# Kumofs memo – Kumo Fast Storage rev.948

FURUHASHI Sadayuki

# Contents

# Chapter 1

# 概要

kumofs は key-value 型のデータを保存する分散ストレージ。key=>value を保存する **set**、key を取得する **get**、key を削除する **delete** の 3 つの操作をサポートする。データを保持する **Server**、Server 群を管理する **Manager**、アプリケーションからのリクエストを Server に中継する **Gateway** の 3 種類のノードでシステムを構成する。データは複数の Server に分散して保存されるため、Server を追加するほど性能が向上する。データは 3 台の Server にコピーされて保存される。2 台までなら Server がダウンしても動作し続ける。Server 群は Manager によって死活監視されている。Server がダウンしたら、その Server は直ちにシステムから切り離される。ただし 1 台か 2 台の Manager が起動していないと Server の切り離しが行われないので、Manager が 1 台も起動していない状態で Server がダウンするとシステムが停止してしまう。Server を追加したり切り離したりした後、その状態をシステムに反映するには、レプリケーションされたデータの再配置を行う必要がある。これは自動では行われず、**kumoctl コマンド**を使って手動で行う。

## 1.1 Consistent Hashing

Consistent Hashing を使ってデータを分散して保存する*1。Server がダウンしたときは、その Server の仮想ノードに **fault フラグ**がセットされる。set/get/delete は fault フラグがセットされた Server をスキップして行われる。つまり、通常動作時はレプリケーションは 3 つ作成されるが、1 台が fault 状態ならコピーは 2 つ、2 台が fault 状態ならコピーは 1 つしか作成されない key が存在することになる。fault 状態の Server が 3 台以上になると、get/set/delete が失敗し続ける key が存在することになる。Server がダウンしても fault フラグがセットされるだけで、レプリケーションの再配置は行われない。fault フラグがセットされた Server が存在する状態で、kumoctl コマンドを使って **detach** コマンドを Manager に送信すると、fault フラグがセットされた Server がハッシュ空間から取り除かれる。同時にレプリケーションの再配置が行われ、すべての key に対してレプリケーションが 3 つ作成されるようにデータがコピーされる。Server が追加されてもすぐにはハッシュ空間には追加されず、レプリケーションの再配置は行われない。新たな Server が起動している状態で、kumoctl コマンドを使って **attach** コマンドを Manager に送信すると、新しい Server がハッシュ空間に追加される。同時にレプリケーションの再配置が行われ、すべての key に対してレプリケーションが 3 つだけ存在するようにデータが移動される。

# TODO: auto-replace

## 1.2 set/get/delete の保証範囲

### 1.2.1 set(key, value)

key=>value を保存する。保存できれば成功を返す。保存できなければエラーを返す。既に key が保存されていたとき、set が成功した場合は key の値は確実に上書きされている。set が失敗したとき、key の値は不定になっている。これは失敗したときにロールバックを行わないため。ロールバックを一貫性を損なうことなく行うための高級なアルゴリズムは実装されていない/使うと性能が低下してしまう。Server はレプリケーション先

の 2 台〜0 台のすべての Server にデータが受信されたことを確認してから Gateway にレスポンスを返す。どれか 1 台でもコピー処理が失敗したらエラーを返す。つまり、アプリケーションに成功が返されたときは fault 状態でないすべての Server にレプリケーションがコピーされており、それ以降に古いデータが読み出されることはない。ただしディスクに書き込まれているとは限らない。

### 1.2.2 get(key)

key を set するリクエストが成功していた場合は、その key に対応する value を返す。set が失敗していた場合は、null か、set に失敗した value が返る。それ以外であれば null を返す。key を set するリクエストが成功して value が保存されていたとしても、レプリケーションされたすべての Server の負荷が非常に高いために応答できない場合は、get がタイムアウトする可能性がある。key が保存されていなかった場合はエラーにならないが、タイムアウトした場合はエラーになる。

