kumofs

FURUHASHI Sadayuki

Contents

Chapter 1

kumofsとは？

kumofs は key-value 型のデータを保存する非常に高速な分散ストレージシステムです。

少ない台数のサーバーでも運用でき、その後負荷に応じてサーバーを追加することで柔軟に性能を向上させていくことができます。また 1 台や 2 台のサーバーに障害が発生しても耐障害性を持ち、単一点障害が一切ありません。1 台のサーバーでも毎秒 10 万回以上の問い合わせに応答できる性能を持ち、低コストで極めて高速なストレージシステムを構築・運用できます。

Chapter 2

特徴とデータモデル

kumofs は以下のような特徴があります：

- 一部のサーバーがダウンしても正常に動き続けます
- サーバーを追加すると読み・書き両方の性能が向上します
- システムを止めずにサーバーを追加できます
- 小さなデータを大量に保存するのに適しています
- 2 台から 60 台程度までスケールします（60 台以上はまだ検証されていません）
- memcached プロトコルを使ってアクセスできます

kumofs に保存できるデータは、**key** と **value** だけで表されるシンプルなデータです。以下の 3 つの操作をサポートしています：

Set(key, value) key と value のペアを保存します。1 つの key-value ペアは合計 3 台のサーバーにレプリケーションされます。Set 操作が成功すると、レプリケーション先のすべてのサーバーに同じ key-value ペアが保存され、その直後からどのクライアントからでも同じ value を Get できます。Set 操作が失敗すると、その key に対応する value は不定になります。その key は再度 Set するか、Delete するか、Get しないようにしてください。

value = Get(key) key に対応する value を取得します。Set 中に Get した場合に古い value が取得されるか新しい value が取得されるかは不定です。ただし新旧が混ざった value にはなることはありません。

Delete(key, value) key に対応する value を削除します。Delete 操作は実際には「削除済みフラグを Set」する操作なので、Set と同じ挙動になります。削除済みフラグは一定の時間が経過すると本当に削除されます。

Chapter 3

インストール

kumofs をコンパイルして実行するには以下の環境が必要です：

- linux >= 2.6.18
- Tokyo Cabinet[1] >= 1.4.10
- MessagePack for Ruby[2] >= 0.3.1
- MessagePack for C++[3] >= 0.3.1（コンパイル時のみ）
- Ragel[4] >= 6.3（コンパイル時のみ）
- g++ >= 4.1（コンパイル時のみ）
- ruby >= 1.8.7
- libcrypto (openssl)
- zlib

./configure && make && make install でインストールできます。

```
$ ./configure
$ make
$ sudo make install
```

[1] http://tokyocabinet.sourceforge.net/
[2] http://msgpack.sourceforge.jp/ruby:install.ja
[3] http://msgpack.sourceforge.jp/cpp:install.ja
[4] http://www.complang.org/ragel/

Chapter 4

チュートリアル

kumofs は以下の 3 つのプログラムで構成されています：

kumo-server 実際にデータを保存する。少なくとも 1 台必要で、後から追加できる。

kumo-manager kumo-server 群を管理する。1 台または 2 台。

kumo-gateway アプリケーションからのリクエストを kumo-server に中継する。アプリケーションを動かすホスト上で起動しておく。

ここでは 3 台のサーバー **svr[123]** を使って分散ストレージを構築する方法を紹介します。**svr1** と **svr2** で kumo-manager を起動し、**svr1,svr2,svr3** で kumo-server を起動します。それから別のクライアント **cli1** からデータを保存取得してみます。

