

Data Segment の分散データベースへの応用

大城 信康 杉本 優 河野 真治

当研究室では並列・分散プログラミングスタイルとして Data Segment, Code Segment によるプログラミング手法を提案している。Data Segment, Code Segment の Java 上の実装として Alice を作成してきた。Alice は分散管理フレームワークであり, Alice を用いることで分散プログラムが比較的楽に行える。そこで, 当研究室で開発している非破壊的木構造データベース Jungle の分散実装を Alice を用いて行った。本研究では, Alice に Jungle の分散実装を通じて, データベースノード間の通信を行う利点と欠点について考察する。

1 動機

IT システムが巨大化していくにつれ, 障害発生事例が社会に与える影響もより大きな物となる。それに伴い, IT システムにおけるディペンダビリティへの注目が増している。

そこで, DEOS プロジェクトは IT システムにおけるディペンダビリティを担保する技術体系をまとめ, 制度化, さらに事業化を目指している。DEOS プロジェクトは 2006 年に独立行政法人科学技術機構 (JST) は CREST プログラムの 1 つとして始まったプロジェクトである。[?]

DEOS プロセスにおいて, 全てのデータを保持する D-ADD (DEOS Agreement Description Database) と呼ばれるデータベースがある。D-ADD には様々なデータが入れられるため, データベースとしてのスケーラビリティが求められる。また, D-ADD は過去のデータを参照してディペンダビリティをあげるデータの抽出といった機能も求められる。本研究は

D-ADD に必要となるスケーラビリティとデータ保持についての研究も兼ねている。

1.1 研究の目的と背景

当研究室では並列・分散プログラムに向けたプログラミングを目指し, データを Data Segment, タスクを Code Segment という単位で扱うプログラミングスタイルの提案を行なっている。そこで Data Segment, Code Segment によるプログラミングを提供する実装として, Java による分散ネットワークフレームワーク Alice を開発している。Alice はノード間のトポロジー生成を提供しており, Data Segment としてデータの送受信をノード間で行うことができる。

また, 当研究室では非破壊的木構造を用いたデータベースである Jungle の開発も行なっている。Jungle はデータを非破壊で保持することでスケーラビリティのあるデータベースを目指している。Jungle はデータの編集を TreeOperationLog という単位で行う。Alice を使いこの TreeOperationLog を各ノード間で送受信することでデータの分散を行うことができる。

本研究では, Alice と Jungle を用いて分散データベースの実装を行う。さらに, 例題のアプリケーションとして掲示板を作成し, 評価を行う。

Nobuyasu OSHIRO, Yu SUGIMOTO, Shinij KONO, 琉球大学大学院理工学研究科情報工学専攻並列信頼研, Dept. Concurrency Reliance Laboratory, Information Engineering Course, Faculty of Engineering Graduate School of Engineering and Science, University of the Ryukyus.

2 分散ネットワークフレームワーク Alice

Alice は当研究室で開発している分散管理フレームワークである。Data Segment と Code Segment による並列・分散プログラミングを提供する。

まず, Data Segment と Code Segment についての説明を行う。

2.1 Data Segment

Data Segment は計算に必要なデータになる。Alice は Data Segment を文字列の Key で管理する。Key 毎にリストが用意され, put された順番で Data Segment は取り出され計算が行われる。Data Segment は Data Segment Manager(以下 DSM) により管理される。DSM はノード毎にキーを持つ。他のノードの DSM にアクセスする場合は Remote DSM 経由で行う。Alice による分散プログラミングはこの Remote DSM の機能を使用する。

Data Segment Manager は API を提供しており, この API を通じて Data Segment のやりとりが行われる具体的には以下の API が用意されている。

- `void put(String key, Value val)`

put は Data Segment をリストへと追加する API である。

- `void update(String key, Value val)`

update はリストに入っている Data Segment を更新する API である。

- `void peek(Receiver receiver, String key)`

peek はリストに入っている Data Segment を取り出す API である。peek により取り出された Data Segment はリストより削除されない。

- `void take(Receiver receiver, String key)`

take はリストに入っている Data Segment を取り出す API である。取り出した Data Segment はリストより削除される。

2.2 Code Segment

Code Segment は Data Segment を受け取り計算を行うコードのことを示す。並列プログラミングにおけるタスクにあたる。Code Segment と Data

Segment は対になっている。Code Segment は計算に使う Data Segment のキーを登録して, そのキーにあたる Data Segment が用意され次第処理が実行される。Code Segment が処理を開始するのに必要な Data Segment を Input Data Segment という。