### 1.2.3 delete(key)

key を削除する。再配置処理を行っている間に delete を行うと、削除されないことがある。また同じ key に対して delete と set をほぼ同時に行うと、削除されないことがある。これはレプリケーションを行う Server 同士のやりとりが、Gateway が送出した delete リクエストと入れ違う可能性があるため。Server の引数を調整することで、delete が一貫性を保たない確率を減らすことができる。

## 1.3 動作環境と制限

### 1.3.1 サーバーの時刻設定

Manager と Server を動作させるホストの時刻設定は、TIME_ERROR_MARGIN 秒（コンパイル時に決定。デフォルトでは 5 秒）以上ずれていると正常に動作しない。また UTC と localtime はどちらかに揃える必要がある。

# TODO

*1[1]ハッシュ関数は SHA-1 で、下位 64 ビットのみ使う。仮想ノードは 128 台

[1] footnote4-1-anchor.tex

# Chapter 2

# インストールと実行

## 2.1 依存関係

### 2.1.1 動作環境

- linux >= 2.6.18
- glibc >= XXX

### 2.1.2 コンパイル時に必要なもの

- g++ >= 4.1
- ragel >= 6.3
- git >= XXX

### 2.1.3 コンパイル時と実行時に必要なもの

- ruby >= 1.8
- rubygems
- libcrypto(openssl)
- zlib >= XXX
- Tokyo Cabinet >= 1.4.9

## 2.2 コンパイル

まず最新の MessagePack をインストールする。

```
$ git clone git://git.sourceforge.jp/gitroot/msgpack/msgpack.git
$ cd msgpack
$ ./bootstrap && ./configure && make
$ sudo make install
$ cd ruby
$ ./gengem
$ sudo gem install gem/pkg/msgpack-*.gem
```

次に kumofs をインストールする。

```
$ ./configure && make && make install
```

以下の 4 つのコマンドがインストールされる：

**kumo-manager** Manager ノード。Server ノードの管理をする。

**kumo-server** Server ノード。実際にデータを保存する。

**kumo-gateway** Gateway ノード。memcached プロトコルのサーバーで、アプリケーションからの要求を Server ノードに中継する。

**kumoctl** Manager ノードを制御するための管理コマンド

**kumolog** バイナリフォーマットのログをテキストフォーマットに変換する

**kumomergedb** コールドバックアップファイルをマージする

### 2.2.1 configure フラグ

–**with-msgpack=DIR** MessagePack がインストールされているディレクトリを指定する

–**with-tokyocabinet=DIR** Tokyo Cabinet がインストールされているディレクトリを指定する

–**enable-trace** 画面を埋め尽くすほど冗長なデバッグ用のメッセージを出力するようにする

–**with-tcmalloc**[**=DIR**] tcmalloc とリンクする

## 2.3 実行例

### 2.3.1 Manager 2 台, Server 4 台を使った冗長構成

**s1**〜**s4** の 4 台でクラスタを構成し、**c1** と **c2** で動作するアプリケーションから利用する例。s1〜s4 で Server を起動し、**s1** と **s2** では同時に Manager も起動する。**c1** と **c2** では Gateway を起動する。

```
[s1]$ kumo-manager -v -l s1 -p s2    # Manager 同士は互いに指定する
[s2]$ kumo-manager -v -l s2 -p s1    # Manager 同士は互いに指定する
[s1]$ kumo-server  -v -m s1 -p s2 -l s1 -s database.tch    # -m と-p で Manager を指定する
[s2]$ kumo-server  -v -m s1 -p s2 -l s2 -s database.tch    # -l は常に自ホストのアドレス
[s3]$ kumo-server  -v -m s1 -p s2 -l s3 -s database.tch    # -s はデータベース名
[s4]$ kumo-server  -v -m s1 -p s2 -l s4 -s database.tch    # -v は冗長なメッセージを出力
[c1]$ kumo-gateway -v -m s1 -p s2 -t 11211    # 11211/tcp で memcached テキストプロトコル
[c2]$ kumo-gateway -v -m s1 -p s2 -t 11211    # を待ち受ける
```

