Issues in "Big-Data" Database Systems

19th International Command and Control Research and Technology Symposium (ICCRTS)
Paper #113

Jack Orenstein
Geophile, Inc.
jao@geophile.com

Marius Vassiliou
IDA Science & Technology Division
mvassili@ida.org

"Big Data"


DATA SCIENTIST
THE SEXIEST JOB OF THE 21ST CENTURY


I ♥
BIG DATA


PLANET HYPE


Harvard Business Review
The True Measures Of Success
10 Rules for Managing Global Innovation
What Ever Happened To Accountability?
GETTING CONTROL OF BIG DATA
How vast new streams of information are changing the art of management


McKinsey Global Institute
May 2011
Big data: The next frontier for innovation, competition, and productivity


BIG DATA
A REVOLUTION THAT WILL TRANSFORM HOW WE LIVE, WORK, AND THINK
VIKTOR MAYER-SCHÖNBERGER
KENNETH CUKIER

Big Data: “3 or 4 Vs”

- Volume
  - Large amounts of data
  - More than can be processed on a single laptop or server?
- Variety
  - Many forms
  - Traditional databases
  - Images
  - Video
  - Documents
  - Complex records
- Velocity
  - Content constantly changing
  - Absorption of complementary data collections
  - Introduction of previously archived data
  - Streamed data arriving from multiple sources
- Veracity
  - Uncertainty
  - Varying quality

See Berman, Jules K. (2013). *Principles of Big Data: Preparing, Sharing, and Analyzing Complex Information*. New York: Elsevier. 261pp.

Characterization of Database Systems (1)

- **Data model**
  - Primitives for structuring data.
- **Schema**
  - How data in that database is organized
  - May be highly constrained (e.g. SQL databases) or provide minimal structuring, (e.g. NoSQL database or POSIX filesystem).
- **APIs**
  - For connecting, retrieving data, loading and modifying data, configuration, and maintenance.
  - APIs are provided as function call interfaces in a variety of languages, as network protocols, and often as function call interfaces conveyed by a network protocol.
- **Query language**
  - In many database systems, APIs carry commands, specified in some query or search language, to the database,
  - e.g. SQL queries or Google-style searches.
  - Not all databases have a query language – some key/value stores provide only APIs.

Characterization of Database Systems (2)

- **Authentication & authorization**
  - Identifying users, limiting access based on identity and IP address.
  - Can be applied to operations, schema objects, or some combination.
- **Session**
  - Implicit: each API call is its own session. *stateless* because there is no state connecting one request by a user to the next.
  - Explicit: Can connect a sequence of API calls from one user, providing some guarantees about that sequence, e.g. transaction consistency and caching.
- **Durability**
  - Data survives the application that created it, and can be shared by users of the database.
  - APIs exhibit great variation in the sequence of actions that lead to durability:
    - Immediate durability following a modification
    - Eventual durability following a modification
    - Durability when instructed through a special API call, (e.g. “commit”).
- **Interactions of concurrent sessions**
  - E.g.: If one session is updating X while another is reading X, what happens to each session? Does the reader see the state before or after the update? If two sessions update X concurrently, what happens to each session and what is the final value of X?

Relational Database System

(Common; Implementations include Oracle, Microsoft SQL Server, IBM DB2)

- **Data model:** Relational data model
- **Schema:** A set of tables whose cell contains atomic values
- **APIs:** Native API (e.g. Postgres's libpq), ODBC, JDBC, command-line interface
- **Query language:** SQL. Also, SQL embedded in a procedural language for stored procedures.
- **Authentication and authorization:** ID Based on host operating system identities, or by identities registered with the database, (depending on the system). Authorization based on a model of privileges associated with tables and views (essentially, table expressions).
- **Session:** Provides both stateless (auto-commit mode) and long-lived sessions.
- **Durability:** Immediate for auto-commit, and on commit otherwise.
- **Interactions of concurrent sessions:** ACID transactions, with varying levels of isolation, that can be set per connection.

