COSC 404 Midterm

Key Concepts:

* Insert and Delete from B+ tree (4 marks)
* Insert with linear hashing (2 marks)
* Perform RAID and index calculations (6 marks)
* Code an iterator in Java (lab 4) (5 marks)
* Create relational query plans in Java (5 marks)

Topics:

1. **Storage**
   1. Memory performance calculations
      1. Transfer time = memoryTransferSize/bandwidth
      2. Read time = readDataSize/readBandwidth
      3. Write bandwidth = IOPS \* writeRequests
   2. Storing records in memory
      1. A record consists of one or more fields grouped together
         1. Each Tuple of a relation is a record
      2. Two main types of records:
         1. Variable length: size of the record varies
         2. Fixed length: all records are the same size
   3. Variable formats
      1. Useful cases:
         1. the data does not have a regular structure in most cases
         2. the data values are sparse in the records
         3. there are repeating fields in the records
         4. the data evolves quickly so schema evolution is challenging
      2. Disadvantages:
         1. Space is wasted by repeating schema information for every record
         2. allocating variable-sized records efficiently is challenging
         3. query processing is more difficult and less efficient when the structure of the data varies
      3. JSON & XML are best described as variable format, variable size
      4. A VARCHAR field is best described as fixed format, variable size
   4. Storing records in blocks
      1. Issues related to storing records in blocks
         1. Separation: how are adjacent records separated?
            1. variable length records can be separated by:

a special separator marker in the block

storing the size of the record at the start of each record

store the length or offset of each record in the block header

* + - 1. Spanning: can records cross block boundaries?
         1. unspanned: do not allocate records across blocks and waste space

Blocks needed for unspanned records = 1record/block

utilization = recordSize/blockSize\*100%

* + - * 1. spanned: start a record at the end of a block and continue on to the next (round up)

each piece must have a pointer to its other part

blocks needed (numBlocks) = numRecords\*recordSize/blockSize

utilization = (numRecords\*recordSize)/(blockSize\*numBlocks)

* + - 1. Clustering: how many records can a block store?
         1. Allocating records of different types together on the same block/file because they are frequently accessed together
         2. Beneficial when DB commonly gets queries with where clause
         3. Harmful for select \* queries because the system reads more blocks and each block read has information that doesn’t answer the query
      2. Splitting: how many blocks are records allocated in?
         1. Split record: a record where portions of the record are allocated on multiple blocks for reasons other than spanning

Used with or without spanning

* + - * 1. Hybrid records

Fixed portions of records are stored with other fixed records on their own block(s)

Variable parts are stored with other variable record parts on their own block(s)

* + - * 1. Efficient & simplifies allocation
        2. Fixed part of record is easier to allocate and optimize for access
      1. Ordering: are the records sorted in any way?
         1. Records in a block are sorted based on the value of 1+ fields
         2. Allows searching for keys and performing joins to be made faster
         3. Physical ordering: records are allocated in their blocks in sorted order
         4. Logical ordering: records are not physically sorted but each has a pointer to the next record in sorted order
      2. Addressing: how each record is referenced
         1. Method for defining unique values or addresses to reference particular records
         2. Physical addressing: a record has a physical address based on the device where it’s stored

Better performance because record can be accessed directly w/o lookup cost

* + - * 1. Logical addressing: records have a key value or other identifier that can be used to lookup their physical address in a table

Does not provide method for locating record directly

More flexible because records can be moved on the physical device and only the mapping table needs to be updated

Easier to move, update, change

* + - 1. Pointer Swizzling: the process for converting disk pointers to memory pointers and vice versa when blocks move between memory and disk
  1. Objectives
     1. **Compare/contrast volatile versus non-volatile memory.**
        1. Volatile memory: retains data only when power is on
           1. DRAM
           2. Main memory
        2. Non-volatile/permanent memory: stores data even when power is off
           1. SSD
           2. Flash memory
           3. Hard drive
     2. **Compare/contrast random access versus sequential access.**
        1. Random access: allows retrieval of any data location in any order
        2. Sequential access: requires visiting all previous locations in sequential order to retrieve a given location
     3. **Use both metric and binary units for memory sizes.**
        1. 1 byte = 8 bits
           1. 1 bit = 1 or 0
        2. 1 byte = 1 character
     4. **List the benefits of RAID and common RAID levels.**
        1. RAID Benefits
           1. Improves reliability through redundantly storing extra data that is used to recover from a disk failure
           2. Parallelism allows for increased throughput and large accesses to reduce response time
        2. RAID Levels
           1. 0: is for high-performance where data loss is not critical (parallelism)