### 2.3.2 localhost でクラスタを構成する

localhost で Manager ノード 1 台、Server 2 台を使ってクラスタを構成する例。

```
[localhost]$ kumo-manager -v -l localhost   # Manager を 1 台で運用するときは-p を省略
                          # kumo-server はポートを変えて起動する
[localhost]$ kumo-server  -v -m localhost -l localhost:19801 -L 19901 -s database1.tch
[localhost]$ kumo-server  -v -m localhost -l localhost:19802 -L 19902 -s database2.tch
[localhost]$ kumo-gateway -v -m localhost -t 11211
```

## 2.4 主な引数

## 2.5 共通

**-o <path.log>** ログを標準出力ではなく指定されたファイルに出力する

**-g <path.mpac>** バイナリログを指定されたファイルに出力する

**-d <path.pid>** デーモンになる。指定されたファイルに pid を書き出す

**-v** WARN よりレベルの低いメッセージを出力する

**-Ci <sec>** タイマークロックの間隔を秒で指定する。小数を指定できる

**-Ys <sec>** connect(2) のタイムアウト時間を秒で指定する。小数を指定できる

**-Yn <num>** connect(2) のリトライ回数を指定する

**-TR <num>** 送信用スレッドの数を指定する

**-TW <num>** 受信用スレッドの数を指定する

### 2.5.1 kumo-manager

**-l <address>** 待ち受けるアドレス。**他のノードから見て**接続できるホスト名とポート番号を指定する

**-p <address>** もし存在するなら、もう一台の kumo-manager のホスト名とポート番号を指定する

**-c <port>** kumoctl からのコマンドを受け付けるポート番号を指定する

–**auto-replace** Server が追加・切断されたときに、マニュアル操作を待たずにレプリケーションの再配置を自動的に行うようにする。実行中でも kumoctl コマンドを使って変更できる

