Bioinformatics Resources
- NoSQL -

Lecture & Exercises
Prof. B. Rost, Dr. L. Richter, J. Reeb
Institut für Informatik I12
Orga - Exam

- Change confirmed: Wednesday, Jul 22\textsuperscript{nd}, 15-17 o’clock
- Interimshörsaal 2
- Please register in TUMonline before Jun 30\textsuperscript{th}
- Lecture evaluation
- Information event about 5./6. semester at 12.00 i.e. today’s lecture will close earlier
Short SQL Recap

- schema
- typed data
- tables
- defined layout
- space consumption is computable
Short SQL Recap

- well defined theory
- relational algebra
- ACID principle
- standardized query language
- fast access with indices
- well supported by software vendors
No SQL

- in principle known for a long time
- Ken Thompson 1978: Key/Value system
- big push in 2000: Web 2.0
- Map/Reduce, BigTable databases
- data volume in the range of TB and PB
- growing relational databases more and more difficult on commodity hardware
- http://www.w3resource.com/mongodb/nosql.php
Definition

- non relational data model
- enables distributed and horizontal scalability
- open source
- no or simple schema
- support for simple data replication
- simple API
- different consistency model
Issues with Relational DB

- is the schema bad, the query also is
- based on strings, susceptible for typos
- errors are not detected at compile time
- cannot be refactored
Categories of No SQL Systems

- Wide Column Stores/ Column Family Systems
- Document Stores
- Key/Values/Tuple Stores
- Graph Databases
Key/Value Systems

- at least very simple schema: key and value
- keys can be grouped in namespaces and databases
- values can be complex besides simple strings there are:
  - hashes
  - set
  - lists
- queries mostly limited to API
Column Family

- keys can point to an arbitrary number of key/value pairs
- nested key/value pairs
- nested columns
Document Stores

- works not on “actual” documents
- structured data like:
  - JSON
  - YAML
  - RDF
Graph Databases

- bases on graph or tree structures to connect elements
- property graph:
  - nodes to reflect items
  - edges to reflect relations
- very suitable for traversing
Theoretical Concepts

- Map/Reduce
- CAP-Theorem/ Eventually Consistent
- Consistent Hashing
- MVCC-Protocol
- Vector Clock
- Paxos
- REST
Map/Reduce

- require a (map/reduce) framework
- designed for efficient handling of data in the order of Tera or Peta bytes
- developed by Google
- patented since 2010
Map/Reduce Details

- originates from functional programming
- parallel processing
- no side effects
- no deadlocks
- no race conditions
- initial datastructure is not altered
- new copy with every level
Map/Reduce Details

- functions like in math:
  - a set of transformation definitions
  - no control structures
  - recursion
  - functions can be used as argument or return value: higher order functions
Map/Reduce Details

- two functions: map, reduce/fold
- used alternating (two phase approach)
- map (in parallel):
  - applied to all elements of list
  - returns a modified list
- reduce:
  - aggregate the return values from map into one result
Map/Reduce Details

- user has to provide:
  - map function
  - reduce function

- framework provides:
  - automatic parallelization and distribution
  - fault tolerance mechanisms for hard- and software failure
  - I/O scheduling
  - status and control information
Pseudocode Example

map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
Characteristics of a Map/Reduce System

- commodity hardware
- Ethernet network
- large number of nodes (>100)
- distributed file system, data is stored in chunks and redundant
- data are local to processing node
CAP and Eventually Consistent

- horizontal scaling of relational databases insufficient
  - too much time to extend database to more computers
  - frequently modification of source code required
- mostly due to implementation of ACID principle
CAP Theorem

- Consistency, availability and partition tolerance cannot all completely satisfied at the same time
- only two of these criteria can be satisfied at the same time, here: availability and partition tolerance is the important combination
- consistency is reduced
Consistency

- after a transaction the database is consistent, i.e.
  - all replicating nodes of database system have the same state after an transaction; changes are propagated to all nodes
  - read access to any node returns the same result
  - this require to wait for the completion of the propagation
Availability

- acceptable response time
- depends on the specific business case
- a certain response time is guaranteed up to a specified load level
Partition Tolerance

- if a node or a connection fails the system remains to be responsive
- in large computer centers those failures are frequent
BASE Consistency Model

- Basically available
- Soft state
- Eventually consistent
Characteristics

- focus on availability
- consistency is less important
- BASE is optimistic about consistency and defines it as a transition process and not as a defined state after a transaction
  -> Eventually Consistency
- consistent at some point in time
- interpretation different between systems
Levels of Consistency

- Causal Consistency
- Read-your-write Consistency
- Session Consistency
- Monotonic Read Consistency
- Monotonic Write Consistency
Consistent Hashing

- belongs to the family of hashing function
- maps elements of (potentially) very large source set to a hash value from a typically much smaller value set
- advantage: constant time
- applications:
  - check sums
  - securing against manipulations
  - fast search in data structures
Consistent Hashing

- here: find a constant place memory for an object
- minimize object movements on addition or removal of nodes
- minimize object movements upon insertions
- distribute equally among resources
- circular hash space
- servers and data object are integrated (clockwise)
- upon insertion or removal only neighbors are affected
Multiversion Concurrency Control (MVCC)

- data objects are versioned
- represents change timeline
- every write access creates a new version
- contains reference to the least recent version
- conflict resolution through explicit version comparison
Multiversion Concurrency Control (MVCC)

- disadvantage of conventional locks:
  - complete tables are locked
  - inefficient if communication time is high because of long cache pipeline or network traffic
  - not 100% guaranteed in distributed systems
  - parallel access are blocked