4.1 kumo-manager を起動

まず最初に **svr1** と **svr2** の 2 台のサーバーで kumo-manager を起動します。kumo-manager には少なくとも自分のホスト名か IP アドレスと、もう 1 台の kumo-manager のホスト名か IP アドレスを指定する必要があります。

```
[svr1]$ kumo-manager -v -l svr1 -p svr2
[svr2]$ kumo-manager -v -l svr2 -p svr1
```

kumo-manager は 19700/tcp を listen します（デフォルト値）。

4.2 kumo-server を起動

次に **svr1,svr2,svr3** の 3 台のサーバーで kumo-server を起動します。kumo-server には少なくとも自分のホスト名か IP アドレス、kumo-manager のホスト名か IP アドレス、データベースのパスを指定する必要があります。

```
[svr1]$ kumo-server -v -l svr1 -m svr1 -p svr2 -s /var/kumodb.tch
[svr2]$ kumo-server -v -l svr2 -m svr1 -p svr2 -s /var/kumodb.tch
[svr3]$ kumo-server -v -l svr3 -m svr1 -p svr2 -s /var/kumodb.tch
```

kumo-server は 19800/tcp と 19900/tcp を listen します（デフォルト値）。

4.3 kumo-server を登録

kumo-server は起動しただけでは追加されません。**kumoctl** コマンドを使って登録します。まずは kumo-manager から kumo-server が認識されているかどうかを確認してみます：

```
$ kumoctl svr1 status
hash space timestamp:
  Wed Dec 03 22:15:55 +0900 2008 clock 62
attached node:
not attached node:
  192.168.0.101:19800
  192.168.0.102:19800
  192.168.0.103:19800
```

not attached node のところに 3 台のサーバーが認識されていることを確認したら、実際に登録します：

```
$ kumoctl svr1 attach
```

最後に本当に登録されたか確認します：

```
$ kumoctl svr1 status
hash space timestamp:
  Wed Dec 03 22:16:00 +0900 2008 clock 72
attached node:
  192.168.0.101:19800  (active)
  192.168.0.102:19800  (active)
  192.168.0.103:19800  (active)
not attached node:
```

無事 **attached node** のところに登録されました。

4.4 kumo-gateway の起動

実際にアプリケーションから kumofs を利用するには、そのホストで kumo-gateway を起動しておきます。kumo-gateway には memcached プロトコルが実装されているので、アプリケーションからは **localhost で動作している memcached** に見えます。kumo-gateway には少なくとも kumo-manager のホスト名か IP アドレスと、memcached を listen するポート番号を指定する必要があります。

```
[cli1]$ kumo-gateway -v -m svr1 -p svr2 -t 11211
```

これで kumofs を利用する準備が整いました。cli1 で localhost の 11211 番に memcached クライアントを使って接続してみてください。

4.5 kumo-server を追加する

別のサーバー **svr4** を追加してみましょう。まず kumo-server を起動し、次に kumoctl コマンドを使って登録します。

```
[svr4]$ kumo-server -v -l svr4 -m svr1 -p svr2 -s /var/kumodb.tch
$ kumoctl svr1 attach
```

これで新しいサーバーが追加されました。

4.6 すべてのプロセスを localhost で動かす

すべてのプロセスを 1 台のホストで動かして試してみるには以下のようにします：

```
[localhost]$ kumo-manager -v -l localhost
[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-server  -v -m localhost -l localhost:19803 -L 19902 -s ./database3.tch
[localhost]$ kumo-gateway -v -m localhost -t 11211
```

Chapter 5

HowTo

5.1 kumo-manager の状態を取得する

kumo-manager はクラスタ全体を管理しているノードです。kumo-manager の状態を取得することでクラスタ全体の状態を知ることができます。kumo-manager の状態を取得するには、**kumoctl** コマンドを使います。第一引数には kumo-manager（複数いる場合はどれか 1 台）のアドレスを指定し、第二引数には **status** と指定します。

```
$ kumoctl svr1 status
hash space timestamp:
  Wed Dec 03 22:15:45 +0900 2008 clock 58
attached node:
  192.168.0.101:19800  (active)
  192.168.0.102:19800  (active)
  192.168.0.103:19800  (active)
  192.168.0.104:19800  (fault)
not attached node:
  192.168.0.104:19800
```