Striping at the block level

Non-redundant

* + - * 1. 1: has redundancy and protection from disk failures with minimum cost

Requires at least 2 disks

Mirrored disks

* + - * 1. 5: offers reliability and increased performance

No single disk bottleneck like RAID 4

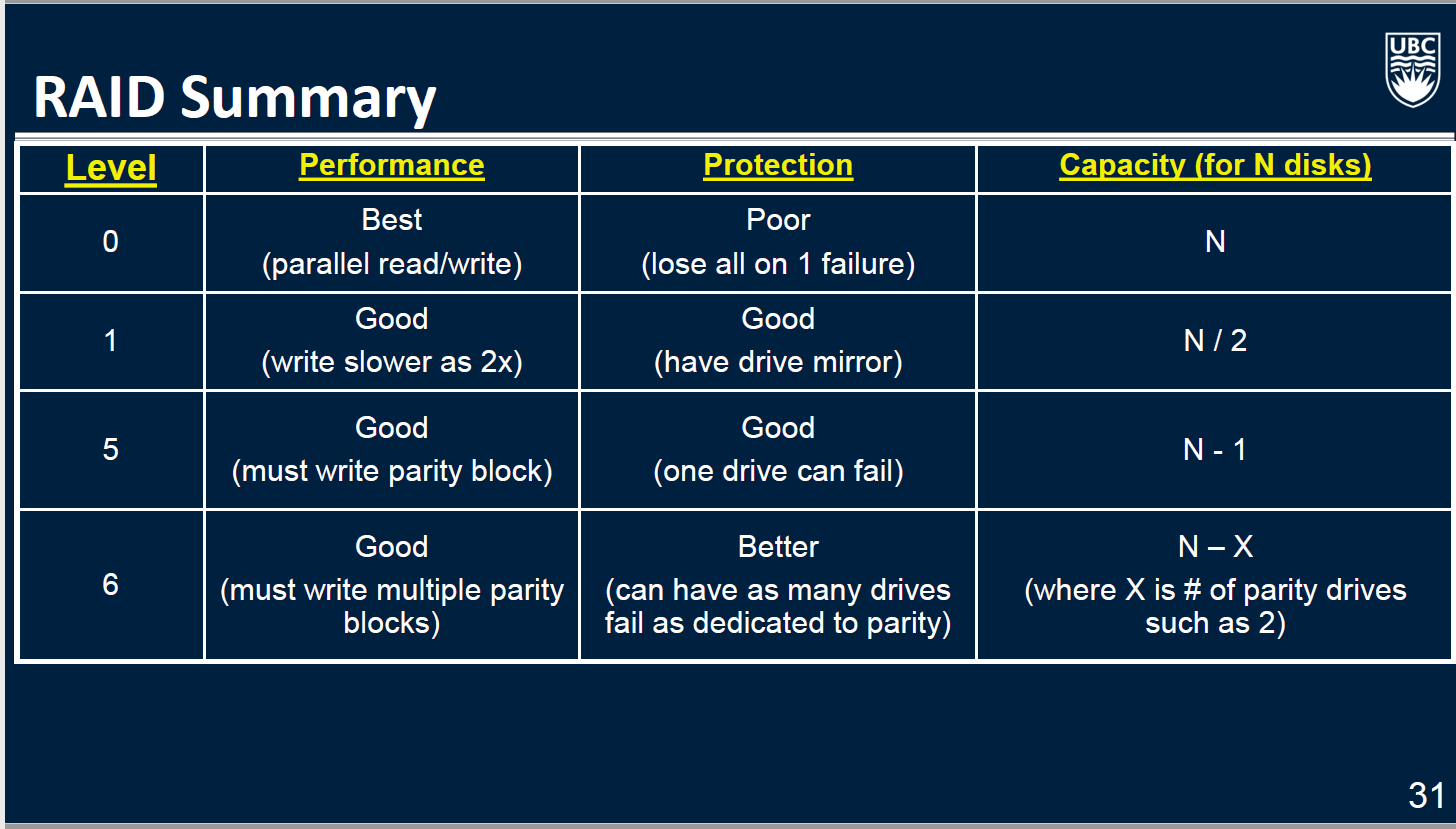
Block interleaved distributed parity

Partitions data and parity among N+1 disks

* + - * 1. 6: offers extra redundancy compared to RAID 5 and is best for mitigating multiple drive failures