### 2.5.2 kumo-server

**-l <address>** 待ち受けるアドレス。**他のノードから見て**接続できるホスト名とポート番号を指定する

**-L <port>** kumo-server が待ち受けるもう一つのポートのポート番号を指定する

**-m <address>** kumo-manager のホスト名とポート番号を指定する

**-p <address>** もし存在するなら、もう一台の kumo-manager のホスト名とポート番号を指定する

**-s <path.tch>** データを保存するデータベースファイルのパスを指定する

**-f <dir>** レプリケーションの再配置に使う一時ファイルを保存するディレクトリを指定する。データベースファイルのサイズに応じて十分な空き容量が必要

**-gS <seconds>** delete したエントリのクロックを保持しておくメモリ使用量の上限を KB 単位で指定する

**-gN <seconds>** delete したエントリのクロックを保持しておく最小時間を指定する。メモリ使用量が上限に達していると、最大時間に満たなくても最小時間を過ぎていれば削除される。

**-gX <seconds>** delete したエントリのクロックを保持しておく最大時間を指定する

### 2.5.3 kumo-gateway

**-m <address>** kumo-manager のホスト名とポート番号を指定する

**-p <address>** もし存在するなら、もう一台の kumo-manager のホスト名とポート番号を指定する

**-t <port>** memcached テキストプロトコルを待ち受けるポート番号を指定する

**-G <number>** get の最大リトライ回数を指定する

**-S <number>** set の最大リトライ回数を指定する

**-D <number>** delete の最大リトライ回数を指定する

**-As** set 操作でレプリケーションするとき、レプリケーション完了の応答を待たずに成功を返すようにする

**-Ad** delete 操作でレプリケーションするとき、レプリケーション完了の応答を待たずに成功を返すようにする

*1[1]ハッシュ関数は SHA-1 で、下位 64 ビットのみ使う。仮想ノードは 128 台

[1] footnote4-1-anchor.tex

# Chapter 3

# kumoctl

kumoctl コマンドを使うと Manager の状態を取得したり、コマンドを送ったりできる。Ruby で書かれたスクリプト。実行するには gem で msgpack パッケージをインストールする。第 1 引数に Manager のホスト名とポート番号を指定し、第 2 引数にコマンドを指定する。

```
$ kumoctl --help
Usage: kumoctl address[:port=19799] command [options]
command:
   status                     get status
   attach                     attach all new servers and start replace
   attach-noreplace           attach all new servers
   detach                     detach all fault servers and start replace
   detach-noreplace           detach all fault servers
   replace                    start replace without attach/detach
   backup  [suffix=????????]  create backup with specified suffix
   enable-auto-replace        enable auto replace
   disable-auto-replace       disable auto replace
```

## 3.1 status

Manager が持っているハッシュ空間を取得して表示する。

```
$ kumoctl localhost status
hash space timestamp:
  Wed Dec 03 22:15:45 +0900 2008 clock 58
attached node:
  127.0.0.1:8000  (active)
  127.0.0.1:8001  (fault)
not attached node:
  127.0.0.1:8002
```

**attached node** はハッシュ空間に入っている Server の一覧を示している。**(active)** は正常動作中の Server で、**(fault)** は fault フラグが立っている Server を示している。**not attached node** はハッシュ空間に入っていないか、入っているが (fault) 状態でまだ再 attach されていない Server の一覧を示している。

レプリケーションの再配置を行ったとき、Manager が 2 台起動していれば 2 つの Manager 間で新しいハッシュ空間が同期される。ただし新しいハッシュ空間が空の時は同期されない。この理由は、障害が発生していた Manager を復旧したときに空のハッシュ空間が同期されてしまう可能性があるため。起動した直後はクロック（後述）が調整されていないために、ハッシュ空間の新旧の比較が正常に機能しない。このため空のハッシュ空間を受け取ったときは無視するようになっている。 # FIXME この動作は正しい？もっと良い回避方法は無いか？

## 3.2 attach

status で **not attached node** に表示されている Server をハッシュ空間に組み入れ、レプリケーションの再配置を開始する。

## 3.3 attach-noreplace

attach と同じだがレプリケーションの再配置を開始しない。ただし再配置をしないまま長い間放置してはいけない。再配置を行わないと、エラーが積もって Gateway から最新のハッシュ空間を要求されたとき（後述）、Gateway が持っているハッシュ空間と Server が持っているハッシュ空間が食い違ってしまう。食い違うと set や delete がいつまで経っても成功しなくなってしまう。

## 3.4 detach

status で **attached node** に表示されていて (fault) 状態の Server をハッシュ空間から取り除き、レプリケーションの再配置を開始する。

## 3.5 detach-noreplace

detach と同じだがレプリケーションの再配置を開始しない。再配置をしないまま長い間放置してはいけない。

## 3.6 replace

レプリケーションの再配置を開始する。

## 3.7 backup

コールドバックアップを作成する。バックアップは Server で作成され、元のデータベース名に suffix を付けた名前のファイルにデータベースがコピーされる。手元にバックアップを持ってくるには、rsync や scp などを使って Server からダウンロードする。suffix は省略するとその日の日付（YYMMDD）が使われる。作成されたバックアップファイルは、kumomergedb コマンドを使って 1 つのファイルに結合することができる。

```
$ kumomergedb backup.tch-20090101 \
              server1.tch-20090101 server2.tch-20090101 server3.tch-20090101
```

# Chapter 4

# kumostat

kumostat コマンドを使うと Server の状態を取得することができる。Ruby で書かれたスクリプト。実行するには gem で msgpack パッケージをインストールする。第 1 引数に Server のホスト名とポート番号を指定し、第 2 引数にコマンドを指定する。

```
Usage: kumostat address[:port=19800] command [options]
command:
   pid                         get pid of server process
   uptime                      get uptime
   time                        get UNIX time
   version                     get version
   cmd_get                     get number of get requests
   cmd_set                     get number of set requests
   cmd_delete                  get number of delete requests
   items                       get number of stored items
```