表示の見方：

hash space timestamp この kumo-manager に登録されている kumo-server の一覧が最後に更新された時刻。サーバーを追加したときとサーバーがダウンしたときに更新されます。

attached node 登録されている kumo-server の一覧。**(active)** と表示されているノードは正常に動作しているノードで、**(fault)** と表示されているノードは障害が発生しているか障害が発生してからまだ再登録されていないノードです。

not attached node 認識されているがまだ登録されていない kumo-server の一覧。

kumoctl コマンドの詳しい使い方はリファレンスを参照してください。

5.2 kumo-server の状態を取得する

kumo-server は実際にデータを保存しているノードです。**kumostat** コマンドを使うと、その kumo-server に保存されているデータの件数や、今までに処理された Get 操作の総数などを取得できます。

```
$ kumostat svr1 items
```

第 1 引数には kumo-server のアドレスを指定します。第 2 引数には主に以下のサブコマンドを指定できます：

uptime kumo-server プロセスの起動時間（単位は秒）

version kumo-server のバージョン

cmd_get Get 操作を処理した回数

cmd_set Set 操作を処理した回数

cmd_delete Delete 操作を処理した回数

items データベースに保存されているデータの件数

kumostat コマンドの詳しい使い方はリファレンスを参照してください。

5.3 kumo-server の負荷をモニタする

kumotop コマンドを使うと、top コマンドのように kumo-server の負荷をモニタリングすることができます。引数には**-m** に続いて kumo-manager のアドレスを指定するか、モニタしたい kumo-server のアドレスを列挙します。

```
$ kumotop -m svr1
```

5.4 落ちた kumo-manager を復旧する

FIXME2 台で冗長構成を取っているときに片方の kumo-manager がダウンしまった場合は、まず kumo-manager を再起動してください。kumo-manager の IP アドレスは障害発生前と同じにしておく必要があります。次に残っていた方の kumo-manager に対して **kumoctl** コマンドの **replace** サブコマンドを発行して、2 台の kumo-manager の情報を同期してください。

また kumo-manager はすべてダウンしてしまってもシステムを止めることなく復旧できます。この場合は、まずダウンしている kumo-server がいるときは再起動してください。次に kumo-manager をすべて再起動させてください。最後に **kumoctl** コマンドの **attach** サブコマンドを使って kumo-server を登録します。

5.5 落ちた kumo-server を復旧する

FIXMEkumo-server が 1 台ダウンすると、一部の key-value ペアの複製が 1 つ減った状態のまま動作し続けます。2 台ダウンすると、1 つか 2 つ減った状態のままになります。この状態から複製の数を 3 つに戻すには、kumo-server を復旧させて kumoctl コマンドを使って再度登録するか、ダウンした kumo-server を完全に切り離します。

まず kumoctl コマンドを使ってどの kumo-server に障害が発生しているかを確認します：

```
[xx]$ kumoctl m1 status    # Manager のアドレスを指定して状態を取得
hash space timestamp:
  Wed Dec 03 22:15:35 +0900 2008 clock 50
attached node:
  192.168.0.101:19800  (active)
  192.168.0.102:19800  (active)
  192.168.0.103:19800  (active)
  192.168.0.104:19800  (fault)
not attached node:
```

(fault) と表示されている kumo-server に障害が発生しています。ここで kumo-server を再起動すると、以下のようになります：

```
[xx]$ kumoctl m1 status
hash space timestamp:
  Wed Dec 03 22:15:45 +0900 2008 clock 58
attached node:
  192.168.0.101:19800  (active)
  192.168.0.102:19800  (active)
  192.168.0.103:19800  (active)
  192.168.0.104:19800  (fault)
not attached node:
  192.168.0.104:19800
```

not attached node のところに表示されている kumo-server は、Manager ノードから認識されているが、まだ登録されていない kumo-server の一覧です。

