Technical Report of iZENELib

Yingfeng Zhang, Kevin Hu

November 16, 2008

Abstract

This document presents the technical report for the project
iZENELib that could be used in the search engine developing process.
iZENELib is expected to contain two parts:AM-Lib, which takes charge
of storage, and IR-Lib, which takes charge of information retrieval and
machine learning.

Contents

1 Document History

Date	Author	Description
2008-11-02	Kevin	Initialize the design issue of AM-Lib.
2008-11-10	Yingfeng	Initialize the design issue of IR-Lib.
2008-11-10	Yingfeng	Create the technical report.

2   Design Goal

iZENELib plans to provide a collection of utilities which could be used in the search engine developing process. *iZENELib* is expected to be composed of two parts—the part taking charge of storage(AM-Lib), and the part in charge of information retrieval and machine learning(IR-Lib). AM-Lib is expected to be composed of two sub-parts, the one which provides common utilities for data storage, and the one providing corpus data management which serves for the IR-Lib. Generic design would be adopted largely in *iZENELib* to provide much more flexibility for component's reusing.

3   Initial architecture design of AM-lib

The future probable usages of AM-lib can be illustrated in several ways.

- In your project, you may want to compare the performance of using several different data structures. AM-lib makes this easy anyway.

```
#include "am/am.hpp''
class MyClass {

public:
        MyClass(AccessMethods<int, string>* pAm): pAm_(pAm){}

        void myFoo()
        {
                pAm_->insert(128, ''Hello! AM-lib'');
                pAm_->getDataBy(128);
                . . .
        }

private:
        AccessMethods<int, string>* pAm_;
}

////////////////////////////////////
#include "myclass.hpp''
#include "am/btree.hpp''
#include "am/rtree.hpp''
#include "am/am.hpp''
int main()
{
        AccessMethods<int, string>* pAm = new BTree<int, string>(...);
        MyClass test1(pAm);
        test1.myFoo();
        delete pAm;
        . . .
        pAm = new RTree<int, string>(...);
        MyClass test2(pAm);
        test2.myFoo();
        delete pAm;
        . . .
}
```

- You can make your modules more reuseable.

```
template<typename AmType>
void foo(AmType& am)
{
        am.insert(128, ''Hello! AM-lib'');
        am.getDataBy(128);
}

////////////////////////////////////
```

```
#include "am/btree.hpp''
#include "am/rtree.hpp''
int main()
{
        BTree<int, string> btree;
        foo(btree);
        . . .
        RTree<int, string> btree;
        foo(btree);
        . . .
}
```

- You don't need to worry about the size of data, cause AM-lib will use disk when data size comes very large.

```
                void foo(AmType& am)
                {
//initialize a 3 dimensions dynamic array
                MulDimDynArray largeArray(100000000, 100000000, 1000000);
                MulDimDynArray smallArray(10, 10, 10);
                smallArray.append(largeArray);
                . . .
                }
```
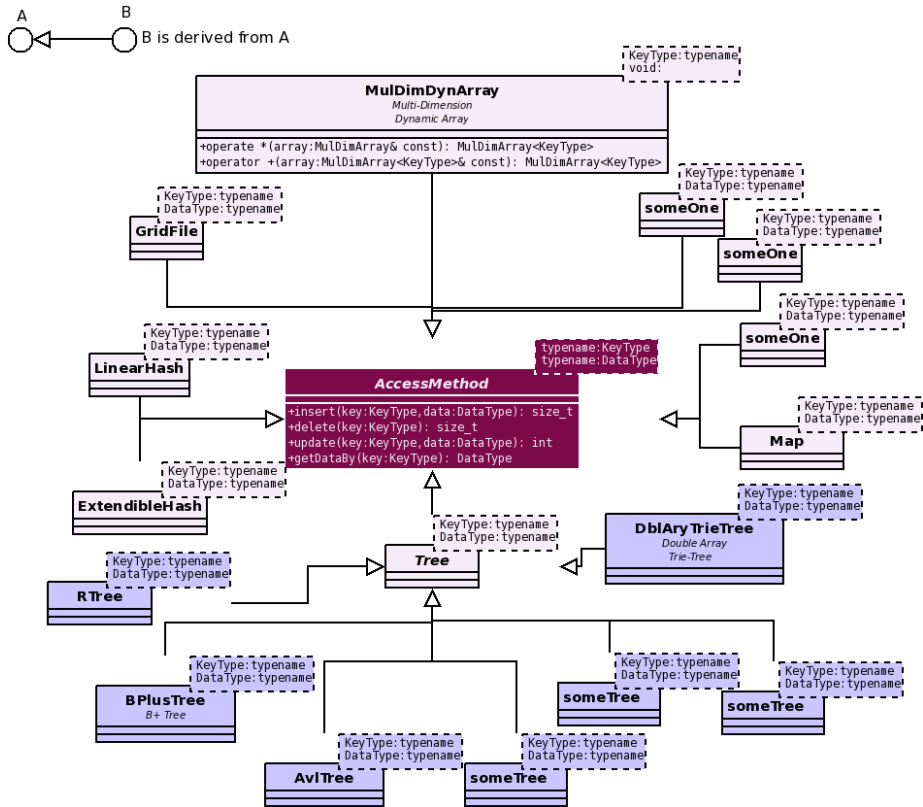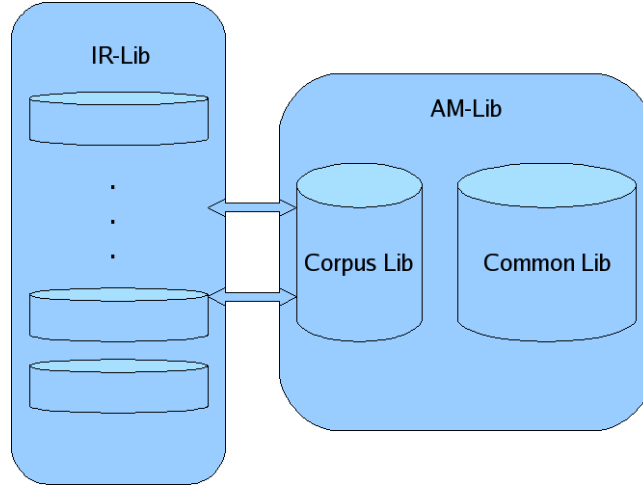