P+Q redundancy scheme

Stores extra info than RAID 5



* + 1. **List different ways for representing strings in memory.**
       1. Null-terminated String: last byte value of 0 indicates end
       2. Byte-length String: length of string in bytes is specified in first few bytes before string starts
       3. Fixed-length String: always the same size
    2. **List different ways for representing date/times in memory.**
       1. Date representations
          1. Integer representation: number of days passed since a given date
          2. String representation: show a date’s components as individual characters of a string (YYYYMMDD)
       2. Time representations
          1. Integer representation: number of seconds since a given time
          2. String representation: hours, minutes, seconds, fractions (HH:MM:SS:FF)
    3. **Explain the difference between fixed and variable length records.**
       1. Fixed-length records: all records have the same size
       2. Variable-length records: all records have the same size
    4. **Compare/contrast the ways of separating fields in a record.**
       1. *No separator*: store length of each field, so do not need a separate separator (fixed length field). Simple but wastes space within a field.
       2. *Length indicator*: store a length indicator at the start of the record (for the entire record) and a size in front of each field. Wastes space for each length field and need to know length beforehand.
       3. *Use offsets*: at start of record store offset to each field
       4. *Use delimiters*: separate fields with delimiters such as a comma (comma-separated files). Must make sure that delimiter character is not a valid character for field.
       5. *Use keywords*: self-describing field names before field value (XML and JSON). Wastes space by using field names
    5. **Define and explain the role of schemas.**
       1. *Schema*: description of the record layout
       2. Contains…
          1. Names and number fields
          2. Size and type of each field
          3. Field ordering in record
          4. Description/meaning of each field
    6. **Compare/contrast variable and fixed formats.**
       1. *Fixed record format*: every record has the same fields with the same types
          1. Relational schemas
       2. *Variable record format*: not all records have the same fields or organization
          1. Record data must be *self-describing*
          2. XML/JSON
    7. **List and briefly explain the six record placement issues in blocks.**
       1. *Separation:* how do we separate adjacent records?
       2. *Spanning:* can a record cross a block boundary?
       3. *Clustering*: can a block store multiple record types?
       4. *Splitting*: are records allocated in multiple blocks?
       5. *Ordering*: are the records sorted in any way?
       6. *Addressing*: how do we reference a given record?