Code Segment では Data Segment の生成を行い, put や update により新たにリストに登録することができる。Code Segment 内で作成し登録される Data Segment は Output Data Segment と呼ばれる。

Code Segment は Input Data Segment と Output Data Segment の API を提供する (図 1)。

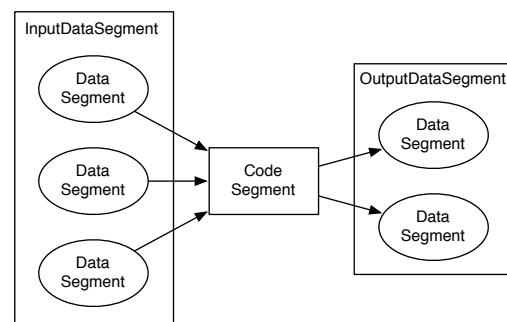


図 1 Data Segment と Code Segment

2.3 MessagePack

Alice における Data Segment のデータ表現には MessagePack を利用している。MessagePack はバイナリをベースにしたシリアライズライブラリーである。また, MessagePack のバイナリにシリアライズできる型のみで構成された Value オブジェクトが用意されている。MessagePack は Java の基本的な型はシリアライズすることができる。

Value オブジェクトは自己記述なデータ形式になっている。独自のクラスでも @Message アノテーションを付けることで Value 型へと変換することができる。その時は MessagePack がシリアライズできる型のみをフィールドに入れなければならない。

3 非破壊的木構造を用いたデータベース Jun-

gle

Jungle はスケーラビリティのある CMS の開発を目指して当研究室で開発されている非破壊的木構造データベースである。一般的なコンテンツマネジメントシステムではログツールや Wiki・SNS が多く、これらのウェブサイトの構造は大体が木構造であるため、データ構造として木構造を採用している。

ここではまず破壊的木構造と、非破壊的木構造の説明をし、Jungle におけるデータ編集の実装について述べる。

3.1 破壊的木構造

破壊的木構造の編集は、木構造で保持しているデータを直接書き換えることで行う。図 2 は破壊的木構造の編集を表している。

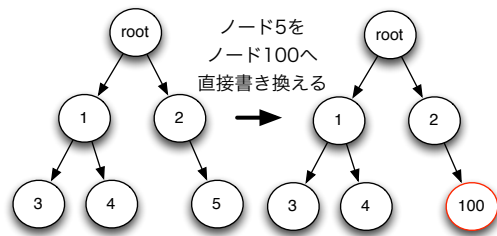


図 2 破壊的木構造の編集

破壊的木構造は、編集を行う際に木のロックを掛ける必要がある。この時、データを受け取ろうと木を走査するスレッドは書き換えの終了を待つ必要があり、閲覧者がいる場合は木の走査が終わるまで書き換えをまたなければならない。これではスケールしにくいと考えられる。

3.2 非破壊的木構造

非破壊的木構造は破壊的木構造とは違い一度作成したデータを破壊することはない。非破壊的木構造においてデータの編集を行う場合は、root から編集のあったノードまでコピーを行い新しく木構造を作成することで行う。編集が行われない部分は参照をもたせ

る。図 3 は非破壊的木構造の編集を表している。

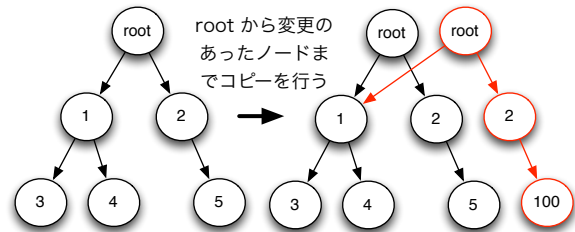


図 3 非破壊的木構造の編集

非破壊的木構造により、木構造を編集しながら走査することが可能となる。よって、破壊的木構造よりスケールすると考えている。

3.3 Jungle におけるデータ編集

木の編集は、通常 Node を書き換えるため Node の API として提供されることが多いが、Jungle では JungleTreeEditor を利用して行う。JungleTreeEditor には編集するためのいくつかのメソッドが用意されており、NodePath と呼ばれるルートノードからノードまでのパスを指定することでノードが編集される。NodePath は、ルートノードからスタートし、ノードの子供の場所を次々に指定していくことで編集対象のノードの場所を表す (図 4)。

