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

実際の業務システム等で、アカウント管理を行う際に既存の Database では、過去の version のデータを参照を行えなかったり、木構造のデータ型を入れる際に面倒だったり、問題がある。
そこで、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。
Jungle は非破壊で過去のデータを変更しないので、過去のデータを簡単に参照できたり、Jungle 自体が木構造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い、その後、当研究室と共同研究を行っている Symphonies 社が開発している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 社が開発しているアカウント管理、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 形式で格納することが可能なので、xml 形式で書き出された 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、本格的なテストが行えるようになった。

3　検索 API の実装

Jungle は、データを格納する API は実装されていたが、データの検索を行う API の実装は行われていなかった。
本研究では 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。
以下に find 関数の使い方を示したソースコードを記述する。

Jungle の Query 部分のソースコード

```
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 つの引数を取り、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。
第一引数には、探索の条件を記述する関数 boolean comdition(TreeNode) を定義した Interface　Query を。第二、第三引数の、String key、String value は後述する Index を使うために使用する。

Interface Query の定義

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

find 関数の処理の流れは、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。しかし、Index を使用することで効率よく探索を行えるようになる。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。赤黒木の長所として、ソート済み二分木の探索なので計算量が O(logN) であること、データ編集時の最悪計算量が

データ構造のうちで最善のものの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、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した際は、新しく TreeMap を作る。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、ということがあげられる。
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。

Index のデータの型

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

最初の TreeMap<String key,TreeMap> は Index を格納する TreeMap である。この TreeMap に対し key で get を行うと、key に対応する Index が登録されている場合、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 にまとめた List が返ってくる。

Index の Update
Index の更新は IndexEditor を用いて行う。
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 に対して両方の変更を Commit する。

IndexEditor の定義

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

5　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 を行う。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。
当研究では maTrix に Jungle を接続するのに必要な Query15 個を完成させた。maTrix の Query を実装するにあたって、問題となったのが、特定の Node の子供に対しての 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る。
この問題は、NodePath に、関数 compare() を実装し解決した。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

今回追加実装した compare の定義

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

6　これから行うべきこと

Index を使用した場合の性能評価
Index と maTrix の Query の実装が終わり、maTrix に接続する準備が整ったので、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。

過去の version の参照 API の実装
Jungle は、過去の version の Tree を保持しているので、簡単に参照できるはずである。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 の設計手法の確立
Jungle は比較的自由にデータを格納することができるので、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