CSE 510 Project Phase 2

Sanchit Aggarwal
saggar26@asu.edu

Zhenyu Bao
zbao6@asu.edu

Matthew Behrendt
mbehren4@asu.edu

Ziming Dong
zdong27@asu.edu

Alexandra Szilagy
aszilagy@asu.edu

Hang Zhao
hzhao64@asu.edu

ABSTRACT

This project phase entailed the implementation of a Bigtable-like DBMS using the packages, classes, and functions provided in the Minibase code, which previously supported a relational, tuple-based system. This implementation modified and expanded upon many of the existing features provided by Minibase and added new classes for defining the new structures required to support a Bigtable system. The system has initially been created to support two operations: one for inserting sets of data into the database at once and another for querying the database and retrieving data conforming to certain input parameters.

KEYWORDS

Bigtable, Maps, B+ Tree Indexing, DBMS Implementation

1 INTRODUCTION

After familiarizing ourselves with Minibase's functionality in the first phase of the project, our team was then required to work with the packages and classes provided in the codebase to replace the current relational system with a new system based on the Bigtable DBMS. In addition to modeling the newly-created data structures after those in Bigtable systems, this system also needed to support several different types of indexing based on the fields in the map structures used to store the data. With this new structure in place, operations for inserting large collections of data into the database and retrieving subsets of this data based on user-specified parameters then needed to be developed. These operations will also calculate and return the number of disk page reads and writes performed over the course of the operation's execution. The rest of this paper describes our implementation of this DBMS and the results from performing the above operations using this new system on a provided set of test data.

1.1 Terminology

Bigtable is a distributed system for storing and managing data using a map data structure in place of the relational structure commonly used in traditional systems. As described in [1], these maps consist of sets of three values: a row label, a column label, and an integer timestamp, which serve as a key that is mapped to a corresponding string value. The set of maps is maintained such that only three maps with the same row and column labels are kept, and when a fourth map with the same labels is added, the corresponding map with the oldest timestamp is replaced. This style of DBMS offers greater scalability than traditional relational systems.

Minibase and, by extension, our expansion of the Minibase code to implement Bigtable, uses a B+ Tree structure for indexing data. This index is a tree-based structure not unlike a binary search tree, with a root node that branches out to multiple child nodes which may branch out to child nodes of their own until the last level of the tree containing the leaf nodes. The index contains nodes consisting of a key, which can be either an integer or a string value, which is paired with a pointer to the corresponding data. These key-pointer pairs are stored in sorted order based on the key in the leaf nodes of the B+ Tree. Data can be indexed based on certain field values, and this indexing strategy can be used to greatly improve the DBMS' performance, particularly in searching for data or scanning through a range of values.

1.2 Goal Description

The primary goal for the second phase of this project was to utilize the provided Minibase code for the implementation of a BigTable-style DBMS. This was accomplished through the creation of a package, BigT, containing the new classes that would be needed to implement this DBMS as well as the modification of many of the existing Minibase classes to replace the previously supported tuple-based system with a system based on the map structures that we had developed.

The BigT package contains three classes which are needed to create the underlying structure for this new system. The first is a map class that defines the structure for storing data in accordance with the map structure defined in [1]. This structure and the additional methods that were implemented in the map class are described in more detail in 2.1. The bigT class is responsible for creating the BigTable and its corresponding files for storing and indexing the data. Its functionality includes the insertion of maps into the table and the retrieval of meta-information about the data stored in the table, and these methods are further described in 2.2. The stream class, whose implementation is detailed in 2.3, contains the necessary methods for scanning through the data in the table to retrieve maps. A menu class was also created in this package to provide an interface for the batch insert and query operations that needed to be implemented. This class receives formatted commands detailing parameters for the data to be inserted or retrieved, parses the input, and performs the requested operation. The implementation of these operations is described in 2.9 and 2.10.

The main packages that needed to be modified to incorporate this new map-based structure in place of the tuple-based structure were the btree and heap packages, which are responsible for maintaining the index and data files in the database, respectively. Several classes in these packages had to be modified to incorporate the map data structure and the newly-created MID class, which replaced the RID class that had previously been used as a pointer to record tuples, and the details of these modifications are provided in 2.5 and 2.6.

In addition to the modification of Minibase classes to support this new structure, some classes had to be replaced with parallel classes that provide essentially identical functionality that works with the new BigT classes. Minibase's TupleUtils class, containing utility functions for comparing and performing other operations on the data stored in tuple structures, was used to create an analogous MapUtils class to provide the same functions for map structures. Similarly, the DB class in the diskmgr package, responsible for creating and maintaining a database of relational tables, was replaced with a class called bigDB which would do the same for a BigTable database. The implementation of these two classes is detailed in 2.4 and 2.7.

1.3 Assumptions

Our implementation of the map data structure assumes that any data which will be inserted into the table will be structured in the same way as the *test.csv* file provided and that none of the field values in any new data provided will have a length exceeding that of the largest value in the corresponding field of this file. This was a necessary assumption for allowing us to determine the values to use for our fixed-length map fields based on the provided test data. No other critical assumptions were made in our implementation of this project phase.

2 IMPLEMENTATION DETAILS

In this section we describe all the additions and modifications that were made on the minibase to accommodate the new map construct and facilitate the storage and retrieval of maps from a new BigTable-style database. We created a new package BigT which contains all the main classes that handle the big table constructs.

2.1 Map Class Implementation

The Map class is created to extend the Minibase to support the new map construct which is analogous to tuples, but unlike the tuple, the map has a fixed structure with four attributes, rowLebals, columnLebals, TimeStamp and Value. We have defined a fixed-length for each attribute to keep a fixed length record format and followed an unpacked page organization where a bitmap directory will keep track of the occupied and free slots. The 4 fixed fields in map along with their type and size are:

Field	Type	Bytes
Row Label	String	30
Column Label	String	20
Timestamp	Integer	10
Value	String	10

Table 1: Map attribute names, type and byte size

In this class, we have lots of functions. The size() function will return the size of the map. Some set() functions that can set each label from String type to Bytes, which can obtain raw data and convert to MAP format. Some get() functions that can return some Labels that we need in the database. This class contains a default constructor, parameterized constructor and a copy constructor for initializing a map and for copy functionalities. The mapInit() and the mapSet() functions initialize and copy respectively using a byte array and an offset as parameters instead of using the constructors. And we can use print() function to print one MAP in the database.