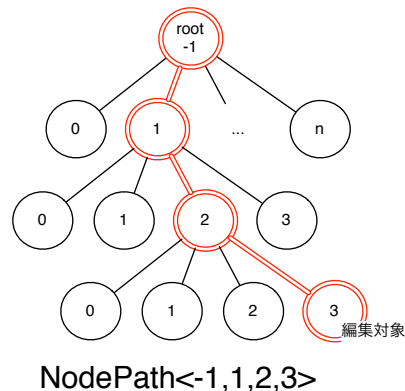


図 4 NodePath

Tree 編集の API として次の 4 つが用意されている。

- `addNewChildAt(NodePath _path,int _pos)`

`NodePath` で指定された `Node` に子供となる `Node` を追加する API である。 `pos` で指定された番号に子供として追加を行う。

- `deleteChildAt(NodePath _path,int _pos)`

`NodePath` と `pos` により指定される `Node` を削除する API である。

- `putAttribute(NodePath _path, String _key,ByteBuffer _value)`

`Node` に `attribute` を追加する API である。 文字列をキーにして `ByteBuffer` によりデータを保持する。

- `deleteAttribute(NodePath _path, String _key)`

`NodePath` により指定される `Node` の `attribute` を削除する API である。 削除する `attribute` は文字列のキーで指定する。

3.4 TreeOperationLog

上記の API を使用すると `Editor` 内部では `NodeOperation` として順次つまれていき、最終的に `commit` されることで編集が行われる。 複数の `NodeOperation` の集まりを `TreeOperationLog` といい、これが編集の単位となる。 例えば、後述する掲示板の実装では 1 つの書き込みに対して 1 つの `Node` を作成し、`attribute` をもたせている。 その時のログは次のようになる。

```
[APPEND_CHILD:<-1>:pos:1]
[PUT_ATTRIBUTE:<-1,1>:key:author,value:oshiro]
[PUT_ATTRIBUTE:<-1,1>:key:mes,value:hello]
[PUT_ATTRIBUTE:<-1,1>:key:key,value:hoge]
[PUT_ATTRIBUTE:<-1,1>:key:timestamp,value:0]
```

大文字の英字は実行した API を表す。 `<>` により囲まれている数値は `NodePath` を示す。 `NodePath` の後ろは `position` や `attribute` の情報を表している。 `NodeOperation` と `NodePath` の組み合わせを `TreeOperation` として扱い、それらいくつか集まりが `TreeOperationLog` となる。

4 Alice を用いた Jungle の分散実装

Alice を用いた `Jungle` のデータ分散は、上記の `TreeOperationLog` を `Data Segment` として扱い他ノードがその `Data Segment` にアクセスできるようにすることで行える。 そのために必要なことは以下となる。

- トポロジーの形成
- `TreeOperationLog` を `MessagePack` によりシリアライズ
- `TreeOperationLog` を扱う `Data Segment` の作成

4.1 トポロジーの形成

Alice はトポロジーの形成を行う機能を提供している。 トポロジー設定用ファイルに従い、ノード同士を接続させる。

トポロジー設定用ファイルは `DOT Language` で記述される。 例えば、5 ノード 2 分木のノードを組みたいときは次のようなファイルになる。

```
digraph test {
node0 -> node1 [label="child1"]
node0 -> node2 [label="child2"]
node1 -> node0 [label="parent"]
node1 -> node3 [label="child1"]
node1 -> node4 [label="child2"]
node2 -> node0 [label="parent"]
node3 -> node1 [label="parent"]
node4 -> node1 [label="parent"]
}
```

これにより形成されるトポロジーを図??に示す。

子供となるノードは "parent" キーにより親の `DSM` (`Remote DSM`) にアクセスすることができる。 また、親も子供となるノードの `DSM` に対して "child1" や "child2" キーによりアクセスすることが可能となる。

Alice ではスクリプトが用意されており、ノードの数と子供の数を指定するだけで `dot` ファイルの作成が行える。 このように、Alice では比較的楽にトポロジーの形成が行える。