NoSQL Systems

Simplicity of design, horizontal scaling

- **Data model:** key/value pairs, where the value is often a hierarchically structured document.
- **Schema:** Loosely enforced or absent.
- **APIs:** Key/value operations available in various language APIs; REST APIs are typically native.
- **Query language:** Absent, or simple.
- **Authentication and authorization:** Usually RFC 2617 (HTTP Authentication: Basic and Digest Access Authentication)
- **Session:** Stateless
- **Durability:** Usually synchronous with updates
- **Interactions of concurrent sessions:** N/A (because sessions are stateless)

Search Engines

e.g., Google

- **Data model:** Text documents in a wide variety of formats
- **Schema:** The recognized document formats
- **APIs:** Search requests encoded as URLs
- **Query language**: Some search engines have query languages controlling ranking of documents, significance of various search terms, required proximity of search terms in the document, and other aspects of searching
- **Authentication and authorization:** Various approaches
- **Session:** Stateless
- **Durability:** Document ingestion is done in a "pipeline" of document processing steps. Hard to make general statements about when documents become durable. However, in general, documents are ingested in batches
- **Interactions of concurrent sessions:** N/A

Issues Associated with High-Volume, Complex, Multisourced Data

- Number of Datatypes
- Schema changes
- Data Volume
- Query Complexity
- Query Frequency
- Update Patterns
- Data Contention and Isolation
- System and Database Administration
- ….
- ….

Number of Datatypes

- **Datatypes (more structure to less):**
  - Collections of records of simple values (numbers, short strings, date/time)
  - Collections of hierarchically structured documents
  - Geographic data
  - Collection of text documents
  - Collection of binary formats (e.g. images, audio)
  - *There are off-the-shelf systems for each of these*

- **Big Data apps need multiple data types. Consequences:**
  - Information of different types has to be connected
    - e.g. documents concerning ownership of a plot of land over time. Overlays for utility lines, geologic surveys, etc.
  - Relational databases are well suited for this metadata management.
  - But RDBs are closed environments, and incorporation of various datatypes is difficult. As a result, need to store external identifiers that cannot be managed correctly in all cases, resulting in inconsistencies, lost data, stale data, etc.
  - Interactions among the datatypes results in extremely complex applications.
  - NoSQL systems place fewer constraints on programming needed to deal with external sources.

Schema Changes (1)

- **Applications like to assume a static set of entities, with each entity having a static set of attributes**
- **Big data applications characterized by a constantly growing set of data, potentially with new schema**
- **Changes to the schema are painful because**
  - Applications need to change
  - Possibly large datasets need to adapt
  - Resulting in system downtime
- **Schema Changes:**
  - *New datasets matching existing schema*
    - E.g., DMV vehicle data for a new model year
    - Data cleansing may be needed. (ETL systems can help)
  - *New datasets for existing entities, but with a different schema:*
    - (e.g., new vehicle data contains a new attribute for degree of electric assist)
    - May need to restructure data (ETL)
    - Differences in entity identification are a problem
      - E.g., some states use SSN, others DL#
  - *New entities related to existing entities*: Entity identification problems again
    - E.g. suppose you get new data for vehicles from EZPass in addition to DMV, and schema is different (key not on VIN, but on read license plate, etc.)
  - *Changes to existing schema*
    - New attributes and entities
    - Obsolete attributes and entities
    - Restructuring
    - Change in relationship (e.g. 1:1 -> 1:n)

Schema Changes (2)

Dealing with schema changes

- **Relational approach**
  - Update schema, migrate data to match
  - Fix applications to use the updated definitions
  - Requires huge effort in planning and implementation.
  - Painful for a short time

- **NoSQL**
  - *Schemaless*
  - But this approach still leaves hard problems to applications.
    - E.g., querying Vehicles means that application has to know about all possible versions of Vehicle definition.
  - Painful forever

- **Version the schema? Not an idea that has been applied commercially**
  - We heard anecdotally that it was tried by Israeli Defense Forces for soldier records—each record tied to a particular schema version

Data Volume

TINY → BIGGER → BIGGER → BIGGER → BIGGER

DB fits in memory

- Still write to disk for durability
- Reads don't depend on disk access
- Query speed depends on
- Query complexity
- Indexing
- Etc.

DB doesn't fit in memory

- Memory serves as cache
- Performance drops fast as cache contains smaller & smaller proportions of DB
- Queries I/O Bound
- Best execution plan is the one that makes best use of the cache
- Increasing I/O burden slows queries down more and more

Distribution helps handle issues

Read Replicas

- Asynchronous replication introduced to support read operations that can be satisfied with slightly stale data.
- Minimizes I/O load on the main database.