2.2 BigT Class Implementation

The BigT class handles the creation and maintenance of all the relevant heap files and indexes to organize the data in the BigTable. The bigt constructor initializes a big table along with the type as a parameter to determine which indexing strategy will be used for that table. The getMapCnt, getRowCnt and getColumnCnt functions are used to return the number of maps, number of distinct rows and number of distinct columns respectively in the big table. This is done using the heapfile getRecCnt function. A separate Heapfile is used to store distinct RowLabel and distinct ColumnLabels. The Heapfile getRecCnt is again used to get the number of records from each of these files to return the distinct number of row labels and column labels. Heapfiles containing the data maps, the distinct rows, and the distinct columns which are maintained during insertion, and calling this function on the respective heapfile returns this value. The deleteBigt function implements the deletion of a big table from the database by destroying its disk pages and removing the random access file used.

Maps in the big table are inserted using the insertMap function, which returns an MID for the map upon successful insertion. The insert function checks first if there are already maps with the same row and column labels existing in the table using the search function in the bigT class which returns a list of maps with matching labels. This search function uses the Row + Column combined index for fast insertion. If there are already three maps with the same labels, the map with the oldest timestamp is deleted using the deleteRecord function. The insertIndex and deleteIndex methods handle the updates on index files of the corresponding bigtable based on its type, whenever a record is inserted or deleted. To handle queries with filters on row, column and value, openStream function is defined to fetch a stream of maps matching the filters in the query and to order the results based on the input order type.

2.3 Stream Class Implementation

The Stream class contains five functions. The first function is to initialize a stream of maps on bigtable, retrieving maps with filters on rows, columns and values. We write our five order types in this class instead of bigT. We have the sort method here by creating comparators classes for RowLabel, ColumnLabel and TimeStamp. Then we can sorted the filtered maps along with these comparators.
For example, if the ordertype =1, then
Collections.sort(query_map, new
RowLabelSorter()
.thenComparing(new ColumnLabelSorter())
.thenComparing(new TimeStampSorter()));
The query_map is the array list of all filtered maps The second function is to get the result of filtered maps and save them into a list called MAP. This third function is called filter compare, we can handle three cases, unspecificied value, sing value and range. The fourth function is to close the stream object. Same as scan.java in heap folder. We use the same way to close and reset the stream object which means unpin all pages. The fifth function is to get the next map, it will go over the query_map and print the next map.

2.4 MapUtils Class Implementation

The MapUtils class was created to replace Minibase's TupleUtils class, providing the same functions in support of the newly-created map data structure. The class contains five functions: CompareMapWithMap, Equal, compareRowCol, Value, and SetValue. CompareMapWithMap takes two map objects and an integer value representing the field number and compares the maps based on this field. The field number values which were used are: 0 for the Row Label, 1 for the Column Label, 2 for the Timestamp, and 3 for the Value. The function retrieves the field value for both map objects using the corresponding get function and returns 0 if they are equal in the given field, 1 if the first map's field value is greater than the second map's, and -1 if the first map's field value is less than that of the second map. The second function, Equal, receives two maps as input arguments and compares them across all four fields, returning $true$ if the two maps are equal in all fields and $false$ otherwise. The next function, compareRowCol, is unique to MapUtils and was created to support the enforcement of the DBMS' restriction on the number of maps with the same row and column labels. This function receives two maps as input and compares the row labels and the column labels of the two maps, returning $true$ only if both of the fields are equal. Value simply takes a map object and an integer for the field number as specified above in 5 and returns the value for the map in the given field. The final function, SetValue, takes two maps and a field number as input. The value from the second map for the corresponding field is retrieved, and the value for the first map in this field is set to this value.

2.5 Modification of btree Classes

Numerous classes in the btree package needed to be modified to support the BigTable implementation. This process primarily entailed replacing RID objects with MID objects in the btree functions. The BT class required modifications to the functions getEntryFromBytes, _printPage, _printTree, and _printAllLeafPages to incorporate MID where RID had previously been used. Similar modifications were made to the class BTFileScan, which is responsible for scanning through the entries in a given index file. The class variable curRid was replaced with a corresponding MID variable, curMid, and the get_next and delete_current functions were modified to use this new variable. The next class to be modified was BTIndexPage, which is responsible for maintaining the collection of key, PageId pairs. This class required modifications to the insertKey, deleteKey, adjustKey, and redistribute functions to accommodate MIDs where RIDs had previously been used. The functions getFirst and getNext were modified in a similar way, and they also required the changing of their return types from RID to MID.

The BTLeafPage class, which has a similar function but is instead responsible for the key, MID pairs, received essentially the same set of modifications. BTSortedPage, which holds abstract records sorted based on their key values, had similar changes made to the functions insertRecord and deleteSortedRecord. This class extends the HFPage class from the heap package and uses the insertion and deletion functions from its superclass, so these functions had to be adapted accordingly as well. The BTreeFile class required the most intensive set of modifications; numerous functions, including _destroyFile, insert, and Delete, used RID in multiple places, and each of these references had to be changed to incorporate MID instead. Finally, the KeyDataEntry and LeafData classes were also modified to use MID in place of RID.

2.6 Modification of heap Classes

We decided to utilize the classes in the heap package for storing our data rather than creating our own from the ground up, which required the modification of these classes to support maps and MIDs in place of tuples and RIDs. The replacement of the references to RID with MID was trivial and done in essentially the same manner as described above in 2.5. The replacement of tuples with maps, however, required a slightly more nuanced approach than simply finding and replacing all references to tuples with maps. We decided to keep tuples for the heap directory pages, as this seemed to be a much more fitting structure than the maps for storing the necessary directory information. As a result, all of the tuples used in the directory page were kept the same. These changes were first introduced to the Heapfile class, and the resulting errors allowed us to trace all of the functions that needed to be modified in other classes.