## 4.1 pid

kumo-server プロセスの pid を取得する

## 4.2 uptime

kumo-server プロセスの起動時間を取得する。単位は秒。

## 4.3 time

kumo-server プロセスが走っているホストの UNIX タイムを取得する。

## 4.4 version

バージョンを取得する。

## 4.5 cmd_get / cmd_set / cmd_delete

それぞれ Gateway からの Get リクエスト、Set リクエスト、Delete リクエストを処理した回数を取得する。

## 4.6 items

データベースに入っているエントリの数を取得する。

# Chapter 5

# ログ

kumo-manager, kumo-server, kumo-gateway は、それぞれ 2 種類のログを出力する:

**テキストログ** 行区切りのテキストフォーマットのログ。通常標準出力に出力される

**バイナリログ** MessagePack でシリアライズされたログ

テキストログは常に出力される。**-v** オプションを付けると冗長なログも出力されるようになる。テキストログはファイルに書き出すこともできるが、ログローテーションはサポートしていない。**-d** <**path.pid**> オプションを指定してデーモンとして起動するか、**-o -**オプションを指定すると、ログに色が付かなくなる。

バイナリログは**-g** <**path.mpac**> オプションを付けたときだけ出力される。**-v** オプションは影響しない。バイナリログは SIGHUP シグナルを受け取るとログファイルを開き直すため、logrotate などを使ってログローテーションができる。

バイナリログは **kumolog** コマンドを使ってテキストに変換して読むことができる。

```
$ kumolog manager.mpac
```

# Chapter 6

# チューニング

## 6.1 データベースのチューニング

Tokyo Cabinet のチューニングによって性能が大きく変わる。kumo-server を起動する前にあらかじめ **tchmgr** コマンドでデータベースファイルを作成しておく。チューニングのパラメータは Tokyo Cabinet のドキュメント参照。http://tokyocabinet.sourceforge.net/spex-ja.html[1]

```
$ tchmgr create /path/to/database.tch 1048568  # バケット数を 2097136 個にして作成
$ kumo-server -m localhost -s /path/to/database.tch
```

## 6.2 タイムアウト時間と keepalive 間隔の調整

# TODO

[1] http://tokyocabinet.sourceforge.net/spex-ja.html

# Chapter 7

# 死活監視と再配置

## 7.1 障害の検出

Manager と Server の接続では、あるノードにリクエストまたはレスポンスを送信しようとしたときに、そのノードとのコネクションが一本も存在せず、さらに connect(2) が 4 回*2 連続して失敗したら、そのノードはダウンしたと見なす。Manager と Server は 2 秒間隔*3 で keepalive メッセージをやりとりしているので、いつも何らかのリクエストかレスポンスを送ろうとしている状態になっている。connect(2) は次の条件で失敗する：

- 接続相手から明示的に接続を拒否された（Connection Refused）
- 接続相手からの応答がない時間が 3 ステップ*4 続いた。1 ステップは 0.5 秒*5

## 7.2 接続の検出

Manager と Server の接続では、あるノードから接続を受け付けた後、そのノードから初期ネゴシエーションメッセージを受け取り、かつそのメッセージのフォーマットが正しければ、そのノードが新たに起動したと見なす。

## 7.3 ハッシュ空間の更新

Consistent Hashing のハッシュ空間を更新できるのは Manager だけで、最新のハッシュ空間は常に Manager が持っている。通常動作時には 1 種類のハッシュ空間しか存在しないが、レプリケーションの再配置を行っている間は 2 種類のバージョンが存在する。最新のもの（Server の追加/切り離しの更新が反映されている）は **whs**、1 つ前のバージョン（Server の追加/切り離しの更新が反映されていない）は **rhs** という名前が付いている。