ここで **attach** コマンドを発行すると、kumo-server が実際に登録され、データの複製が 3 つになるようにコピーされます：

```
[xx]$ kumoctl m1 attach    # attach
[xx]$ kumoctl m1 status
hash space timestamp:
  Wed Dec 03 22:15:55 +0900 2008 clock 62
attached node:
  192.168.0.101:19800  (active)
  192.168.0.102:19800  (active)
  192.168.0.103:19800  (active)
  192.168.0.104:19800  (active)
not attached node:
```

attach コマンドを発行して新たに kumo-server を登録するとき、データの複製をコピーするための比較的大規模なネットワークトラフィックが発生することに注意してください。

kumoctl コマンドで、attach コマンドの代わりに **detach** コマンドを発行すると、fault 状態の kumo-server が切り離されます。このときも複製の数が 3 つになるようにデータのコピーが行われます。

```
[xx]$ kumoctl m1 detach    # detach
[xx]$ kumoctl m1 status
hash space timestamp:
  Wed Dec 03 22:15:55 +0900 2008 clock 62
attached node:
  192.168.0.101:19800  (active)
  192.168.0.102:19800  (active)
  192.168.0.103:19800  (active)
not attached node:
  192.168.0.104:19800
```

FIXME kumo-server を復旧するときにのデータベースファイルの扱い delete したデータが復活する件

5.6 新しい kumo-server を追加する

TODO

5.7 あるデータがどの kumo-server に保存されているか調べる

kumofs はデータを複数の kumo-server に分散して保存します。ある key が実際にどの kumo-server に保存されているかを調べるには、**kumohash** コマンドを使います。

```
$ kumohash -m svr1 assign "the-key"
```

TODO

5.8 バックアップを作成する

TODO

5.9 バックアップから復旧する

サーバーのお引っ越しなどでバックアップを使ってクラスタを復旧したい場合は、次の手順で行います：

1. バックアップを作成する **kumoctl** コマンドの **backup** サブコマンドを使って、データベースファイルのバックアップを作成しておきます。

2. バックアップファイルを 1 つにまとめる データベースファイルのバックアップは、それぞれの kumo-server で作成されるので、複数のファイルに分割されています。これを **kumomergedb** コマンドを使って 1 つのデータベースファイルに結合します。

3. まとめたデータベースファイルを配る kumomergedb コマンドを使ってまとめたデータベースファイルを、kumo-server を起動するすべてのサーバーに配ります。

4.kumo-server を起動する kumo-server を起動します。このとき-s オプションには先ほど配ったデータベースファイルを指定します。

5.kumoctl で attach する **kumoctl** コマンドの **attach** サブコマンドを使って kumo-server を登録します。このときその kumo-server が持っている必要の無いデータがデータベースファイルから削除されます。

これで復旧は完了です。

Chapter 6

トラブルシューティング

6.1 Set が必ず失敗する

何度やっても Set 操作が必ず失敗してしまう場合は、以下の項目を調べてみてください：

kumo-server は登録されているか kumoctl MANAGER status を使って kumo-manager の状態を調べて、kumo-server が **attached node" として登録されており、**(active)"' と表示されていることを確認してください。

サーバーによってルーティング表が食い違っていないか kumostat SERVER whs

6.2 Set がたまに失敗する

memcached のクライアントライブラリの実装によっては、デフォルトのタイムアウト時間が非常に短く設定されていることがあります。Set 操作は比較的時間がかかるので、タイムアウトしてしまっている可能性があります。memcached のクライアントライブラリの設定で、タイムアウト時間を長くしてみてください。

6.3 Set できるのに Get できない

kumofs は、Set/Delete 操作用と Get 操作用の２つのルーティング表を使います。通常この２つは同じですが、サーバーの追加や復旧の処理中に通信エラーが発生すると、2 つのルーティング表が食い違ってしまうことが有り得ます。**kumostat** コマンドの **hscheck** サブコマンドを使うと、２つのルーティング表が食い違っていないかを確認することができます。もし食い違っていた場合は、**kumoctl** コマンドの **replace** サブコマンドを使うと修復できます。

6.4 データベースファイルには保存されているのに取得できない key がある

kumofs はデータを複数の kumo-server に分散して保存しますが、保存するときに所定の kumo-server に保存し、取得するときは所定の kumo-server に保存されていることを前提とします。このため何らかの理由でデータが所定の kumo-server とは違う kumo-server に保存されていると、そのデータはデータベースファイルには保存されていても取得できません。**kumoctl** コマンドの **full-replace** サブコマンドを使うと修復できます。ただし full-replace を使うと大きなネットワークトラフィックが発生するので注意してください。