Our approach also required the implementation of functions for returning maps and tuples for a given MID since RID had been replaced but the same could not be done for all tuples. In the HFPage class, we created functions for inserting, deleting, and retrieving both records and maps based on MIDs. The Scan class, which allows for the scanning of heapfiles, was modified similarly, with the replacement of RID with MID and the implementation of functions such as getNext for returning map objects rather than tuples.

2.7 BigDB Class Implementation

The bigDB class in the diskmgr package was created to serve as an analogous class to DB in the same package, creating and maintaining a database of Bigtables instead of the relational tables used by Minibase. Many aspects of the two classes are essentially identical, but there are some key differences. The first is that this class uses the pcounter class described in 2.8 to track the number of reads and writes, incrementing the read count in the read_page function and the write count in write_page. Next is the change between createDB and openDB. createDB will create a new random access file that stores only binary to maintain the heap and index files. The number of pages for this file is fixed at creation thus a large page number (10000) is used to ensure enough pages when running the batch insert process. The openDB will simple open the random access file and read the database header page to set the number of pages this database has.

2.8 Disk Manager Read/Write Counting

At the end of Batch Insert and Query, the program outputs the number of how many times the system read a disk page and how many times the system wrote to a disk page. Counting the number of reads and writes of the disk pages required the creation of a class called pcounter. This class is simply used to increment the number of reads and writes, which could also reset the values to 0 through a function called initialize. Because pcounter's functions are all static, it is possible to call them without creating a pcounter object. It is beneficial that object creation is not necessary because then the object does not need to be passed to the Disk Manager as a parameter. Inside the read_page function, the readInc function is called directly after the call to read the buffer. The readInc function increments the count of page reads. Inside the write_page function is the same as the read_page function, except of course, writeInc is called to increment the count of page writes. Both batchInsert and query call initialize before they process any data so that the functions can output the number of reads and writes processed during the entire duration of the functions.

2.9 Batch Insert Implementation

For the Batch Insert program, we can use this program to store all maps which in the bigtable. Based on the test file which is given for this project. In the csv file: the first column will be the row label, the second column will be the column label, the third column will be the timestamp value, the last column which in the test file will be the each map's value. At this point, we can insert each row to bigtable as a map.

When we call the batch insert function, we can provide the bigtable name and type to get a bigtable type, we have five different indexing strategies to use for database so we have five different types of bigtable. We write the code to see if database already on the list, otherwise it will create new a database type with its name. We also need to input the number of buffer pages for database, it tells batch insert program that max buffer pages to access to the bigtable and return maps. The last thing is to input the filename, it tells which file is used to do the batch insert. Moreover, we want to make sure that there is no mistake to count the number of disk pages that were read and written.

The class BufMgr defines how pinpage, unpinpage and replacement policy works, for our program, we use the clock replacement policy to replace buffer page when it is full. Also, when program check if the bigtable already exists in the database, it will try to pin pages for this database, else we will create the database and unpin pages, then flush all pages in the new database. We know we need to unpin pages before flush all pages, because flush all pages will write all the dirty pages to the disk, but if there are any pinned pages, it throws the exception. So we want to make sure you are calling flush all pages only after unpin all the pages. We also have a hard time to test how many initial pages we need to store all the records from the input file, we want to make sure that our program can store any size of file, thus we decided to let page size grow dynamically to make insertion efficient. To test the batch insert program, we can use command like:

batchinsert {$Path-to-src$}/src/BigT/test.csv 1 dbtest
The Simple result will be:

```
Starting Batch Insert
No Database on the list
DBname:dbtest1
Replacer: Clock

Dominica,Zebra,46067,1
inserting map to page x slot x
```

2.10 Query Implementation

A basic overview of the query function process is to first, retrieve the database that will be filtered. Second, filter the maps in the database to retrieve the ones that satisfy the specified filters. The maps are first filtered by an index depending on what index type is used, then the rest of the labels are compared to determine if all of the map's labels satisfy the corresponding filters. Third, the remaining maps are sorted to satisfy the order type that was requested by the user. Lastly, the query function prints the maps as output to the user.

The query program will access the database and return the matching maps as an array list. The user can input the: database name, order type, filters, and number of buffer pages. The order type is the same order type mentioned in the bigT table implementation section of the paper. The critical part of the query is to recognize the filter label, there is the row filter, column filter and value filter. The query function handles three different types of filters: a range, an equality, and an unspecified value that returns all labels of the label type. If the input filter type is a single value, query will return all maps that match the filter ordered by requested order. If the input filter type is range, then the query function will print all maps that fall in between and all maps that equal the beginning or the end of the range. Consider the following input:

BIGTABLENAME 1 1 [Singapore,Switzerland] Camel * 50

A possible output of this query would be:

```
[Singapore, Camel, 47254, 3864]
[Singapore, Camel, 93436, 21588]
[Singapore, Camel, 76949, 23607]
[Slovakia, Camel, 79003, 39228]
[Slovakia, Camel, 67146, 46989]
[Slovakia, Camel, 34886, 47719]
```

The NUMBUF parameter is sent to the buffer so that it knows the max number of Buffer pages that are allowed for query processing. The program should also output the number of disk pages that were read and written separately. This requirement was implemented by calling the pcounter class from the diskmgr package.

The query function handles many different filters, filter types, and order types by calling other functions and reusing them. Query calls filterComp, compares the type of filter with the corresponding label, and returns true if the label satisfies the query. The function does this by checking if the label is within the range or equal to the filter. It is possible to reuse filterComp for all three labels, row, column, and value, because the function takes in a parameter called labelType that tells the function what label to retrieve from the map.

