Project Report
for
Implementation of data compression in Simputer DBMS

Amar Agrawal 04329017

Jatin Bharadia 04329R02

October 24, 2004

Acknowledgement

We would like to thank Prof. Krithi Ramamritham for providing us the opportunity to work on the DELite project. We would like to thank him for his invaluable time and guidance. We would like to thank the TAs Aruna Adil and Aswini Rao for their feedback and support.

Abstract

Compression can be used in Database Management system to reduce the space used for storage as well faster query processing. It also reduces the network bandwidth required for transmitting the relation data over some communication channel. Handheld DBMS have different requirements and constraints from Disk based DBMS. These factors should be considered while designing a DBMS for handheld devices, like a Simputer. The Simputer has tighter memory and processing constraints that conventional disk based DBMS. These constraints make compression in Simputer DBMS more lucrative than disk based DBMS.In this project we explored the possibiliy of implementing compression for a handheld DBMS. We also implemented one such light weight compression method for the DELite project.

Contents

1 Introduction **4**

2 Design **5**
- 2.1 System Architecture . . . 5
- 2.2 Functional Design . . . 5
- 2.3 Data Design . . . 5

3 Implementation **6**
- 3.1 Data Structures . . . 6
  - 3.1.1 Relation Structure . . . 6
- 3.2 Compression Algorithm . . . 7
  - 3.2.1 NULL Elimination . . . 7
  - 3.2.2 LZ77 Algorithm . . . 7

4 Module API **8**
- 4.1 Directory Structure . . . 8
- 4.2 Module files . . . 8
- 4.3 Functions . . . 9
  - 4.3.1 Compress Relation . . . 9
  - 4.3.2 Decompress Relation . . . 9
  - 4.3.3 Insert Values into Compressed Relation . . . 9
  - 4.3.4 Display a Compressed Relation . . . 10
  - 4.3.5 Delete tuple from Compressed Relation . . . 10

5 Integration with DELite **11**
- 5.1 Makefiles . . . 11
- 5.2 Compilation and Execution . . . 11

6 Test and Results **12**
- 6.1 STUDENT . . . 12
- 6.2 AQUA . . . 13

7 Conclusion and Future work **15**
- 7.1 Conclusion . . . 15
- 7.2 Future Work . . . 15

Chapter 1

Introduction

Compression can be used in Database Management system to reduce the space used for storage, faster query processing and reduced bandwidth utilization. These benefits are a tradeoff against additional processing and main moemory utilization for compression/decompression process. We have tried to implement a light weight compression method for Simputer DBMS. As the processing power of a simputer is limited the compression process should be less demanding. Hence sophisticated compression techniques which are computationaly intensive could not be used.

The compression module tries to reduce the redundency in the stored data in order to achieve reasonable compression while at the same time being light weight with a small memory footprint. The compression module does compression at two levels in order to achieve better compression ratio.

We have also tried to integrate several DBMS methods directly on the compressed database thereby eliminating the need for decompressing the entire relation at the time of such operations. For example, Insert, Delete and Show can work directly on the compressed relations.

Chapter 2

Design

2.1 System Architecture

The compression functions are a part of a saparate module. This module is integrated with the project. A suitable API for the module is porvided that can be used by the system to interact with the module. The system assumes the underlying file storage which is carried forward by the compression module.

2.2 Functional Design

The module provides a set of functions that form the compression module which can be used to compress the database. Further operation on the compressed database can also be performed by invoking additional procedures implemented as a part of the compression module. The system provides functions for compression/decompressing relations. It also provides functions as addition, deletion and display for the compressed relation.

2.3 Data Design

The earlier DELite system used a fixed length record format for storace of flat relations. In case of fixed length records the system assumes the worst case length of an attribute. Hence lot of storage space is wasted in form of null characters. Variable length records can be used in order to save space by eliminating null characters. The compression

Figure 2.1: Relation with variable length record

module uses the concept of variable length record in order to reduce storage space wastage. The variable length records require the storage of record offsets as a part of relation metadata. The figure 2.1 shows a way to represant variable length records in a relation.