Manager は kumoctl コマンドでレプリケーションの再配置を行うように指令されると、まず Server の追加/切り離しを whs に反映する。もう 1 台の Manager が存在すればその Manager に更新した whs を送信する。次に認識しているすべての Server に whs を送信し、レプリケーションのコピーを行うようにコマンドを送る。Server は自分が持っている whs と Manager から送られてきた whs を比較し、必要なら他の Server にデータのコピーを行う（このときデータベースを上から下まで読み込む）。Server はコピーが終わったら whs を rhs にコピーする。Server はすべてのデータを確認し終えたら、Manager にコピーが終了した旨を通知する。Manager はすべての Server でコピーが終了した通知を受け取ったら、whs を rhs にコピーする。また、認識しているすべてのサーバーにレプリケーションの削除を行うようにコマンドを送る。Server は whs を参照して、自分が持っている必要がないデータがデータベースの中に入っていたら、それを削除する（このときもデータベースを上から下まで読み込む）。

Manager はレプリケーションのコピーを行っている最中に Server がダウンしたことを検知したら、すべての Server からレプリケーションのコピーが終了した通知を受け取っても、レプリケーションの削除を行わない。

Server は Gateway から get/set/delete リクエストを受け取ったとき、その key に対する割り当てノードが本当に自分であるか確認するために、get の場合は rhs を、set/delete の場合は whs を参照する。

## 7.4 レプリケーションの再配置アルゴリズム

# TODO レプリケーションの再配置アルゴリズム logic/srv_replace.cc:Server::replace_copy()

# Chapter 8

# レプリケーション

## 8.1 set/delete の伝播

Gateway に set リクエストを送信すると、key にハッシュ関数を適用してハッシュ空間から検索し、一番最初にヒットした Server に対して set リクエストが送信される。set リクエストを受け取った Server は、key のハッシュをハッシュ空間から検索し、自分が確かに最初にヒットする Server かどうか確かめる。そうでなければ Gateway に「ハッシュ空間が古いぞ」とエラーを返す。次に Server は、自分の次の Server と次の次の Server にデータをコピーする。このときコピー先の Server に fault フラグが立っていたら、その Server にはコピーしない。

Gateway は set/delete が何回失敗しても、次の Server にフォールバックすることはない。set 先の Server が別の Server に切り替わるのは、Manager から新しいハッシュ空間を届いたときのみ。

以上の仕組みから、ある key を set/delete するときは必ず単一の Server を経由することになる。このためほぼ同時に set/delete されても必ず順序が付けられ、常に最新の結果がだけが残る。

## 8.2 get のフォールバック

Gateway は get リクエストがタイムアウトしたり失敗したりすると、ハッシュ空間上の次の Server にリクエストする。それでもタイムアウトしたときは次の次の Server にリクエストする。リトライ回数の上限に達するまで、最初の Server →次の Server →次の次の Server →最初の Server →…とリトライが繰り返される。

get は Manager から新しいハッシュ空間が届くのを待つことなくフォールバックする。

## 8.3 タイムアウト

Gateway でも Server でも Manager でも、リクエストを送ってから 10 ステップ（1 ステップは 0.5 秒*6）の間にレスポンスが返ってこないと、そのリクエストはタイムアウトしてエラーになる。プログラムから見て TCP コネクションが確立しているか否かはタイムアウトには関係しない。コネクションが確立していなくても時間以内に再接続してレスポンスが返れば正常通り処理が続行され、コネクションが確立していても時間以内にレスポンスが返ってこなければタイムアウトする。

Gateway は Server に送ったリクエストがエラーになった回数が 5 回*7 以上失敗すると、Manager から最新のハッシュ空間を取得する。

## 8.4 リトライ

Gateway は set は最大 20 回*8 まで、delete は最大 20 回*9 まで、get は最大 5×(レプリケーション数==3 + 1) 回*10 までリトライする。制限回数までリトライしても失敗したらアプリケーションにエラーが返される。