6.5 kumo-server が 3 台以上ダウンしてしまった

kumo-server ダウンしてしまっても、データベースファイルが残っていればデータを復旧することができます。
TODO

Chapter 7

コマンドリファレンス

7.1 configure フラグ

–**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

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

–**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

7.2 共通のコマンドライン引数

-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。**-**を指定すると標準出力に出力する。省略するとログに色を付けて標準出力に出力する

-v WARN よりレベルの低いメッセージを出力する

-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

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

-Ys <**sec**> connect(2) のタイムアウト時間を指定する。単位は秒で、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**> ワーカースレッドの数を指定する

-TW <**num**> 送信用スレッドの数を指定する

7.3 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

-a Server が追加・離脱されたときに、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

-Rs 自動的な再配置が有効なときに、サーバーの追加・離脱を検出してからレプリケーションの再配置を開始するまでの待ち時間を指定する。単位は秒

7.4 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**[**#xmsiz=SIZE**][**#rcnum=SIZE**]> 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

7.4.1 削除済みフラグの回収

Delete 操作は実際には削除済みフラグを Set する操作です。しかし削除済みフラグは時間が経過すると回収され、本当に削除されます。削除済みフラグが回収されると一貫性は保証されません。削除済みフラグは以下の条件で回収されます：

Delete してから一定の時間が経過した この時間は kumo-server の**-gX** オプションで指定できます。

Delete フラグを記憶するメモリ使用量の上限に達し、かつ Delete してから一定の時間が経過した この時間は kumo-server の**-gN** オプションで指定できます。メモリ使用量の上限は**-gS** オプションで指定できます。

削除済みフラグを記憶するメモリ使用量の上限に達したが、Delete してから一定の時間が経過していない場合は、削除済みフラグはデータベースファイルの中に放置されます。放置された削除済みフラグは、次に再配置操作が行われたときに回収されます。

7.5 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

-b <**port**> memcached バイナリプロトコルを待ち受けるポート番号を指定する（EXPERIMENTAL）

-G <**number**> Get 操作の最大リトライ回数を指定する

-S <**number**> Set 操作の最大リトライ回数を指定する

-D <**number**> Delete 操作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

7.5.1 非同期レプリケーション

kumofs ではデータを set したり delete したりするときレプリケーションを行いますが、デフォルトではレプリケーションが完了するまで待ってから（すべてのサーバーから応答が帰ってきてから）アプリケーションに応答が返されます。これを 1 台の kumo-server に書き込みが完了した時点で応答を返すようにすると（非同期レプリケーション)、更新系の応答時間が大幅に短縮されます。ただし非同期レプリケーションを有効にすると、成功応答が帰ってきたとしても、必ずしもレプリケーションが成功していることが保証されず、そのため複数の kumo-server 間でデータの一貫性が保たれているとが保証されなくなります。Set 操作のレプリケーションを非同期にするには、kumo-gateway のコマンドライン引数に**-As** を、Delete 操作のレプリケーションを非同期にするには**-Ad** を追加してください。

7.6 kumoctl

kumoctl コマンドは kumo-manager に様々なコマンドを発行するための管理コマンドです。第一引数に kumo-manager のアドレスを指定し、第二引数にサブコマンドを指定します。

```
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
   full-replace               start full-replace (repair consistency)
   backup  [suffix=????????]  create backup with specified suffix
   enable-auto-replace        enable auto replace
   disable-auto-replace       disable auto replace
```