The query function is called by the Stream class. The Stream class takes the resulted array list and sorts it according to the order type. The Stream function sorts the list by each label separately. For example, if the user input an order type of 1, the row label of every map in the list is compared and then the column labels are compared among the row labels that equal each other and finally the timestamps are compared if both the row and column labels are the same.

3 INTERFACE SPECIFICATIONS

Return Type	Signature
void	bigT(String name, int type)
void	deleteBigt()
void	closeBigt()
int	getMapCnt()
int	getRowCnt()
int	getColumnCnt()
void	deleteRecord(MID mid)
MAP	getMap(MID mid)
MID	insertMap(byte[] mapPtr)
Map<MID,MAP>	search(byte[] mapPtr)
void	insertIndex(KeyClass key, MID mid, String index_name)
void	deleteIndex(KeyClass key, MID mid, String index_name)
ArrayList<MID>	scanForKey(String index_name, KeyClass lo_key, KeyClass hi_key)
Stream	openStream(HashMap<String, bigT> db_list, String name, int type, int ordertype, String rowFilter, String columnFilter, String valueFilter, int numbuf)

Table 2: bigT Methods

Return Type	Signature
void	MAP()
void	MAP(int size)
void	MAP(byte[] amap)
void	MAP(MAP fromMap)
String	getRowLabel()
String	getColumnLabel()
int	getTimeStamp()
String	getValue()
void	setRowLabel(String val)
void	setColumnLabel(String val)
void	setTimeStamp(int val)
void	setValue(String val)
byte[]	getMapByteArray()
byte[]	getMapData()
void	print()
int	size()
void	mapCopy(MAP fromMap)
void	mapInit(byte[] amap)
void	mapSet(byte[] frommap, int offset)
MAP	setIntFld(int fldNo, int val)
MAP	setFloFld(int fldNo, float val)
MAP	setStrFld(int fldNo, String val)

Table 3: MAP Methods

Return Type	Signature
void	Stream(HashMap<String, bigT> db_list, String name, int type, int ordertype, String rowFilter, String columnFilter, String valueFilter, int numbuf)
ArrayList<MAP>	query(bigT db, String name, int type, String rowFilter, String columnFilter, String valueFilter)
boolean	filterComp(int labelType, String filter, MAP bmap) throws IOException
void	closestream()
void	reset()
void	getNext()

Table 4: Stream Methods

Return Type	Signature
int	CompareMapWithMap(MAP m1, MAP m2, int map_fld_no)
boolean	Equal(MAP m1, MAP m2)
boolean	compareRowCol(MAP m1, MAP m2)
String	Value(MAP map, int fldno)
void	SetValue(MAP value, MAP map, int fld_no)

Table 5: MapUtils Methods

4 INSTALLATION AND EXECUTION INSTRUCTIONS

The instructions to run the minibase and test its functionalities are as follows-

(1) Copy the “minjava.tar.gz” file to your account in a Unix/Linux based system.
(2) Uncompress the file using the command- "tar -xvf minjava.tar"
(3) Modify the ’Makefiles’ to reflect your directory structure.
(4) In the "src" directory, build the source by running the command - "make db"
(5) Now to start the program from the "src" directory, use the command - "make menu"
(6) This previous command will print a list of options. Execute any command listed in the options in the given format to achieve desired results.

To exit the program, simply press Ctrl+C or type ’exit’ in the command line. The command specifications for batch insert and queries are as follows-

1. Batch insert:
batchinsert [DataFile] [IndexType] [DB_Name] [NumBuff]

The batch insert function can be invoked from the command line using the above format. The parameters like ’DataFile’ defined the location of the csv data file from which the data has to be extracted and inserted into the database. ’IndexType’ defines the type of index to be generated for that table. There can be 5 types of indexes-

- Type 1: No index
- Type 2: Index on row labels
- Type 3: Index on column labels
- Type 4: Index column label and row label (combined key) and one index on timestamps
- Type 5: Index row label and value (combined key) and one index on timestamps

Use the integer value corresponding to the type needed. The ’DB_Name’is the Big Table Name in which the table has to be inserted. And the ’NumBuff’ placholder is to define the number of buffer pages to be used.

2. Query:
query [DB_Name] [IndexType] [OrderType] [RowFilter] [ColumnFilter] [ValueFilter] [NumBuff]

The parameters in query includes search key filters like, ’RowFilter’, ’ColumnFilter’, ’ValueFilter’ to find maps with equality search or range search based on the values given for the search keys. To skip using any search key, use- ’*’. To specify a range from x to y, use the format- ’[x,y]’. The ’DB_Name’ defines the Big Table name in which the maps have to be searched in. The ’NumBuff’ parameter is used to set the number of buffer pages to be used for the query. The ’IndexType’ defines the type of the index which was initially created for that table during insertion. The ’OrderType’ defines the order in which the results have to be sorted. The order type take the integer value of the corresponding types mentioned below-

- Type 1: Results are first ordered in row label, then column label, then time stamp.
- Type 2: Results are first ordered in column label, then row label, then time stamp.
- Type 3: Results are first ordered in row label, then time stamp.
- Type 4: Results are first ordered in column label, then time stamp.
- Type 5: Results are ordered in time stamp.

5 RELATED WORK

Bigtable DBMS implementation and its possible applications and benefits have been discussed in numerous academic works. For instance, a paper by Keith B. Hall, Scott Gilpin, and Gideon Mann [2] discusses the utility of Bigtable in conjunction with the MapReduce processing algorithm for implementing various methods and strategies for distributed optimization of large volumes of data. Using two large datasets of click-through data, one with hundreds of millions of data points and another with over one billion, the authors demonstrate that Bigtable and MapReduce can implement various types of distributed optimization strategies and achieve high levels of performance with some of them, namely IPM SGD, which had high accuracy and low resource usage for both datasets. Another paper by Wei Wei, Ting Yu, and Rui Xue [3] presents an expansion on the current Bigtable technology with regard to the integrity of Bigtable data stored in a public cloud. This proposed concept, referred to as iBigTable, seeks to improve upon Bigtable's current implementation of distributed authentication data with a new design that "removes the dependency between the authenticated data structures of tablets" and "stores the root hash of each tablet in the data owner" (p. 343). Using the HBase code, they implemented a prototype of this new design and found iBigTable to be a promising implementation of data integrity for a Bigtable-based DBMS without imposing significant overhead on the system.

