NoSQL

CSC230 Database Technologies for Analytics

15 November 2016

- relational databases
  - many products available
  - many people who know how to use relational databases
  - long history, lots of experience from which to draw lessons
- need for an alternative to relational databases
  - need to work with larger amounts of data (larger than older technology can handle)
  - need to work with multi-structured data
  - need a means to develop software more quickly
  - need for lower cost (open source)
- reasons to shift both operational and analytical applications away from relational technologies
- developer's perspectives (they influence choice of technology)
  - data types now different than in the past
    - structured
    - semi-structured
    - unstructured
    - polymorphic (many structures in same data)
    - cannot know data types in advance—types of data change over time (rapidly)
  - agile development model has replaced the waterfall model
    - waterfall—12-18 months between each new release of a product
    - agile—weeks (sometimes just days) between releases
    - agile—small teams
  - software-as-a-service (SAAS)
    - always on
    - clients around the world

        * connect via a great variety of hardware/ coupled to a great variety of software
    - "scale-out architectures"
        * open source software
        * commodity servers (many cheap, off-the-shelf computers rather a few, expensive, high performance machines)
        * cloud computing (computers off-site, connections thru the Internet)

- common characteristics of NoSQL systems
    - flexible data model
    - higher scalability
    - superior performance

- trade-offs to achieve advantages
    - give up expressive query language
    - give up secondary indices
    - give up consistency

- 5 dimensions for evaluating NoSQL/non-relational databases
    - data model
        * document model
            · intuitive, natural, general-purpose
            · documents—fields with types: string, binary, date, array, etc.
            · like JSON
            · like objects used in high-level languages
            · data not spread over multiple tables
            · query based on any combination of fields
            · (largely) eliminate joins
            · schema are dynamic—each document can contain different fields
        * graph model
            · nodes, edges, properties
            · useful for modeling social networks, supply chains
            · time required to learn this different model
        * key-value/wide column model
            · pairs of attribute names and attribute values
            · no set schema: good for polymorphic and unstructured data
            · attractive performance and scalability

        - data retrieved by primary key
        - value accessible only through key
        - wide-columns: sparse, distributed, multi-dimensional, sorted maps
        - wide-columns: columns can be grouped
        - narrow set of applications
    * bottom line
        - all of these data models provide schema flexibility
        - document model has widest applicability
        - document model easiest because of correspondence to objects
        - wide column: more granular access than key-value, less flexibility than document model
- query model
    * more efficient queries that possible with relational model
    * document stores allow richest query functions
    * key-store/wide column stores allow fastest queries
        - but limited query functions
        - additional costs at application level
- consistency model
    * relational model guaranteed consistency—people expect consistency
    * non-relational models: multiple copies of data (for availability, scalability)
    * non-relational models: consistent or eventually consistent
    * different approaches to coding in each case
    * MongoDB: tunable consistency, choice made at query level
    * eventually consistent: sychronization of copies over time
    * idempotent commands: same results every time (does not depend upon history)
    * eventually consistent: good for read-only systems, systems with infrequent changes
    * eventually consistent: advantage on inserts, additional costs and complexity on updates, reads
- API model
    * idiomatic drivers
        - tailored to way of working in each high-level language
        - easier to learn, use
        - MongoDB provides drivers for 10 languages, community provides drivers for 30 more languages
    * RESTful APIs

            - simple, familiar
            - latency inherent to HTTP
        * SQL-like
            - goal is to make learning curve shorter, less steep
            - much less powerful, expressive than full SQL
            - often support for queries but not inserts, updates
            - performance suffers, maintenance harder if SQL-like language fools programmers into assuming relational structure
    - commercial support and community strength
        * evaluate strength of company
        * evaluate company's commitment to technology, product
        * how many companies? how many customers?
        * how much competition? how many alternatives?
        * how much software (for development, testing, documentation), references and tutorials, case studies has community developed?

- MongoDB
    - maintain foundation established by developers of relational systems
    - expressive query language: access, manipulate, operational, analytical
    - consistency
    - integration, management: security, monitoring
    - flexible data model
    - scalability and performance
    - always-on, global
        * distribute databases over many nodes in many places
        * use from anywhere
        * use anytime