attach サブコマンドは、認識しているが登録されていない kumo-server を実際に登録します。**detach** サブコマンドは、fault 状態の kumo-server を切り離します。**attach-noreplace** サブコマンドは **attach** と同じですが、kumo-server を登録した後に key-value ペアの複製の再配置を行いません。**detach-noreplace** サブコマンドは **detach** と同じですが、kumo-server を切り離した後に複製の再配置を行いません。**replace** サブコマンドは複製の再配置だけを行います。attach-noreplace サブコマンドと detach-noreplace サブコマンドは attach と detach を同時に行いときのみ使用し、すぐに replace サブコマンドを使って再配置を行ってください。再配置を行わないまま長い間放置してはいけません。**backup** サブコマンドは、データベースファイルのバックアップを作成します。バックアップは認識しているすべての Server ノード上で作成されます。バックアップファイルのファイル名は、元のデータベースファイル名に第三引数で指定した suffix を付けたファイル名になります。suffix を省略するとその日の日付 (YYMMDD) が使われます。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 つのファイルにまとめることができます。

7.7 kumostat

FIXMEkumostat コマンドを使うと kumo-server の状態を取得することができます。第一引数に kumo-server のホスト名とポート番号を指定し、第二引数にコマンドを指定します：

```
Usage: kumostat server-address[:port=19800] command
       kumostat -m manager-address[:port=19700] command
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
rhs                     get rhs (routing table for Get)"
whs                     get whs (routing table for Set/Delete)"
hscheck                 check if rhs == whs
set_delay               maximize throughput at the expense of latency"
unset_delay             minimize latency at the expense of throughput"
```

-m に続いて kumo-manager のアドレスを指定すると、kumo-manager から kumo-server 一覧を取得し、attach されていて active なすべての kumo-server の状態を表示します。

pid kumo-server プロセスの pid

uptime kumo-server プロセスの起動時間（単位は秒）

time kumo-server プロセスが走っているホストの UNIX タイム

version kumo-server のバージョン

cmd_get Gateway ノードからの Get 操作を処理した回数

cmd_set Gateway ノードからの Set 操作を処理した回数

cmd_delete Gateway ノードからの Delete 操作を処理した回数

items データベースに入っているエントリの数

rhs Get に使われるルーティング表

whs Set/Delete に使われるルーティング表

hscheck rhs と whs が同じかどうかチェックします

set_delay 遅延を犠牲にして最大スループットを最大化する

unset_delay 最大スループットを犠牲にして遅延を最小化する

rhs と whs が食い違っている場合は、再配置を実行中か、前回の再配置が失敗している可能性があります。

7.8 kumotop

FIXMEkumotop コマンドを使うと kumo-server の状態を定期的に更新しながら表示することができます。引数に監視したい kumo-server のアドレスを指定します。kumo-server のアドレスは複数指定できます：

```
Usage: kumotop server-address[:port=19800] ...
       kumotop -m manager-address[:port=19700]
```

-m に続いて kumo-manager のアドレスを指定すると、kumo-manager から Server 一覧を取得し、attach されているすべての Server ノードの状態を表示します。

7.9 kumomergedb

FIXMEkumomergedb コマンドを使うと、複数のデータベースファイルを 1 つにまとめることができます。第一引数に出力先のファイル名を指定し、第二引数以降にまとめたいデータベースファイルを指定します。

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

7.10 kumohash

kumohash コマンドを使うと、ある key がどの kumo-server に保存されるかを計算することができます。

```
Usage: kumohash server-address[:port=19800] ... -- command [options]
       kumohash -m manager-address[:port=19700] command [options]
command:
   hash  keys...              calculate hash of keys
   assign  keys...            calculate assign node
   dump                       dump hash space
```

hash key のハッシュ値を計算する

assign key がどの kumo-server に保存されるかを計算する

dump Consistent Hashing のルーティング表を表示する

7.11 kumolog

FIXMEkumolog コマンドはバイナリ形式のログを人間にとって読みやすいテキストに変換して表示します。

```
kumolog [options] <logfile.mpac>
```

-f, –follow **tail -f** と同じ効果

-t, –tail 最後の N 個のログだけ表示する（デフォルト: N=10）

-h, –head 最初の N 個のログだけ表示する（デフォルト: N=10）

-n, –lines=[-]N N を指定する

7.11.1 ログ

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

テキストログ 行区切りのテキストフォーマットのログ。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バイナリ形式のログ

