Memory taken (how big the data structure is… 2GB, etc.)
Use 2 months instead of the 5 months

- Report on structure of our dataset after indexing

Experimental results

- Run experiments on 2 separate team member's computers, see how performance differs

Visualizations

- What happened to performance as test size increases
- Characterizing datasets (probably tables rather than graphs)
- Distribution of words to files (histogram)

AVL tree

- Height 21 (should be bigger)
- 178 seconds
- Average words in linked list = 268

Binary search tree

- 112 seconds

Idea is to use same dataset in same order for all data structures
Relational databases may not be the solutoin to all problems:

- Schemas evolve over time
    - Using data model w/ weaker schema enforcement could be beneficial
- Not all apps need full ACID compliance
    - These rules can be slow
- Joins (result of breaking data up into multiple tabes) are expensive
    - In relational schema you break data into tables b/c the more you split it up, the less data duplication you have
- A lot of data is semi structured or unstructured
- Joins, ACID slows data and is less performant

Scalability

- Vertical: more powerful systems on 1 computer
    - But more power is not always the answer
- Horizontal: split into multiple nodes -> cheaper
    - Distributed system: collection of independent computers that appear as 1. Operate concurrently, fail independently

Distributed Storage: 2 directions

- Protect data by replicating
    - purpose: can globally distribute easily -> geographic availability (CDN provides this); if 1 computer dies we'll be ok
- Increase speed by sharding

Distributed data stores

- Data stored on >1 node is replicated
    - Blocks of data should be available on multiple nodes
- Distributed databases can be relational or non
- System needs to be partition tolerant (can run even w/ network partition)

Protecting our data - CAP theorem

- Can only have 2/3 dimensions, never all 3
- Consistency: every read receives the most recent write or error thrown
    - Distributed data has inconsistent structure globally, so a read may not be the most recent
- Availability: every request receives response but there is no guarantee that the response contains the most recent write
- Partition tolerance: system operates as soon as there is partition in network

Ex. Consistency + availability: system always responds with latest data and every request gets a response, but not good w network issues…
CAP in reality

- If you cannot limit the number of faults, requests can be directed to any server, and you insist on serving every request, then yo ucannot possibly be consistent
- You must always give up something: consistency, availability, tolerance for failure

ACID transactions

- Focus on data safety
- Pessimistic concurrency model
    - Assumes that transactions wil lconflict/there will be a problem
    - Assumes that if something could go wrong, it will
- Conflicts prevented by locking resources until transaction is complete
- Optimistic concurrency
    - Assumes conflicts are unlikely, even if there is, it'll be ok
    - Transactions do not obtain locks on data when they read or write
    - Add last update timestamp and version # columns to every table, read them when changing, check at end of transaction to see if any other transaction has caused it to be changed
    - Don't actually do anything unless there's a conflict
    - Low conflict systems
        - Read heavy system
            - Cana't have conflicts when you're only reading
            - Conflicts that arise can be handled by rolling back and re-running transaction that notices a conflict
            - Optimistic concurrency better
        - High conflict system
            - Roll back and rerun transactions the encounter conflict -> less efficient
            - Locking scheme preferable

No SQL

- Originally described relational database system that did not use SQL
- Nowadays its more like not relations DBs
- Relax restrictions around ACID

ACID alternative for distributed systems: BASE

- Basically Available:
    - Guarantees availability of data (per CAP) but response can be feailur/eunreliable bc data is inconsistent/changing state
    - System appeared to work most of time
- Soft state:

        - state of system could change over time, even w/o input. 2 nodes may not have same data for a period of time.
        - Ultimately updates will catch up and all nodes will have same data (eventual consistency)
- Eventual consistency
    - As long as writes, all nodes will have same data

Categories of NoSQL DBs

- Document DBs
    - Store JSON
- Graph DBs
- Key value
    - Designed around 3 principles:
        - simplicity:
            - Very simple compared to TDBMS
        - Scalability:
            - Operate in eventual consistency model
        - Speed:
            - Deployed as in memoy DB
            - No notion of persistence to secondary storage
    - Use cases:
        - Store intermediate results from data preprocessing/EDA very efficiently
        - Caching scenarios
            - User profiles
            - Shopping cart data
                - Data needs to be available across browsers, machines, sessions
            - Storing session info
- Columnar DBs
- Vector
- Redis (remote directory server)
    - Open source, in memoryy
    - Sometimes called a data structure store
    - Primarily KV store but can be use w/ other models: graph, spatial, etc.
    - can get benefits of math thats more persistent

- Amazon not able to keep up w/ web traffic on their site… came up with distributed key value store -> that became dynamo DB
- Supports durability by saving snapshots to disk + keeping journal of changes that can be used for roll forward if there's failure
- Incredibly fast
- Not meant to handle complex data (only supports lookup by key)
    - In that case, go to document DB
- Data types
    - keys:
        - Strings tpically
    - values:
        - Strings, linked lists, sets, sorted sets, hashes…
- Provides 16 unnamed databases by default
- Use data grip (same commands as CLI) to control
    - Can access redis through python

Hash methods
Create slot
Compute slot
Bum docs num tokens search set base size search time
expeirment.py