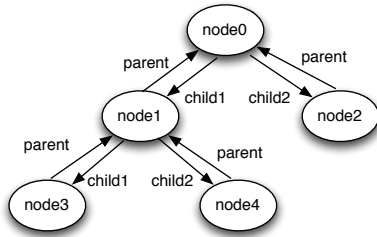


図 5

4.2 TreeOperationLog の MessagePack によるシリアライズ

TreeOperationLog は MessagePack でシリアライズできないものを List のフィールド変数に保持している。そのため、TreeOperationLog はそのままでは MessagePack でシリアライズすることはできない。フィールドとして保持しているものは全て MessagePack でシリアライズできるものにする必要がある。そこで、フィールドで保持しているものを Value 型に変換するための Container クラス作成をそれぞれ行った。ログに関連するクラス全てをシリアライズするクラスを行った後に、それら全てをまとめる DefaultTreeOperationLogContainer クラスの作成を行った。このクラスは TreeOperationLog を Value 型へと変換しフィールド変数で保持する。実際に TreeOperationLog のシリアライズを行うソースを次に示す。

```

public void unconvert(Iterable
  <TreeOperation> _log) throws IOException{
  MessagePack msgpack
  = SingletonMessage.getInstance();
  List<Value> list
  = new LinkedList<Value>();
  for(TreeOperation op : _log) {
    NodeOperation nOp
    = op.getNodeOperation();
    NodePath nPath
    = op.getNodePath();
    DefaultTreeOperation treeOp
  = new DefaultTreeOperation(nPath, nOp);

```

```

    DefaultTreeOperationContainer c
  = new DefaultTreeOperationContainer();
  c.unconvert(treeOp);
  Value v = msgpack.unconvert(c);
  list.add(v);
  }
  Value listValue
  = msgpack.unconvert(list);
  logValue = listValue; // field variable
  }

```

上記のソースでは主に MessagePack により List<TreeOperation> で保持していた TreeOperation を List<Value> へと変換させている。SingletonMessage は使用する MessagePack を Singleton で使用するためのクラスである。これにより何度も MessagePack を new することを防いでいる。

また、Data Segment にするにあたり、TreeOperationLog の保持だけでなく、編集した木の名前やリビジョン番号、変更を行ったノードの情報をノードの名前といった情報も保持させる。DefaultTreeOperationLogContainer の生成から Data Segment へ put を行なっているソースを次に示す。

```

DefaultTreeOperationLogContainer container
  = new DefaultTreeOperationLogContainer();
  container.setTreeName(_treeName);
  container.setUUID(_uuid);
  container.setUpdaterName(_updaterName);
  container.setRevision(nextRevision);
  container.setPosition(pos);
  container.unconvert(_log);
  container.setTimeStamp(timestamp);
  HashLogUpdateCodeSegment cs
  = new HashLogUpdateCodeSegment(
    container.getHashLogString());
  cs.ods.put("log", container);
  cs.ods.put により Data Segment のリスト "log" に対してログの put を行う。

```

4.3 ログを扱う Data Segment

Alice の各ノードは "log", "childLog" というキー

でログを扱う (図 5).

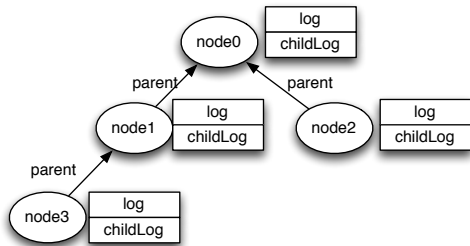


図 6 形成されるトポロジと Data Segment

”log”にはそのノードが行った木の編集のログが入る。また、子供となるノードは”parent”というキーを使うことで親ノードの Data Segment Manager にアクセスすることができる。子供となるノードは親の”log”を待ち反映する Code Segment (LogUpdateCodeSegment) を走らせており、ログが put されるとそのデータを受け取り Code Segment の処理が行われる (図 6).

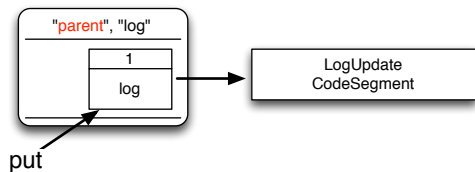


図 7 親ノードの更新を行う Code Segment

”childLog”には子供となるノードが行った編集のログが入れられる。ノードは”childLog”の Data Segment にデータが入るの待っている Code Segment が常に走らせており、子供が行った木の編集が”childLog”に put されることで親へとデータの伝搬が行われる (図 7).