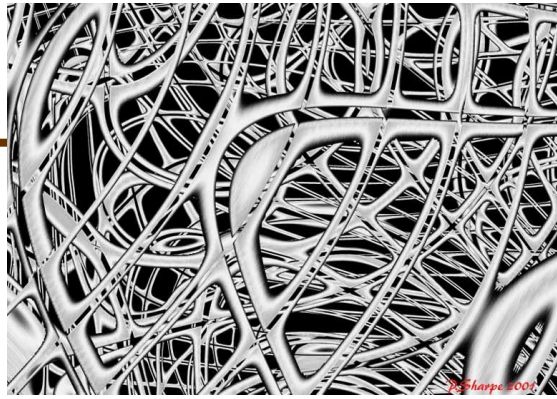
Vertical distribution

- Different data collections on different machines. Only effective if operations tend not to require data on multiple machines
- Otherwise, cost of coordinating data access across machines, especially for transactional updates, cancels out performance gains

Horizontal distribution

- Sharding
- Distributes items in a single data collection across a set of machines.
- As database keeps growing, more shards introduced to handle load.
- Used in NoSQL systems
- Scalable for single writes & single reads
- Complex operations (joins, multiple indexe) do not benefit

Query Complexity (1)



(We use relational terminology here, but comments apply more generally)

- **Key/value lookup is cheap (msec or less), regardless of data volume or use of distribution.**
- **Joins: can be expensive depending on**
  - Data volume (secs, mins, hours)
  - Use of distribution
  - Availability of suitable indexes
- **As joins multiply, can get a lot more expensive; can be exponential**
  - Selection of join order and join algorithms is crucial
  - Difference between a good execution plan and a bad one is orders of magnitude in execution time

- **Aggregation (analysis): needs to scan huge volumes of data**

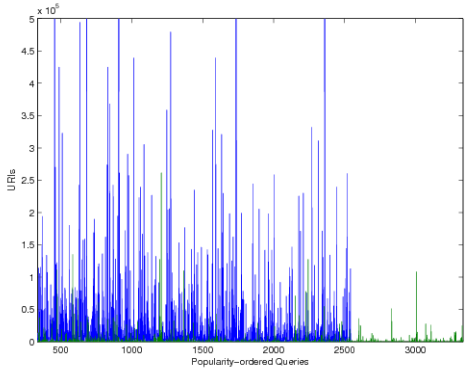
Query Complexity (2)

- **For Big Data applications:**
  - Extreme data volumes drive up execution times for everything but key/value lookup (the simplest queries).
  - As new data sources are imported, queries will have more joins, making good optimization of joins even more important. Probably worthwhile to invest more time evaluating alternative plans.
  - Alternative: Denormalize, but this is still expensive for big databases. May also force applications to change. (Relational views can counteract this to some extent.)

- **NoSQL addresses the low end of the complexity scale:**
  - Good at scalable key/value lookup
  - Good at exploiting parallelism for queries that fit the map/reduce paradigm
  - Anything else needs to be programmed at the application level
  - But more open to dealing with complexity in datatypes, entities outside the database

- **SQL is better at complex queries**
  - Query optimization
  - Rich set of data processing primitives
  - But only applicable within the RDB itself
  - Painful, rudimentary, marginally effective capabilities for dealing with external data sources

Query Frequency



- **Databases supporting web applications tend to have two kinds of queries:**
  - Simple queries relating to a user's action,
    - e.g. login, add to a shopping cart, checkout.
  - Analytic queries
    - Netflix recommendations, website analytics, tracking current trends
- **Frequency of arrival of simple queries is proportional to web site usage.**
- **Analytic queries are run behind the scenes, are more complex and expensive, and run more predictably, based on enterprise requirements.**
- **Big data considerations:**
  - What is the profile of database usage?
  - What has to scale?
  - How predictable are queries?
  - Is it the same set of complex queries over and over? Or are they only mostly the same?