Chapter 3

Implementation

3.1 Data Structures

The logical structure mentioned in the previous chapter is implemented using two files.

- The Meta File - It stores the record offsets and compressed size for each tuple.
- The Data File - It stores the compressed tuples in the relation.

3.1.1 Relation Structure

The Data file is stored as a sequence of variable length records which can be accessed using the data in the meta file.

The structure of the Metafile is shown in figure 3.1. The first two bytes of the metafile are size(in bytes) of the record offset and record size. The later portion of the metafile represent a sequence of record-offset and record-size pairs for each tuple in the relation. The end of metafile is marked by the offset to the end of file.

Figure 3.1: Metafile Structure

The size of relation is not fixed. It can grow and shrink dynamically. So the representation of record offsets and record size using fixed size elements is not a good idea. Using smaller size offset limits the relation size while using larger byte offsets wastes space. This problem is solved by using variable size for record offset and record length. This is implemented by storing the size(in bytes) of record offsets and record size at the start of metafile. There are two advantages of this scheme.

1. The record size is not bounded as the offset and record length size can grow and shrink dynamically.
2. The space is not wasted by using more bytes to store offset and record length.

3.2 Compression Algorithm

The compression modules perfoms compression at two levels.

1. All the null characters are eliminated from the fixed length records.
2. The record is compressed using LZ77, a dictionary based compression algorithm.

3.2.1 NULL Elimination

The elimination of NULL characters in the record before the tuple compression improves the compression ratio. Otherwise the LZ77 compression algorithm also encodes the NULL characters considering them to be useful data which reduces the compression gains.

3.2.2 LZ77 Algorithm

The principle of the LZ77 compression algorithm is to store repeated occurrences of strings as references to previous occurrences of the same string. The point is that the reference consumes less space than the string itself, provided that the string is long enough.

The coder uses a brute force approach to finding string matches in the history buffer.The decompression is very fast, and the compression ratio is often very good.

The reference to a string is coded as a (length,offset) pair, where the length indicates the length of the string, and the offset gives the offset from the current data position. To distinguish between string references and literal strings (uncompressed bytes), a string reference is preceded by a marker byte, which is chosen as the least common byte symbol in the input data stream (this marker byte is stored in the output stream as the first byte).

Occurrences of the marker byte in the stream are encoded as the marker byte followed by a zero byte, which means that occurrences of the marker byte have to be coded with two bytes. The lengths and offsets are coded in a variable length fashion, allowing values of any magnitude (up to 4294967295 in this implementation).

Chapter 4

Module API

4.1 Directory Structure

Figure 4.1 shows the directory structure of the DELite project and the compression module files in it.