4.4 Merge algorithm の設計

Jungle はログの衝突が起きた場合に、Merge を行うことで衝突を解決する。今回実装した掲示板における Merge algorithm は単純な実装である。書き込み

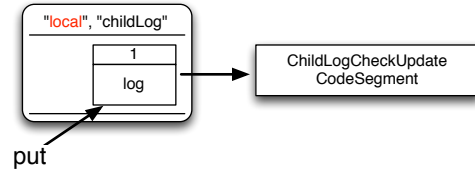


図 8 子供のノードの更新を行う Code Segment

のタイムスタンプと既書き込まれたデータのタイムスタンプを比べ、ソートを行うだけである。

掲示板における Merge はそれで十分である。しかし、ブログや Wiki といった CMS を設計するさいにはもっと複雑な Merge になる。どのように Merge algorithm を実装していくかはよく考えて行わなければならない。

5 Cassandra との比較

Cassandra は複数のサーバで動作を想定した分散データベースである。Cassandra は分散 Key-Value ストアデータベースであり、Dynamo?[] と BigTable?[] を合わせた特徴を持っている。データの Read と Write に対して Consistency Level を設定することができるのが特徴である。Write に関してはデータの書き込みが全体、過半数、もしくは 1 つのノードに書き込まれたかどうかの整合性の設定ができる。Read に関しては最初にデータを持っていたノードか、全体を検索して最新のタイムスタンプを確認するかといった整合性の設定ができる。Consistency Level にもよるがつまり Cassandra はデータについて複数のノードに問い合わせていることになる。

5.1 Jungle のデータ要求

Jungle ではデータの要求が行われた場合、手元にあるデータを返す。データの編集が行われた場合は、他ノードへとログを伝搬していく (図 9).

この時、別のログと衝突が起きた場合は衝突を検出したノードが Merge を行う。その後 Merge を行ったログをまた他ノードへと伝搬させていく。

Jungle は非破壊により過去のデータを持っているため Merge を行うことができる。それにより最新の

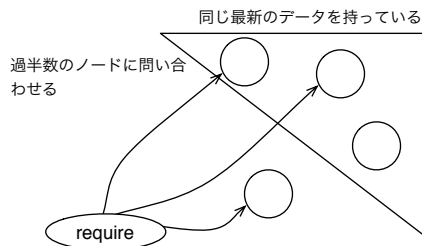


図 9 Cassandra におけるデータの更新・読み込み

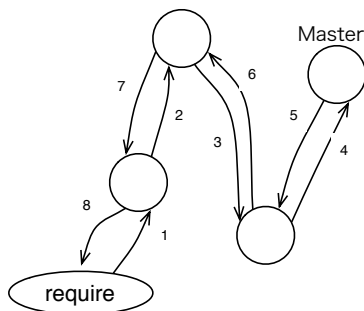


図 10 Jungle のデータ伝搬

データでなくてもデータを返すことができ、それがスケラビリティに繋がると考えている。

6 掲示板による Jungle の性能評価

Jungle の例題のアプリケーションとして掲示板を作成した。この掲示板は組み込みウェブサーバーである Jetty をフロントエンドにしたウェブアプリケーションである。今回作成した分散バージョンと元のバージョンをシングルサーバ上で動かし、2つのベンチマークをとり性能比較を行う。ベンチマークは学科が提供する VMWare のクラスタを用いて行う。

6.1 実験方法

複数のクラスタから並列に 3000 回アクセス (HTTP Reauest) を行い、それぞれのクラスタの平均時間をとる。クラスタ 1 台から 45 台まで順次並列にアクセスを行いグラフ化する。

6.2 実験環境

掲示板を動かすサーバーの環境を図??に示す。

表 1 掲示板を動かすサーバーの仕様

名前	概要
CPU	Intel®Xeon®CPU X5650@2.67GHz*2
物理コア数	12
論理コア数	24
Memory	132GB
OS	Fedora 16
JavaVM	1.6.0_39-b04

サーバにアクセスを行うクラスタの使用を図??に示す。

表 2 検証に利用する VMWare クラスタの仕様

名前	概要
CPU	Intel®Xeon®CPU X5650@2.67GHz
Memory	8GB
OS	CentOS 5.8
HyperVisor	VMWare ESXi

6.3 実験結果

7 まとめ

今回, Alice を用いて Jungle の分散データベースの実装を行った。Jungle の編集のログを Data Segment として Alice 上で他ノードへと送ることで実装することができた。Alice にはトポロジーの形成の機能などがあり、比較的楽に分散データベースの実装を行うことができた。

実装を行った分散データベースで掲示板の作成をし、元の Jungle との性能比較も行った。しかし、今回はシングルサーバ上でのテストのみであり、分散させた状態でのベンチマークをとることはできなかった。次回は、Cassandra でも同じ分散プログラムを作り、分散環境における性能比較を行いたい。それには分散データベースにおけるベンチマークの取り方の検証から行わなければならない。

また、Alice 内部では `java.util.concurrent.Executor`

を使用している. そのため, Alice を使用しての分散
プログラムは Alice 自身の Thread Pool との調和も

考えて行く必要もある. Alice の Thread Pool との
競合についても検証していきたい.