*1[1]ハッシュ関数は SHA-1 で、下位 64 ビットのみ使う。仮想ノードは 128 台

*2[2]–connect-retry-limit で指定

*3[3]–keep-alive-interval 引数で指定

*4[4]–connect-timeout-steps 引数で指定

*5[5]–clock-interval 引数で指定

*6[6]–clock-interval 引数で指定

*7[7]–renew-threashold 引数で指定

*8[8]–set-retry 引数で指定

*9[9]–delete-retry 引数で指定

*10[10]係数は–get-retry 引数で指定

[1] footnote4-1-anchor.tex
[2] footnote4-2-anchor.tex
[3] footnote4-3-anchor.tex
[4] footnote4-4-anchor.tex
[5] footnote4-5-anchor.tex
[6] footnote4-6-anchor.tex
[7] footnote4-7-anchor.tex
[8] footnote4-8-anchor.tex
[9] footnote4-9-anchor.tex
[10] footnote4-10-anchor.tex

# Chapter 9

# クロック

データベースに保存されているすべての value や、ハッシュ空間には、クロック（=タイムスタンプ）が付与されている。value 同士やハッシュ空間同士でどちらが新しいかを比べるために利用している。ref:Lamport Clock の解説[1]

## 9.1 クロックのフォーマット

クロックは 64 ビットの整数で、上位 32 ビットには UNIX タイム（精度は秒）、下位 32 ビットには Lamport Clock が入っている。UNIX タイムが上位に入っているので、Server/Manager 同士の時刻が 1 秒以上ずれていると、Lamport Clock に関係なく間違った比較が行われてしまう。

## 9.2 データベースのフォーマット

データベースに key を保存するとき、先頭の 64 ビットに key のハッシュを負荷して保存する。データベースに value を保存するとき、先頭の 64 ビットにクロックを付加して保存する。またその次の 64 ビットも予約してあるが、使っていない。

```
Database entry format
Big endian

key:
+--------+-----------------+
|   64   |       ...       |
+--------+-----------------+
hash
         key

value:
+--------+--------+-----------------+
|   64   |   64   |       ...       |
+--------+--------+-----------------+
clocktime
         meta
                  data
```

[1] http://funini.com/kei/logos/clock.shtml

## 9.3 レプリケーションでの利用

Server から別の Server にデータをコピーするとき、後から来た set リクエストのレプリケーションが、先に来た set リクエストのレプリケーションを追い抜いて先行してしまうことが発生し得る。Server はレプリケーションを受け取ったとき、既に保存されている value のクロックと新たに届いた value のクロックを比べ、新たに届いた方が新しかった場合のみデータベースを更新する。レプリケーションの再配置を行うとき、ほとんどの場合はレプリケーションされたどの Server も同じデータを持っているが、set が失敗していた場合は異なるデータを持っている可能性がある。このときどの Server が持っているデータが最新なのか比べる必要があり、クロックを利用して比較する。

## 9.4 Manager 間の協調動作での利用

Manager が 2 台動作しているとき、どちらが持っているハッシュ空間が最新なのかを比べる必要がある。ハッシュ空間を更新するときに更新した時のクロックを付与しておき、比較するときにこのクロックを利用する。

*1[2]ハッシュ関数は SHA-1 で、下位 64 ビットのみ使う。仮想ノードは 128 台

*2[3]–connect-retry-limit で指定

*3[4]–keep-alive-interval 引数で指定

*4[5]–connect-timeout-steps 引数で指定

*5[6]–clock-interval 引数で指定

*6[7]–clock-interval 引数で指定

*7[8]–renew-threashold 引数で指定

*8[9]–set-retry 引数で指定

*9[10]–delete-retry 引数で指定

*10[11]係数は–get-retry 引数で指定

[2]footnote4-1-anchor.tex
[3]footnote4-2-anchor.tex
[4]footnote4-3-anchor.tex
[5]footnote4-4-anchor.tex
[6]footnote4-5-anchor.tex
[7]footnote4-6-anchor.tex
[8]footnote4-7-anchor.tex
[9]footnote4-8-anchor.tex
[10]footnote4-9-anchor.tex
[11]footnote4-10-anchor.tex