Update Patterns

- **Kinds of updates:**
  - Small transactions:
    - Insert a new entity, update or delete an entity.
  - Bulk import of data into existing collection
  - Bulk import of new collection
  - Bulk update (10% pay cut for everyone)
  - Bulk deletion (remove lots of rows from a collection)
  - Truncation (drop a collection)

- **Relational database utilities can help with many of these**

- **Scalability for small transactions is easy to achieve using NoSQL approaches, or distributed relational databases**

- **Fixed-content (no update) filesystems can handle bulk import in a scalable way**

- **Need to understand BD requirements better to determine if there is a gap between requirements and capabilities of existing systems.**

Data Contention and Isolation

Issues:

- **Are we dealing primarily with data that must be updated in place, or with a situation where updates come in the form of a new import?**
- **What are the requirements regarding visibility of bulk imported data during the import?**
- **Do bulk imports tend to affect existing data?**
  - (E.g., is there related data updated during the import?)
- **How prevalent are updates to existing data?**
- **What is the prevalence of hotspots (area of frequent update, e.g. a counter)?**
- **What sort of isolation is needed for these updates?**
  - **Tradeoff: concurrency vs. anomalies**

System and Database Administration

- **Issues:**
  - Backup/restore
  - Storage management
  - Replication

- **Consequences for big data:**
  - Backup/restore may be infeasible.
    - Some very large archives rely on replication to a second site for backup.
  - If many different component systems are used (databases, search engine, filesystems):
    - Each has is own system administration tasks
    - They all have to replicated (or otherwise backed up)
    - Any way to unify all this maintenance?

Overall—New Directions (1)

- **Many Big Data applications have**
  - Large volumes of data of multiple datatypes
  - The need to integrate new sources of data
- **Applications must integrate data across these collections**
- **There is currently no technology that can universally achieve this**
  - Individual databases can scale for simple queries
  - Or they can perform complex queries but not scale
  - Integration of databases, even within a data model, is difficult
  - No systematic approach for integration across data models
  - So querying is a matter of laborious application development
- **It is often stated that Big Data is not tractable using conventional relational database technology**
- **But Relational database technology – *concepts* if not current products – may still be the right starting point for building database systems that can support the performance, scalability, and integration demands of Big Data applications**

Overall—New Directions (2)

Reasons not to discount relational concepts, and new directions:

- **Language: SQL has proven the feasibility and power of non-procedural query languages in the context of the relational model.**
  - Very rich paradigm for application development, so there has been little impetus to take a general approach to extension and integration
  - Customers have gotten used to doing this integration on their own, at the application layer
  - But new requirements call for the steady incorporation of or integration with new data sources and databases, and solving this problem at the application layer doesn't look like a good long term solution
  - It is therefore necessary to either think about extending SQL and the relational model to meet this challenge, or to start developing a new non-procedural language.
  - There have been a few attempts to extend SQL: with Google-style search capabilities; with GISs; with filesystems, but it is to soon to tell if this is the right way to proceed
- **Query processing: SQL has succeeded because non-procedural queries can be reliably compiled into efficient execution plans.**
  - Query engines and optimizers of relational database products responsible for widespread adoption of SQL
  - Because SQL is standardized, developments in query processing technology have been focused on SQL and the relational model
  - If a future version of SQL, promoting better integration, is to succeed, then query processing technology will again be the determining factor
  - To date, any ability to query external APIs or data sources has been tacked on, repurposing existing mechanisms.
  - Need more research on how query processing would be accomplished if interactions with external sources were a top-level design goal

Overall—New Directions (3)

- **Data integration:**
  - Many ETL systems, data cleaning suites, and ad hoc applications have been built for the purpose of combining data sources that share a data model but implement different schemas.
  - As a result, the mechanics of importing data are often not too difficult, but specifying how the import should be done is difficult.
  - Recurring problems include resolving identity, resolving differences in representation (units, encodings, etc.), cleaning up data, and handling data discovered to be in error in the process of integration

- **Scalability:**
  - Shared-nothing, sharded architectures achieve scalability only for the very simplest queries and updates. Query optimization research in distributed systems is mostly about performance, not scalability. It is currently unclear how to obtain scalability for complex queries.

- **Transaction management:**
  - Research in this area needs to start with the application area. If the major database updates are the incorporation of new data sources, does that lead to simplifications? How common are concurrent updates to the same data? What sort of isolation is required?

Overall—New Directions (4)

Three broad avenues to support the creation of big data applications in the future:

(1) Continue to leave application developers to do the required integration of high-volume, dynamic, disparate and heterogeneous data sources themselves, repeatedly facing the same difficult implementation issues

- Ignore the problem?

(2) Radically rethink the entire problem space, developing entirely new, still unimagined approaches from scratch

- May end up being necessary
- But not a first resort
- May amount to throwing out a lot of sunk effort

(3) Use the data modeling, language, and implementation approaches taken by relational database systems as a starting point for supporting big data.

- A logical starting point