6 CONCLUSIONS AND EXPLANATION OF TEST RESULTS

There were a few things that the team learned during the development of the Bigtable database. One thing learned was that it is of utmost importance to ensure that any database system is efficient. The program started out with an insertion time of fifteen minutes or more. Such a large insertion time caused the team to waste time during testing and would have surely infuriated any users who had a large amount of data to process. The team made it a priority to reduce it to a more reasonable time and the time was eventually reduced to five minutes using the correct indexing techniques. The team is confident that, at the current build, the system's run time is reasonable for the average user. The main challenge was to maintain at-most 3 copies of a map the with same row and column label which resulted in additional reads to check the previous instances. This time was reduced using an index to search for the inserted records and delete whenever a fourth instance is being inserted.

The results from the queries indicated that the search key terms that were present in the index were retrieved faster compared to searches on non-indexed search keys in the query. We compared the statistics such as disk read/write and process time to determine which index works the best. Some problems we encountered include range search, search without index, and sorting the results. Search without index was the most difficult and time consuming since we have to scan the entire database and compare each map with the filters. Range search was difficult due to the range of strings from the rowLabel filter and columnLabel filter. This is more difficult with an index as we need to know each map's row and column label is within range. Since, it was always an un-clustered index as the data inserted was not sorted on the index keys, the range search results took longer time as the subsequent maps in the range were present on different pages leading to more seek and rotate times. This is less of a problem when using an index since bTree key scan provides the capability of scanning a range of string values.

REFERENCES

[1] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, pp. 1–26, 2008.

[2] K. B. Hall, S. Gilpin, and G. Mann, "Mapreduce/bigtable for distributed optimization," in *Neural Information Processing Systems Workshop on Leaning on Cores, Clusters, and Clouds*, 2010, http://lccc.eecs.berkeley.edu/.

[3] W. Wei, T. Yu, and R. Xue, "Ibigtable: Practical data integrity for bigtable in public cloud," in *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 341–352. [Online]. Available: https://doi.org/10.1145/2435349.2435399

A SPECIFICATION FOR ROLES OF THE GROUP MEMBERS

- Sanchit Aggarwal: BigT functions, Batch Insert, Command line invocation program, Integration and Testing, Report
- Zhenyu Bao: Map, Batch Insert, Query about Filter, Testing and Report
- Matthew Behrendt: BigT, MapUtils, btree, index, Heapfile, Testing, Report
- Ziming Dong: BigT Stream class, BigT three sorters classes, Stream five ordertypes, Testing, Report.
- Alexandra Szilagy: Query, filter compare, Disk Manager Read/Write Counting, Report, Testing
- Hang Zhao: Heapfile, Record Search, BigT functions, Batch Insert, bigDB, Testing, Report, Helped with query and sorting,

B TEST RESULTS

The results of the program are separated into two sub-sections. The first sub-section is the batch insert results, which displays the input, the number of disk page reads and writes, the map count, the distinct row and column count, and the amount of time that the process took to complete. For each test of batch insert, the data set remained the same, so that there was no variation in the number of insertions or deletions. The only things that were changed for each test was the index type and the number of buffer pages. Each index type was tested twice, once for 25 buffer pages, and again for 50 buffer pages.

The second sub-section is the query results, which displays the input, the output, the number of disk page reads and writes, and the amount of time that the process took to complete. Each index type was tested five times, one for each order type. This was to ensure that the output for each query had the same maps as output, and each query had a different order for the output.

B.1 Batch Insert Results

Index Type: 1
Dataset: Full Testdata
NumBuff: 25
batchinsert test_full 1 dbtest 25
Number of disk page reads: 137804136
Number of disk page writes: 315464
MAP Count: 28254
Distinct Row Count: 100
Distinct Column Count: 100
This process took: 368713 milliseconds

Index Type: 1
Dataset: Full Testdata
NumBuff: 50
batchinsert test_full 1 dbtest 50
Number of disk page reads: 137462169
Number of disk page writes: 240899
MAP Count: 28254
Distinct Row Count: 100
Distinct Column Count: 100
This process took: 373635 milliseconds

Index Type: 2
Dataset: Full Testdata
NumBuff: 25
batchinsert test_full 2 dbtest 25
Number of disk page reads: 138053165
Number of disk page writes: 428752
MAP Count: 28254
Distinct Row Count: 100
Distinct Column Count: 100
This process took: 424443 milliseconds

Index Type: 2
Dataset: Full Testdata
NumBuff: 50
batchinsert test_full 2 dbtest 50
Number of disk page reads: 137669137
Number of disk page writes: 343318
MAP Count: 28254
Distinct Row Count: 100
Distinct Column Count: 100
This process took: 423065 milliseconds

Index Type: 3
Dataset: Full Testdata
NumBuff: 25
batchinsert test_full 3 dbtest 25
Number of disk page reads: 138053870
Number of disk page writes: 428526
MAP Count: 28254
Distinct Row Count: 100
Distinct Column Count: 100
This process took: 764984 milliseconds

Index Type: 3
Dataset: Full Testdata
NumBuff: 50
batchinsert test_full 3 dbtest 50
Number of disk page reads: 137670472
Number of disk page writes: 343330
MAP Count: 28254
Distinct Row Count: 100
Distinct Column Count: 100
This process took: 794226 milliseconds

Index Type: 4
Dataset: Full Testdata
NumBuff: 25
batchinsert test_full 4 dbtest 25
Number of disk page reads: 262797951
Number of disk page writes: 589394
MAP Count: 28254
Distinct Row Count: 100
Distinct Column Count: 100
This process took: 2079071 milliseconds