Figure 1: Initial design of AM-lib classes

Figure 2: AM-Lib and IR-LIB

4   Corpus Management

Corpus management component of AM-Lib provides storage services for IR-Lib. It is necessary because each component of IR-Lib will read data from corpus and output the results to a generic structs, therefore refactoring this common part into a single library will improve the system's reusage.

Existing project as *SML* provides similiar components as DOCUMENTBAG,etc. We can base corpus management on SML. What's more, IR-Lib needs a powerful matrix component to store the middle temporary computation outputs and the ultimate results. Through research, such a matrix component is expected to be included in *iZENELib*:

- A general matrix that has both memory and file version.

  Memory version of matrix has been provided by *Boost*, and file version has been implemented by *Clustering-framework*. Therefore it is necessary to provide the general matrix with an improvement on them.

- SVD Decomposition, QR Decomposition, LU Decomposition, Eigenvalue Decomposition.

  These utilities are extremly useful in machine learning and information retrieval.

- Hessian matrix

  Hessian matrix is useful in Bayesian inference and decision theory.

During the developing process of *iZENELib*, more components of matrix would perhaps be included if necessary.

5   Components of IR-Lib

IR-Lib is a collection of algorithms in machine learning and information retrieval, together with the Corpus Management component of AM-Lib, it could

4

provide a generic framework for search applications. Both machine learning and information retrieval have covered lots of fields, therefore, the main purpose of IR-Lib is to provide a scalable framework together with general algorithms, then in future, more advanced algorithms could be added easily.

The relationship between machine learning and information retrieval is very close and machine learning could be seen as the lower layer to provide methods for information retrieval's usage. Therefore IR-Lib could be composed of two layers. In addition, there exists some fields in information retrieval that has not adopted methods provided by machine learning, such as recommendation systems, preprocessing, etc. We will talk about all the components of IR-Lib one by one.

6 Machine learning Components

- Supervised Learning.
  Wisenut-classifier has implemented most of the basic supervised learning methods. We plan to replace the interface to *SML* with the interface to new corpus management component of AM-Lib, and then refactor it to the generic design.

- Unsupervised Learning.
  Clustering-framework has already doen a good job of it. Therefore, the relavant job of this component is to make *Clustering-framework* suit for the whole framework of IR-Lib.

- Learning Complex Models.

    1. EM(Expectation—Maximization), which is also a basic learning approach in semi-supervised learning.
    2. Hidden Markov Models.
    3. Sampling method, including MCMC.
    4. Graphical Models, graphical models including following directions, each of which is under hot research, we are not sure whether it is possible to implement all of them, just try to do that.
        (a) Bayesian Network.
        (b) Markov Random Fields.
        (c) Conditional Random Fields.