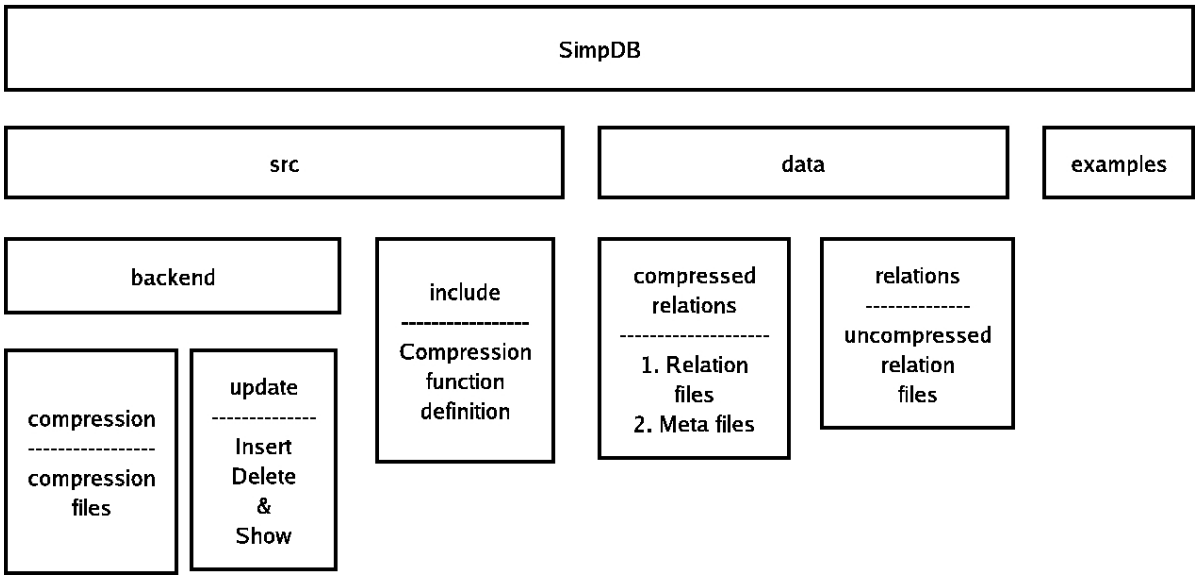

Figure 4.1: Directory Structure

4.2 Module files

The compression module is implemented in the following files

1. commonfunction.c - The file contains common functions used by the compression functions.

2. commonfunction.h - The header file for the common functions.

3. compress.c - The file implements functions to compress relations and to insert new tuples into the relation.

4. compress.h - The header file for the compress functions.

5. decompress.c - The file implements functions to decompress relations and to fetch compressed tuples from the relation.

6. lz.c - Implements the LZ77 compression algorithm.

7. lz.h - The header file for the LZ77 compresss function.

8. Makefile - The make file used to compile the module invoked by the global make file.

4.3 Functions

4.3.1 Compress Relation

Syntax	`compress_table(char *tablename)`
Description	The function is invoked by the parser to compress a relation in the database.
Input	The name of the relation to be compressed.
Returns	Nothing

4.3.2 Decompress Relation

Syntax	`int decompress_table(char *tablename)`
Description	The function is invoked by the parser to decompress a relation in the database.
Input	The name of the relation to be decompressed.
Returns	Nothing

4.3.3 Insert Values into Compressed Relation

Syntax	`void InsertCompressedValues(CATSTRUCT(simp_relations) relstruct, struct AttrValParse *attrvalinfo )`
Description	The function is invoked by the parser to insert a tuple in compressed relation.
Input	Structure containing the Relation Catalog data and tuple value to be inserted.
Returns	Nothing

4.3.4 Display a Compressed Relation

Syntax	`void jireadComptuples(CATSTRUCT(simp_relations) relstruct)`
Description	The function is invoked by parser to show a compressed relation.
Input	Structure containing the Relation Catalog data.
Returns	Nothing

4.3.5 Delete tuple from Compressed Relation

Syntax	`unsigned int DeleteCompressedTuples(`
	`CATSTRUCT(simp_relations) relstruct,`
	` struct AttrIdPredValParse *aipv)`

Description	The function is invoked by parser to delete tuples from a compressed relation.
Input	Structure containing the Relation Catalog data and list of attributes to match.
Returns	Nothing

Chapter 5

Integration with DELite

5.1 Makefiles

The module is integrated with the entire package by including the entry of the module's makefile in the main makefile which calls the makefiles of subsequent modules at lower level. These makefiles compiles and make the executables for each module which are inturn linked together to form the final executable of the DELite project.

5.2 Compilation and Execution

The compilation of the entire code is quiet simple by executing make in the `simpdb` directory. This creates two executable files in the `simpdb/bin` directory

1. `initdb` - This file initializes the database files.
2. `sqlplus` - This file is used to execute sql statements(in a file) on the database.

The following sequence of statements should be executed..

```
simpdb> make
simpdb> cd bin
simpdb/bin> ./initdb
simpdb/bin> ./sqlplus ../example/example.sql
```

Chapter 6

Test and Results

The following sections presents two test scenarios for the project. Files related to both the test cases are in the `example/compression/<case_name>` directory.