Index Type: 4
Dataset: Full Testdata

```
NumBuff: 50
batchinsert test_full 4 dbtest 50
Number of disk page reads: 137681546
Number of disk page writes: 331144
MAP Count: 28254
Distinct Row Count: 100
Distinct Column Count: 100
This process took: 1397369 milliseconds
```

```
   Index Type: 5
Dataset: Full Testdata
NumBuff: 25
batchinsert test_full 5 dbtest 25
NumBuff: 50
batchinsert test_full 5 dbtest 50
Number of disk page reads: 138297150
Number of disk page writes: 536949
MAP Count: 28254
Distinct Row Count: 100
Distinct Column Count: 100
This process took: 486801 milliseconds
```

```
   Index Type: 5
Dataset: Full Testdata
NumBuff: 50
batchinsert test_full 5 dbtest 50
Number of disk page reads: 262686870
Number of disk page writes: 590296
MAP Count: 29461
Distinct Row Count: 100
Distinct Column Count: 100
This process took: 887896 milliseconds
```

B.2 Query Results

```
Equality Search
Alabama Isogomphodon 86872
```

```
   Range Search
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000]
```

```
   Index Type: 1
Dataset: Full Testdata
NumBuff: 25
query dbtest 1 1
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

[Alabama, Isogomphodon, 49873, 86872]
Number of maps found: 1
Number of disk page reads: 2211
Number of disk page writes: 0
This process took: 36 milliseconds
```

```
   Order Type: 1
query dbtest 1 1
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

[Alabama, Angel_shark, 3572, 19874]
[Alabama, Carcharhiniformes, 34194, 17759]
[Alaska, Baboon, 40857, 17321]
[Alaska, Black_panther, 8596, 17176]
[Arizona, Apristurus, 9522, 2]
[Arizona, Camel, 4031, 12678]
[Australia, Angel_shark, 6430, 1245]
[Australia, Baboon, 4641, 1855]
[Barbados, Baboon, 117, 14069]
[Barbados, Black_panther, 21258, 17754]
[COSTA_RICA, Apristurus, 49281, 142]
[COSTA_RICA, Badger, 48979, 10130]
[COSTA_RICA, Beaver, 34773, 18779]
[COSTA_RICA, Bison, 38910, 15330]
[COSTA_RICA, Bullhead_shark, 21214, 19236]
[CZECH_REPUBLIC, Apristurus, 2986, 14504]
[CZECH_REPUBLIC, Bison, 26463, 13604]
[California, Angel_shark, 46014, 144]
[California, Bison, 24690, 11495]
[Canada, Apristurus, 29465, 16212]
[Canada, Bison, 11004, 14726]
[Canada, Black_panther, 27117, 16109]
[Canada, Bullhead_shark, 1613, 1984]
[Canada, Carcharhiniformes, 20923, 10381]
[Colombia, Badger, 5555, 14737]
```

```
[Colombia, Carcharhiniformes, 5823, 10314]
[Croatia, Badger, 2226, 10079]
Number of maps found: 27
Number of disk page reads: 2211
Number of disk page writes: 0
This process took: 37 milliseconds


   Order Type: 2
query dbtest 1 2
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

[Alabama, Angel_shark, 3572, 19874]
[Australia, Angel_shark, 6430, 1245]
[California, Angel_shark, 46014, 144]
[Arizona, Apristurus, 9522, 2]
[COSTA_RICA, Apristurus, 49281, 142]
[CZECH_REPUBLIC, Apristurus, 2986, 14504]
[Canada, Apristurus, 29465, 16212]
[Alaska, Baboon, 40857, 17321]
[Australia, Baboon, 4641, 1855]
[Barbados, Baboon, 117, 14069]
[COSTA_RICA, Badger, 48979, 10130]
[Colombia, Badger, 5555, 14737]
[Croatia, Badger, 2226, 10079]
[COSTA_RICA, Beaver, 34773, 18779]
[COSTA_RICA, Bison, 38910, 15330]
[CZECH_REPUBLIC, Bison, 26463, 13604]
[California, Bison, 24690, 11495]
[Canada, Bison, 11004, 14726]
[Alaska, Black_panther, 8596, 17176]
[Barbados, Black_panther, 21258, 17754]
[Canada, Black_panther, 27117, 16109]
[COSTA_RICA, Bullhead_shark, 21214, 19236]
[Canada, Bullhead_shark, 1613, 1984]
[Arizona, Camel, 4031, 12678]
[Alabama, Carcharhiniformes, 34194, 17759]
[Canada, Carcharhiniformes, 20923, 10381]
[Colombia, Carcharhiniformes, 5823, 10314]
Number of maps found: 27
Number of disk page reads: 2211
Number of disk page writes: 0
This process took: 33 milliseconds
```

```
   Order Type: 3
query dbtest 1 3
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

[Alabama, Angel_shark, 3572, 19874]
[Alabama, Carcharhiniformes, 34194, 17759]
[Alaska, Black_panther, 8596, 17176]
[Alaska, Baboon, 40857, 17321]
[Arizona, Camel, 4031, 12678]
[Arizona, Apristurus, 9522, 2]
[Australia, Baboon, 4641, 1855]
[Australia, Angel_shark, 6430, 1245]
[Barbados, Baboon, 117, 14069]
[Barbados, Black_panther, 21258, 17754]
[COSTA_RICA, Bullhead_shark, 21214, 19236]
[COSTA_RICA, Beaver, 34773, 18779]
[COSTA_RICA, Bison, 38910, 15330]
[COSTA_RICA, Badger, 48979, 10130]
[COSTA_RICA, Apristurus, 49281, 142]
[CZECH_REPUBLIC, Apristurus, 2986, 14504]
[CZECH_REPUBLIC, Bison, 26463, 13604]
[California, Bison, 24690, 11495]
[California, Angel_shark, 46014, 144]
[Canada, Bullhead_shark, 1613, 1984]
[Canada, Bison, 11004, 14726]
[Canada, Carcharhiniformes, 20923, 10381]
[Canada, Black_panther, 27117, 16109]
[Canada, Apristurus, 29465, 16212]
[Colombia, Badger, 5555, 14737]
[Colombia, Carcharhiniformes, 5823, 10314]
[Croatia, Badger, 2226, 10079]
Number of maps found: 27
Number of disk page reads: 2211
Number of disk page writes: 0
This process took: 36 milliseconds


   Order Type: 4
query dbtest 1 4
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

[Alabama, Angel_shark, 3572, 19874]
[Australia, Angel_shark, 6430, 1245]
[California, Angel_shark, 46014, 144]
```

```
[CZECH_REPUBLIC, Apristurus, 2986, 14504]
[Arizona, Apristurus, 9522, 2]
[Canada, Apristurus, 29465, 16212]
[COSTA_RICA, Apristurus, 49281, 142]
[Barbados, Baboon, 117, 14069]
[Australia, Baboon, 4641, 1855]
[Alaska, Baboon, 40857, 17321]
[Croatia, Badger, 2226, 10079]
[Colombia, Badger, 5555, 14737]
[COSTA_RICA, Badger, 48979, 10130]
[COSTA_RICA, Beaver, 34773, 18779]
[Canada, Bison, 11004, 14726]
[California, Bison, 24690, 11495]
[CZECH_REPUBLIC, Bison, 26463, 13604]
[COSTA_RICA, Bison, 38910, 15330]
[Alaska, Black_panther, 8596, 17176]
[Barbados, Black_panther, 21258, 17754]
[Canada, Black_panther, 27117, 16109]
[Canada, Bullhead_shark, 1613, 1984]
[COSTA_RICA, Bullhead_shark, 21214, 19236]
[Arizona, Camel, 4031, 12678]
[Colombia, Carcharhiniformes, 5823, 10314]
[Canada, Carcharhiniformes, 20923, 10381]
[Alabama, Carcharhiniformes, 34194, 17759]
Number of maps found: 27
Number of disk page reads: 2211
Number of disk page writes: 0
This process took: 33 milliseconds
```