- Dimensionality Reduction. We plan to implement PCA at first, more approaches could be done in future if possible.

In summary, machine learning are still under fast developing process, therefore only some basic directions would be included into this library, we hope a good design framework could be provided in order that more learning approaches could be included into this library easily in future.

7 Information Retrieval Components

- Text Pre-Processing
  Text pre-processing techniques are mature and have been implemented by existing projects, therefore we can refactor them from existing code.

  1. Stopword Removal
  2. Stemming
  3. TF-IDF
  4. Tokenization
  5. Feature Selection
  6. Duplicate Detection

- Language Models
  It is necessary to refactor and integrate Jinglei's work into the library.

- Topic Modeling
  Topic modelling is a hot research direction and lots of new approaches appear continuously. We only plan to provide some topic modelling methods including LSI, LDA and 4-level PAM. We hope more topic modeling approaches could be easily added to this library.

8 Access Methods Library Design

8.1 Summery of exiting library

As Figure 3 shows, YLIB is a great work which includes 5 parts as algorithm, container, database, data processor and network. And SML focuses on how to deal with corpus and categorize. SF1Lib is a new library focuses on manipulating database. Our work is giving these library a new face using generic programming. For so many modules and classes in these library, just a few of them relates with the concept access methods. And we conclude the basic functions of AM is QUID (querying, updating, insertion and deletion). Policy-based design in book 'Modern C++ Design: Generic Programming and Design Patterns Applied' gives us a very good clue to do this work.

8.2 Policy-based AM

Policy-based design treats every changeable parts as a policy and integerates all the policies in a host which is treated as an interface for application developers and coordinates all the policies to give the functions. What a user of policy-based designed library need to know is what kind of policy he wants to use and where's the host class. Thus, what are the changeable parts of AM-Lib? They are the place to store the data, main memory or disk, if data stored on disk, the type of cache used, and data type, key type, and the container with different algorithms. We can divide access methods and data structure into two parts. One is some kind of easy data structure without specific algorithms like vector, list etc. The other is some complex structure like tree and table with some certain algorithm. The design of these two parts should be different but their

interface to the client should be the same. The simple data structure could be like follow.

```
//Library code
template
<
        class T,
        class StorageStrategy = MemStorage//if data size is very large,
                                          //we need to change it into DiskStorage
>
class Array;
/////////////////////////////////////////
template
<
        class T,
        class StorageStrategy = MemStorage
>
class List;
/////////////////////////////////////////
. . .
///////////The main class for these simple data structure//////////
template
<
        class T,
        template<class, class> class ConcreteDS,
        class SearchStrategy = BinarySearch,
        class SortStrategy = QuickSort,
        class StorageStrategy = MemStorage
>
class SimpleAccessMethod : public ConcreteDS<T, StorageStrategy>;
```

Some kind of complex DSs (data structures) are different. The data manipulated by these DSs are composited of key and data. They have their own algorithms for sorting and searching. And for some complex situations, the key could be multiple.

```
//Library code
template
<
        class KeyType,
        class DataType,
        class StorageStrategy = MemStorage//if data size is very large,
                                          //we need to change it into DiskStorage
>
class BPTree;
/////////////////////////////////////////
template
<
        class KeyType,
        class DataType,
        class StorageStrategy = MemStorage
>
class HashTable;
/////////////////////////////////////////
. . .
///////////The main class for these complex data structure//////////
template
<
        template<class> class KeyTypeList,
        class DataType,
        template<class, class, class> class ConcreteDS,
        class StorageStrategy = MemStorage
>
class ComplexAccessMethod : public ConcreteDS<KeyTypeList<ConcreteDS>, StorageStrategy>;
```

The final step is to combine SimpleAccessMethod and ComplexAccessMethod together to have a generic face for client to call. To be continued ...

9 Schedule

Kevin and I will take main charge of implemeting *iZENELib*, however, since both of us have other projects to maintain and therefore we could not guarantee the certian time to finish this project. We hope to finish it before Spring Festival at the end of Jan,2009.

MileStone	Finish Date	In Charge	Description
AM-Lib	2008-12-15	Kevin	AM-Lib has been finished
Basic IR-Lib	2008-12-15	Yingfeng	Basic IR-Lib could be provided, including corpus management, basic supervised and unsupervised learning.
Pre-Processing	2008-12-31	Kevin	Text pre-processing component is encapsulated.
Complex machine learning	2009-01-15	Yingfeng	Learning methods for complex models have been finished.
Language models and topic models	2009-01-31	Yingfeng	Finish IR-Lib.

Figure 3: Summery of exiting library