6.1 STUDENT

Create file

```
CREATE TABLE STUDENT (ROLL_NO INTEGER, NAME CHAR(50), ADDRESS CHAR(150),
                                                BIRTHDATE CHAR(15), CPI FLOAT);
```

Insert file

```
INSERT INTO STUDENT VALUES (101,‘Bharadia Jatin Arvindkumar’,
                ‘10C Maninagar Ahmedabad Gujrat 380008’,‘18th Apr 1982’,9.1);
INSERT INTO STUDENT VALUES (102,‘Agrawal Amar Pyarelal’,
         ‘A29 Asara Housing Society Hotgi Road Solapur’,‘20th July 1982’,8.5);
INSERT INTO STUDENT VALUES (103,‘Nadahali Tejaswi’,
             ‘24 Yahoo Engineer Street Bangalore 2417316’,‘8th May 1978’,9.5);
```

Compress file

```
COMPRESS TABLE STUDENT;

Original File Size = 669

Compressed File Size = 276

Compression = 58.74 %
```

Show

```
SHOW TABLE STUDENT;
```

Delete

```
DELETE FROM STUDENT WHERE ROLL_NO=102;
```

Decompress

```
DECOMPRESS TABLE STUDENT;

Time for decompression = 0.000000 seconds.
```

6.2 AQUA

Create file

```
CREATE TABLE AQUA (MSGID INTEGER, AUTHOR CHAR(50), TEXT CHAR(400), DATE CHAR(10));
```

Insert file

```
INSERT INTO AQUA VALUES (101,‘Rajiv Ghosh’,
                            ‘I have 20Kg tomatoes for sale Any buyers’,‘20th MAY’);
INSERT INTO AQUA VALUES (102,‘Ketan Kalgaonkar’,
     ‘Does anyone know where I can get good quality seeds for my farm’,‘21st MAY’);
```

Compress file

```
COMPRESS TABLE AQUA;

Original File Size = 928

Compressed File Size = 174

Compression = 81.25 %
```

Show

```
SHOW TABLE AQUA;
```

Delete

```
DELETE FROM AQUA WHERE MSG_ID=101;
```

Decompress

```
DECOMPRESS TABLE AQUA;

Time for decompression = 0.000000 seconds.
```

Chapter 7

Conclusion and Future work

7.1 Conclusion

We have tried to implement a light weight compression module which is suitable for a constrained system such as a Simputer. The project studies the feasibility of implementing such an idean on DELite DBMS. The results obtained so far are promising. The compression ratio obtained were encouranging and the added support for variable recorts opened new avenues for a efficientstorage mechanism. The project demonstrates that various database operations can be made compression aware to increase performance as there is no need to decompress the entire relation before such operations could be performed.

7.2 Future Work

The query evaluation engine was not fully functional at the time of the project. One of the possible enhancements could be to implement compression aware query evaluation using the existing framework.

The compression algorithm has a clean interface with the module so that a better compression algorithm could be easily integrated with the system.

A granularity of the system could be made finer by applying comression algorithms on attributes instead on records. The Attribute level compression was avoided in order to increase compression gains.

Bibliography

[1] "Memory Constrained DBMS with Updates", Ashwini G. Rao, M. Tech. Project report, IITB.

[2] "An open source DBMS for handheld devices", Rajkumar Sen, M. Tech. Project report, IITB.

[3] "Data Compression: A Performance Enhancement Tool", Gautam Ray, Jayant R. Haritsa and S. Seshadri, COMAD, 1995.

[4] "The Implementation and Performance of Compressed Databases", Till Westmann, Donald Kossmann, Sven Helmer and Guido Moerkotte, SIGMOD, 1998.

[5] "Database System Concepts", Silberschatz, Korth and Sudarshan, McGraw-Hill International Fourth Edition, 2002.