Order Type: 5

```
query dbtest 1 5
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

[Barbados, Baboon, 117, 14069]
[Canada, Bullhead_shark, 1613, 1984]
[Croatia, Badger, 2226, 10079]
[CZECH_REPUBLIC, Apristurus, 2986, 14504]
[Alabama, Angel_shark, 3572, 19874]
[Arizona, Camel, 4031, 12678]
[Australia, Baboon, 4641, 1855]
[Colombia, Badger, 5555, 14737]
[Colombia, Carcharhiniformes, 5823, 10314]
[Australia, Angel_shark, 6430, 1245]
[Alaska, Black_panther, 8596, 17176]
[Arizona, Apristurus, 9522, 2]
[Canada, Bison, 11004, 14726]
[Canada, Carcharhiniformes, 20923, 10381]
[COSTA_RICA, Bullhead_shark, 21214, 19236]
[Barbados, Black_panther, 21258, 17754]
[California, Bison, 24690, 11495]
[CZECH_REPUBLIC, Bison, 26463, 13604]
[Canada, Black_panther, 27117, 16109]
[Canada, Apristurus, 29465, 16212]
[Alabama, Carcharhiniformes, 34194, 17759]
[COSTA_RICA, Beaver, 34773, 18779]
[COSTA_RICA, Bison, 38910, 15330]
[Alaska, Baboon, 40857, 17321]
[California, Angel_shark, 46014, 144]
[COSTA_RICA, Badger, 48979, 10130]
[COSTA_RICA, Apristurus, 49281, 142]
Number of maps found: 27
Number of disk page reads: 2211
Number of disk page writes: 0
This process took: 41 milliseconds
```

Index Type: 1
Dataset: Full Testdata
NumBuff: 50

```
query dbtest 1 1 Alabama Isogomphodon 86872 50

Number of disk page reads: 2211
Number of disk page writes: 0
This process took: 84 milliseconds
```

Order Type: 1

```
query dbtest 1 1
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50
Number of disk page reads: 2211 Number of disk page
writes: 0 This process took: 45 milliseconds
```

