 paxos03 　~/Memcache　 /pss/Memcache/0

1 paxos04 　~/Memcache 　/pss/Memcache/1

2 paxos05 　~/Memcache 　/pss/Memcache/2

説明

Host ホスト名

Id 管理ポート

Bin ロードモジュールの配置先

Root ログ、データ等（未使用）

### 遠隔操作コマンドシェル

|  |  |
| --- | --- |
| mem\_deploy.sh　conf | ホームディレクトリ下の構成ファイル（conf）に従って、ディレクトリの作成、コマンド配布、構成ファイル配布を行う。 |
| mem\_start.sh　conf　[id] | PaxosMemcacheを遠隔起動する。  Idが指定されてなければ全てを起動する。 |
| mem\_stop.sh　conf　[id] | PaxosMemcacheを遠隔停止する。  Idが指定されてなければ全てを停止する。 |
| mem\_adm.sh shutdown conf | PaxosMemcacheを遠隔停止する。 |
| mem\_probe.dh　{event|item|pair}　conf　id | Idのイベント/アイテム/対情報を参照する。 |
| mem\_adm.sh add conf [local:]port server[:port] | PaxosMemcacheに対を登録する。Localが指定されていなければ当該hostとなる。 |
| mem\_adm.sh del conf [local:]port server[:port] | PaxosMemcacheに対を削除する。Localが指定されていなければ当該hostとなる。 |
| mem\_adm.sh pair conf id | Idの対情報を参照する。 |
| mem\_adm.sh event conf id | Idのイベント情報を参照する。 |
| mem\_adm.sh item conf id | Idのアイテムキャッシュ情報を参照する。 |
| mem\_adm.sh log conf id | Idのキャッシュログを吐き出す。 |

　サンプルデータは、「mem\_sample.sh」を参照。

### libmemcachedの取得

wget https://launchpad.net/libmemcached/1.0/1.0.18/+download/libmemcached-1.0.18.tar.gz

# tar xvf libmemcached-1.0.18.tar.gz

# cd libmemcached-1.0.18

# ./configure

# make

#make install

注意：memcapable.ccにバグ(if(hostname)->if(!hostname))

### memcachedの取得

「apt-get　install　memcached」

による。

　ただし、/etc/memcached.confで「#-l　127.0.0.1」とする。

　起動停止は、「service　memcached　｛start|stop｝」

# 試験

## 環境

試験スクリプト(test1.sh)は以下である。

#!/bin/sh

if [ $# -lt 2 ]

then

echo "USAGE:test1.sh service memcached"

exit 1

fi

service=$1

ID=0

conf=$HOME/\_$service.conf

client=/pss/libmemcached-1.0.18/clients

memcached=$2

local=127.0.0.1

while read id host bin root

do

if [ $id = $ID ]; then break; fi;

done < $conf

if [ $id = $ID ]

then

echo "=== memcapable ==="

ssh -t $host "cd $client;time ./memcapable -h $memcached"

ssh -t $host "cd $client;time ./memcapable -h $local"

echo "=== memcp ==="

ssh -t $host "cd $client;time ./memcp --servers=$memcached memcp.cc"

ssh -t $host "cd $client;time ./memcp --servers=$local memcp.cc"

echo "=== memcat ==="

ssh -t $host \

"cd $client;time ./memcat --servers=$memcached memcp.cc --file=z"

ssh -t $host \

"cd $client;time ./memcat --servers=$local memcp.cc --file=z"

echo "=== ascii->binary ==="

ssh -t $host "bash<<-\_EOF\_

cd $client

./memcp --servers=$local memcp.cc

./memcat --servers=$local memcp.cc --file=z --binary

diff z memcp.cc

\_EOF\_"

echo "=== binary->ascii ==="

ssh -t $host "bash<<-\_EOF\_

cd $client

./memcp --servers=$local memcat.cc --binary

./memcat --servers=$local memcat.cc --file=z

diff z memcat.cc

\_EOF\_"

while read id host bin root

do

if [ $id = "0" ]

then

host0=$host

bin0=$bin

root0=$root

fi

if [ $id = "1" ]

then

host1=$host

bin1=$bin

root1=$root

fi

done < $conf

echo "### delegation check ###"

ssh -t $host0 "cd $client;time ./memcp --servers=$memcached memcp.cc"

ssh -t $host0 "cd $bin0;./PaxosMemcacheAdm 0 item"

ssh -t $host1 "cd $client;time ./memcp --servers=$memcached memcp.cc"

ssh -t $host0 "cd $bin0;./PaxosMemcacheAdm 0 item"

fi

exit 0

## 性能

|  |  |  |  |
| --- | --- | --- | --- |
| PaxosMemcache | Memcapable | Memcp | Memcat |
| 無（直接） | real 0m6.359s  user 0m0.000s  sys 0m0.027s | real 0m14.902s  user 0m0.000s  sys 0m0.022s | real 0m0.014s  user 0m0.004s  sys 0m0.008s |
| 有（ｸﾗｲｱﾝﾄｷｬｯｼｭ） | real 0m14.902s  user 0m0.000s  sys 0m0.022s | real 0m0.014s  user 0m0.004s  sys 0m0.008s | real 0m0.011s  user 0m0.000s  sys 0m0.012s |

＊Memcp、memcatの対象ファイルは10kB程度のサイズである。

＊MemcatのPaxosMemcacheでは直前のコマンドで既にクライアントキャッシュ化されているので、直接memcachedにアクセスするよりも3ms程度速い。この差は、ネットワークのオーバーヘッドに相当すると考える。

## Memcached試験

　Memcachedパッケージのｔ配下のperl試験についてコメントを記述する。

### Cas.t

　エラー時の仕様書に記載していない応答がある。

### flush-all.t

　「flush\_all　exptime」の一定時間後のパージについて未実装だったので、Timerを導入し実装する。

1. デレゲーション管理サーバーが高可用性（障害なし）でなければ、同期制御ができない。 [↑](#footnote-ref-1)