テキストログは常に出力されます。**-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ります。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ません。デフォルトでは優先度によってログに色が付きますが、**-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ります。

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ます。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ます。

Chapter 8

チューニング

8.1 データベースファイルのチューニング

Tokyo Cabinet のハッシュデータベースのチューニングによって、性能が大きく変わります。データベースをチューニングするには、kumo-server を起動する前に、**tchmgr** コマンドを使ってデータベースファイルをあらかじめ作成しておきます。tchmgr コマンドのパラメータについては、Tokyo Cabinet のドキュメントを参照してください。Tokyo Cabinet のパラメータのうち、拡張メモリマップのサイズ（xmsiz）とキャッシュ機構（rcnum）は kumo-server のコマンドライン引数で指定します。kumo-server の**-s** オプションで、データベースファイル名の後ろに**#xmsiz=XXX** と指定すると拡張メモリマップのサイズを指定できます。**#rcnum=XXX** と指定するとキャッシュ機構を有効化できます。

```
[svr1]$ kumo-server -v -m mgr -l svr1 -s "database.tch#xmsiz=600m#rcnum=4k"
```

8.2 スレッド数のチューニング

CPU のコア数が多い場合は、kumo-server や kumo-gateway のワーカースレッドの数（-TR 引数）を増やすと性能が向上します。ハードウェアスレッド数+2 くらいが目安です。デフォルトは 4 です。保存するデータのサイズが大きい場合は、kumo-server や kumo-gateway の送信用スレッドの数（-TW 引数）を増やすと性能が向上する可能性があります。デフォルトは 2 です。

Chapter 9

FAQ

9.1 kumofs の名前の由来は？

kumo は空に浮かぶ**雲**を意味しています。**fs** は fast storage の略です。

9.2 サーバーに障害が発生したと判断される基準は？

メッセージを送ろうとしたところ、接続済みのすべての TCP コネクションでエラーが発生し、再接続を試みても失敗して再接続のリトライ回数が上限に達したら、そのノードはダウンしたと判断します。TCP コネクションが切断されただけではダウンしたとは判断せず、メッセージの送信に失敗しても制限回数以内に再接続することができたら、メッセージは再送されます。kumo-server と kumo-manager は常に keepalive メッセージをやりとりしており、いつもメッセージを送ろうとしている状態になっています。kumo-manager は kumo-server がダウンしたらできるだけ早く検出して fault フラグをセットし、正常なアクセスを継続させます。

9.3 どの kumo-server にデータを保存するかを決めるアルゴリズムは？

kumofs は Consistent Hashing と呼ばれるアルゴリズムを利用しています。ハッシュ関数は SHA-1 で、下位の 64 ビットのみを使います。1 台の物理ノードは 128 台の仮想ノードを持ちます。

データを取得するときは、kumo-gateway が key にハッシュ関数を掛けてハッシュ表から担当ノードを計算し、担当ノードからデータを取得します。取得に失敗したときは、ハッシュ表上でその次に当たるノードから取得します。それでも失敗したらその次の次のノードから取得します。それでも失敗したら最初の担当ノードに戻ってリトライします。

データを変更するときは、kumo-gateway が key にハッシュ関数を掛けてハッシュ表から担当ノードを計算し、担当ノードにデータを送信します。取得する場合とは異なり、次のノードにフォールバックすることはありません。

担当ノードは Set や Delete 操作を受け取ると、データをハッシュ表上で次のノードと、次の次のノードにレプリケーションします。

担当ノードは kumo-gateway からリクエストを受け取ったとき、本当に自分が担当ノードであるかどうかを自分が持っているハッシュ表を使って確認します。間違っていた場合はリクエストを拒否します。このように必ず特定の担当ノードだけがデータを変更でき、他のノードが同じタイミングで同じ key-value を変更することがないようになっています。

担当ノードを選ぶとき fault フラグがセットされているノードはスキップします。このため一部の担当ノードがダウンしている状態でも正常なアクセスを続けられます。