Order Type: 2

```
query dbtest 1 2
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50

Number of disk page reads: 2211
Number of disk page writes: 0
This process took: 36 milliseconds


   Order Type: 3

   query dbtest 1 3
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50

Number of disk page reads: 2211
Number of disk page writes: 0
This process took: 36 milliseconds


   Order Type: 4

   query dbtest 1 4
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50

Number of disk page reads: 2211
Number of disk page writes: 0
This process took: 37 milliseconds


   Order Type: 5

   query dbtest 1 5
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50

Number of disk page reads: 2211
Number of disk page writes: 0
This process took: 38 milliseconds


   Index Type: 2
Order Type: 4
Dataset: Full Testdata
NumBuff: 25

   query dbtest 2 1 Alabama Isogomphodon 86872 25

Number of disk page reads: 6505668
Number of disk page writes: 240
This process took: 3003 milliseconds

   Order Type: 1
query dbtest 2 1
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25
Number of disk page reads: 6505668
Number of disk page writes: 240
This process took: 16744 milliseconds


   Order Type: 2
query dbtest 2 2
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

Number of disk page reads: 6505668
Number of disk page writes: 240
This process took: 14954 milliseconds


   Order Type: 3
query dbtest 2 3
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

Number of disk page reads: 6505668
Number of disk page writes: 240
This process took: 17734 milliseconds


   Order Type: 4
query dbtest 2 4
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

Number of disk page reads: 6505668
Number of disk page writes: 240
This process took: 14708 milliseconds


   Order Type: 5
query dbtest 2 5
```

```
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

Number of disk page reads: 6505668
Number of disk page writes: 240
This process took: 15801 milliseconds


   Index Type: 2
Dataset: Full Testdata
NumBuff: 50

   query dbtest 2 1 Alabama Isogomphodon 86872 50

Number of disk page reads: 1233210
Number of disk page writes: 41
This process took: 2834 milliseconds

   Order Type: 1
query dbtest 2 1
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50
Number of disk page reads: 12801314
Number of disk page writes: 483
This process took: 29085 milliseconds


   Order Type: 2
query dbtest 2 2
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50

Number of disk page reads: 12801314
Number of disk page writes: 483
This process took: 30443 milliseconds


   Order Type: 3
query dbtest 2 3
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50

Number of disk page reads: 12801314
Number of disk page writes: 483
This process took: 30258 milliseconds


   Order Type: 4
query dbtest 2 4
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50
Number of disk page reads: 12801314
Number of disk page writes: 483
This process took: 30342 milliseconds


   Order Type: 5
query dbtest 2 5
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50
Number of disk page reads: 12801314
Number of disk page writes: 483
This process took: 29184 milliseconds


   Index Type: 3
Dataset: Full Testdata
NumBuff: 25

query dbtest 3 1 Alabama Isogomphodon 86872 25
Number of disk page reads: 608522
Number of disk page writes: 25
This process took: 24476 milliseconds

Order Type: 1
query dbtest 3 1
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50
Number of disk page reads: 5881141
Number of disk page writes: 224
This process took: 24776 milliseconds

Order Type: 2
query dbtest 3 2
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50
Number of disk page reads: 5881141
Number of disk page writes: 224
```

```
This process took: 24771 milliseconds

Order Type: 3
query dbtest 3 3
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50
Number of disk page reads: 5881141
Number of disk page writes: 224
This process took: 25245 milliseconds

Order Type: 5
query dbtest 3 5
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50
Number of disk page reads: 5881141
Number of disk page writes: 224
This process took: 25180 milliseconds

Index Type: 4
Dataset: Full Testdata
NumBuff: 25

query dbtest 4 1 Alabama Isogomphodon 86872 25

Number of disk page reads: 3709
Number of disk page writes: 1
This process took: 26 milliseconds

Order Type: 1
query dbtest 4 1
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

Number of disk page reads: 3161312
Number of disk page writes: 121
This process took: 7507 milliseconds

Order Type: 2
query dbtest 4 2
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

Number of disk page reads: 3161312
Number of disk page writes: 121
This process took: 7425 milliseconds

Order Type: 3
query dbtest 4 3
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

Number of disk page reads: 3161312
Number of disk page writes: 121
This process took: 7429 milliseconds

Order Type: 4
query dbtest 4 4
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

Number of disk page reads: 3161312
Number of disk page writes: 121
This process took: 7411 milliseconds

Order Type: 4
query dbtest 4 5
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

Number of disk page reads: 3161312
Number of disk page writes: 121
This process took: 7719 milliseconds

Index Type: 4
Dataset: Full Testdata
NumBuff: 50

query dbtest 4 1 Alabama Isogomphodon 86872 50

Number of disk page reads: 3709
Number of disk page writes: 1
This process took: 22 milliseconds

Order Type: 1
query dbtest 4 1
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50
```

```
Number of disk page reads: 3161409
Number of disk page writes: 121
This process took: 23510 milliseconds

Order Type: 2
query dbtest 4 2
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50

Number of disk page reads: 3161409
Number of disk page writes: 121
This process took: 23382 milliseconds

Order Type: 3
query dbtest 4 3
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50

Number of disk page reads: 3161409
Number of disk page writes: 121
This process took: 25465 milliseconds

Order Type: 4
query dbtest 4 4
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50

Number of disk page reads: 3161409
Number of disk page writes: 121
This process took: 20233 milliseconds

Order Type: 5
query dbtest 4 5
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50

Number of disk page reads: 3161409
Number of disk page writes: 121
This process took: 22215 milliseconds

Index Type: 5
Dataset: Full Testdata
NumBuff: 25
query dbtest 5 1 Alabama, Isogomphodon, 86872 25
```

```
Number of disk page reads: 1270
Number of disk page writes: 1
This process took: 224 milliseconds

Order Type: 1
query dbtest 5 1
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

Number of disk page reads: 5997898
Number of disk page writes: 202
This process took: 18843 milliseconds

Order Type: 2
query dbtest 5 2
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

Number of disk page reads: 5997898
Number of disk page writes: 202
This process took: 18023 milliseconds

Order Type: 3
query dbtest 5 3
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

Number of disk page reads: 5997898
Number of disk page writes: 202
This process took: 18466 milliseconds

Order Type: 4
query dbtest 5 4
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 25

Number of disk page reads: 6000313
Number of disk page writes: 204
This process took: 17407 milliseconds

Order Type: 5
query dbtest 5 5
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
```

```
[1000,2000] 25

Number of disk page reads: 6000313
Number of disk page writes: 204
This process took: 16847 milliseconds

  Index Type: 5
Dataset: Full Testdata
NumBuff: 50
query dbtest 5 1 Alabama, Isogomphodon, 86872 25:

Number of disk page reads: 2504
Number of disk page writes: 1
This process took: 235 milliseconds

Order Type: 1
query dbtest 5 1
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50

Number of disk page reads: 11799355
Number of disk page writes: 405
This process took: 36934 milliseconds

Order Type: 2
query dbtest 5 2
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50

Number of disk page reads: 11799355
Number of disk page writes: 405
This process took: 35476 milliseconds

Order Type: 3
query dbtest 5 3
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50

Number of disk page reads: 11799355
Number of disk page writes: 405
This process took: 36764 milliseconds

Order Type: 4
query dbtest 5 4
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50

Number of disk page reads: 11799355
Number of disk page writes: 405
This process took: 35191 milliseconds

Order Type: 5
query dbtest 5 5
[Alabama,Croatia]
[Angel_shark,Carcharhiniformes]
[1000,2000] 50

Number of disk page reads: 11799355
Number of disk page writes: 405
This process took: 34625 milliseconds
```