1. **Indexing**
   1. **Objectives**
      1. Explain the types of indexes
      2. Perform calculations on how fast it takes to retrieve one record or answer a query given a certain data file and index type
      3. **Define:**
         1. Index file: the file that stores the index information
         2. Data file: the file that actually contains the records
         3. Search key: the set of attributes stored by the index to find the records in the data file
            1. Does not have to be unique
            2. More than one record may have the same search key value
         4. Index entry: one index record that contains a search key value and a pointer to the location of the record with that value
      4. **List the index evaluation metrics/criteria:**
         1. Functionality: measured by the types of queries it supports
            1. Exact match on search key
            2. Query on a range of search key values
         2. Performance: measured by the time required to excecute queries and update the index
            1. Access time
            2. Update
            3. Insert
            4. Delete time
         3. Efficiency: measured by the amount of space required to maintain the index structure
      5. **Explain the difference between the different types of indexes**
         1. Dense index: has an index entry for every record in the data file
         2. Sparse index: has index entries for only some of the data file records (often by block)
         3. Primary index (clustering indexes): sorts the data file by its search key (doesn’t have to be the same as primary key)
         4. secondary index: does not determine the organization of the data file
         5. single-level index: has only one index level
         6. multi-level index: has several levels of indexes in the same file
      6. **List the techniques for indexing with duplicate search keys**
         1. Create an index entry for each record
            1. Wastes space (key/value are repeated for each record)
         2. Use buckets/blocks to store records with the same key
            1. Index entry points to the first record in the bucket
            2. All other matching records are retrieved from the bucket
      7. **Discuss some of the issues in index maintenance**
         1. as the data file changes, the index must be updated as well
         2. similar to ordered file maintenance but index files are smaller
         3. techniques for managing data file (dense & sparse indexes):
            1. using overflow blocks
            2. re-organizing blocks by shifting records
            3. adding or deleting new blocks in the file
      8. **Compare/contrast single versus multi-level indexes**
         1. Multi-level index: has more than one index level for the same data
            1. Each level of the multi-level index is smaller
            2. Can process each level more efficiently
            3. First level can be sparse or dense
            4. All higher levels must be sparse
            5. Index maintenance time increases with each level
         2. Single-level index has only one index level
            1. Can be sparse or dense
      9. **Explain the benefit of secondary indexes on query performance**
         1. Simpler maintenance of secondary index
            1. Secondary index changes only when primary index changes, not when the data file changes
      10. **Index calculations**
          1. Index entries/block = blockSize/idxEntrySize
          2. Records/block = blockSize/recordSize
          3. Number of index blocks = numRecords/(idxEntries/block)
          4. Number of blocks = numRecords/(records/block)
          5. Binary search blocks retrieved = ciel(log2(numBlocks))+1
          6. Linear search blocks retrieved = numBlocks/2
2. **B-Trees**
   1. **Objectives**
      1. **Insert and delete from a B-tree and a B+ tree**
      2. **Calculate the maximum order of a B-tree**
      3. **Calculate query access times using B-tree indexes**
      4. **Compare/contrast B-trees and B+ trees**
3. **R-Trees**
   1. **Objectives**
      1. **Explain the difference between an R-tree and a B+ tree**
      2. **List some types of spatial data**
      3. **List some types of spatial queries**
      4. **List some applications of spatial data and queries**
      5. **Explain the idea of insertion in an R-tree**
4. **Hashing**
   1. **Objectives**
      1. **Perform open address hashing with linear probing**
      2. **Perform linear hashing**
      3. **Define:**
         1. **Hashing:**
         2. **Collision:**
         3. **Perfect hash function:**
      4. **Calculate load factor of a hash table**
      5. **Compare/contrast external hashing and main memory hashing**
      6. **Compare/contrast B+ trees and linear hashing**
5. **SQL Indexing**
   1. **Objectives**
      1. **Use index structures in SQL using CREATE/ALTER commands**
      2. **Perform insertions and searches using partitioned hashing**
      3. **Perform searches using grid files**
      4. **Understand how bitmap indexes are used for searching and why they provide a space and speed improvement in certain cases**
6. **Query Processing**
   1. **Objectives**
      1. **Diagram the query processor components and explain their function**
      2. **Calculate block access for one-pass algorithms**
      3. **Calculate block accesses for tuple and block nested joins**
      4. **Perform two-pass sorting methods including all operators, sort-join and sort-merge-join and calculate performance**
      5. **Perform two-pass hashing methods including all operators, hash-join and hybrid-hash-join and calculate performance**
      6. **Explain the goal of query processing**
      7. **List the relational and set operators**
      8. **Diagram and explain query processor components**
      9. **Explain how index and table scans work and calculate the block operations performed**
      10. **Write an iterator in java for a relational operator**
      11. **List the tuple-at-a-time relational operators**
      12. **Illustrate how one-pass algorithms for selection, project, duplicate elimination, and binary operators work and calculate performance and memory requirements**
      13. **Calculate performance of tuple-based and block-based nested loop joins given relation sizes**
      14. **Perform and calculate performance of two-pass sorting based algorithms, sort-merge algorithm, set operators, sort-merge-join/sort-join**
      15. **Perform and calculate performance of two-pass hashing based algorithms, hash partitioning, operation implementation and performance, hash join, hybrid-hash-join**
      16. **Compare/contrast sorting versus hashing methods**
      17. **Calculate performance of index-based algorithms, cost estimate, complicated sections, index joins**
      18. **Explain how two-pass algorithms are extended to multi-pass algorithms**
      19. **List some recent join algorithms**
7. **Query Optimization \*\*Not on Midterm\*\***
   1. **Objectives**
      1. **Convert an SQL query to a parse tree using a grammar**
      2. **Convert a pars tree to a logical query tree**
      3. **Use heuristic optimization and relational algebra laws to optimize logical query trees**
      4. **Convert a logical query tree to a physical query tree**
      5. **Calculate size estimates for selection, projection, joins, and set operations**
      6. **Explain the difference between syntax and semantic validation and the query processor component responsible for each**
      7. **Define:**
         1. **Valid parse tree**
         2. **Logical query tree**
         3. **Physical query tree**
         4. **Join-orders**
         5. **Left-deep**
         6. **Right-deep**
         7. **Balanced join trees**
      8. **Explain the difference between correlated and uncorrelated nested queries**
      9. **Define and use canonical logical query trees**
      10. **Explain issues in selecting algorithms for selection and join**
      11. **Compare/contrast materialization versus pipelining and know when to use them when building physical query plans**