Database Jungle に関する研究

115731 金川竜己　指導教員：指導教員名

1　研究目的

当研究室ではデータの編集の際に一度木構造として保存したデータには触れず、新しく木構造を作成してデータの編集を行う非破壊的木構造を用い¿ たデータベースである Jungle を開発している。

業務システムでアカウント管理を行う際に既存の Database では、木構造のデータ型を入れる処理の煩雑さや、過去の version におけるデータの参照が出来ないと言った問題がある。

Jungle は非破壊で過去のデータを変更しないので、過去のデータを参照することができる。Jungle は木構造のデータベースなので、木構造のデータをそのまま格納出来る。

本研究では Jungle に、検索 API、Index、過去データの参照の実装を行った。さらに、当研究室と共同研究を行っている Symphonies 社が開発した、アカウント管理システム maTrix に Jungle を組み込む。

2　maTrix

maTrix とは Symphonies 社が開発した、アカウント管理、許諾判定システムのことである。

maTrix は人、役職、役割、権限と言った木構造の組織、ポリシーファイルの２つのデータを持っている。maTrix が保持している人、役職、役割等のデータはお互いに参照している。ポリシーファイルは、組織の中で申請等を行った際に、どの権限によってその申請が許諾されるのかを指定している。

組織のデータ、ポリシーファイル共に木構造のデータであるため、木構造のデータベースである Jungle には、そのまま格納できる。

また、maTrix は、いつ、誰が、どんな申請をしたか、と言った過去の承認情報を保持するので、過去の組織のデータを参照する必要が出てくる。よって過去のデータの参照が出来る Jungle と相性が良い。

maTrix はデータを xml 形式で出力することが出来る。

xml 形式で出力された maTrix のデータを Jungle に格納するために、SAX を用いて、Jungle 用の xmlReader を作成した。xmlReader を作成したことにより、実際に maTrix から出力されたデータを Jungle に取り込み、テストを行うことを可能にした。

引数	関数での使われ方
Query	探索の条件を記述した Interface(図 2)
Key	後述する Index を使うために使用する
value	後述する Index を使うために使用する

3　検索 API の実装

Java8 の新機能である lambda 式を用いてデータの検索を行う find 関数の実装を行った。

図 1 に find 関数の使い方を示したソースコードを記述する。

図 1

```
Iterator<Pair<TreeNode, NodePath>> pairPersonIterator =
                        traverser.find((TreeNode node) -> {
  String element = node.getAttributes().getString("element");
  if (element == null)
    return false;
  if (element.equals("Person"))
    return true;
  return false;
}, "element", "Person");
```

find 関数は引数に Query、String key、String value の 3 つを取る。条件に一致した Node と、その Node までの Path を１つにまとめた Pair の Iterator を返す。

図 2

```
public interface Query {
  boolean condition(TreeNode node);
}
```

find 関数は、まず初めに、Index があるかどうかを調べる。index がある場合は Index を使用し探索を行う。Index がない場合は、Index を作成しながら Tree を全探索する。

上記の Jungle の Query 部分のソースコードは、”Person”というデータを持った Node と、その Node までの Path の Pair の iterator を返す。 検索 API は、他に特定の Node 以下に対して検索を行う findInSubTree(Query,node,key,value) も実装した。

4　Index の実装

Jungle の探索は Tree を全探索するので、探索の計算量は O(n) となり、非常に効率が悪い。そこで、Index を使用することで、探索効率の向上を計った。Index の実装には、functionalJava の TreeMap を使用した。

TreeMap は、Key と Value のペアを用いて赤黒木を構築

する。赤黒木は、ソート済み二分木の探索なので計算量がO(logN) である。さらにデータ編集時の最悪計算量が、他の木構造と比べ最善のものの１つであるので、安定した速度でデータの編集が行える。また、FunctionalJava の TreeMap は immutable なので一度作られた Tree に対して更新が行われない。つまり新しい要素を追加する際は、新しく TreeMap を作ることになる。なので Tree は、各 version ごとに固定の Index を持つことが出来る。また、新しく Tree を作る際に、過去の Tree の一部を再利用するのでメモリの使用量を抑えることが出来る。

Index は各ユーザーがローカルに Index を持つ on the fly 形式で実装する。

図 3

```
TreeMap<String key,
 TreeMap<String value,List<Pair<TreeNode,NodePath>>>>
```

図 3 は Index の型である。

最初の TreeMap<String key,TreeMap> は Index を格納する TreeMap で、この TreeMap に対し key で get を行うと、key に対応する Index を取得できる。取得した Index に対し、value で get を行うと、value の値を持つ Node と、その Node までの Path の２つを Pair の List が返ってくる。

5 IndexJungleTreeEditor の実装

Index の更新は IndexEditor を用いて行う。図 4 に IndexEditor の定義を記述する
IndexEditor の edit は Index の更新を行い、Index 更新後の IndexJungleTreeEditor を返す。Jungle で Tree の編集を行う際は、JungleTreeEditor を使用し、Node の add、delete、値の put、delete を行う。Tree に対して変更を加えると、それに伴い、Index も更新する必要が出てくる。そこで JungleTreeEditor の機能を拡張し、IndexJungleTreeEditor を作成した。
IndexJungleTreeEditor では、Tree の更新と同時に IndexEditor を用いて Index の更新も行い、Tree に対して両方の更新を Commit する。

図 4

```
public interface IndexEditor {
  Either<Error, IndexJungleTreeEditor>
    edit(TreeNode root,TransactionManager txManager,
        TreeEditor editor,TreeOperationLog log,
        TreeMap<String, TreeMap<String,
        List<Pair<TreeNode,  NodePath>>>> index);
}
```

6 maTrix に必要な Query の作成

maTrix は、許諾判定を行う際に、組織構造に対して Query を発行する。maTrix と Jungle の接続を行うにあたり、組織構造に対する Query は必要不可欠である。
maTrix に Jungle を接続するのに必要な Query 実装した。maTrix の Query を実装する際に、問題となったのが、特定の Node の以下の Tree に対して、index を使った検索である。Index には、Tree 全体のデータが入っているので、Index を使用した検索は必然的に Tree 全体に対する検索になってしまう。
この問題は、NodePath に、関数 compare() を実装し解決した。compare の定義を図 5 に示す。関数 compare は引数に比較対象の NodePath を受け取り、その Path が自分の下にあるかどうかを調べる関数である。
compare を使用することにより、特定の Node 下の探索を行う際、特定の Node の Path と Index で取ってきた結果に対し、compare を使い、フィルタリングを行うことで、Index を用いた特定の Node 下の探索を可能にした。

図 5

```
public boolean compare(NodePath path);
```

7 これから行うべきこと

実際に maTrix と Jungle の接続を行い、既存の maTrix と JunglemaTrix の性能を評価し、本当に早くなったのか確かめる。この時に Index の性能も評価する。

Jungle は、過去の version の Tree を保持しているので、過去の version の uuid を指定して自由に過去の version を参照できるようにする。

Jungle は比較的自由にデータを格納することができる。なので、Jungle を実際に使用する際に、どこまでを一つの木として扱うか、等をまとめた JungleDB の設計手法を確立させる必要がある。

参考文献

[1] 玉城将士 非破壊的木構造を用いた分散 CMS の設計と実装

[2] 大城信康 分散 Database Jungle に関する研究

[3] Eric Redmond and Jim R. Wilson 7 つのデータベース 7 つの世界