Section 1: Week 1: Database Bibliography

Nate Bachmeier

TIM-8120: Distributed Systems

December 8th, 2019

North Central University

Bibliography

The growth of data creation from sources such as IoT, Cloud, Big Data, and Mobile (ICBM) is increasing at an exponential pace. This explosive volume of information is forming in different shapes, with varying degrees of structure. Traditional database patterns and practices are unable to manage these data sets efficiently, which is driving enterprise environments to invest in new technologies. Merely adding new widgets to the network topology will not solve the challenges, and existing business processes will also need a revision. Through a combination of these ideas, enterprises can evolve their data pipelines and unlock the insights into more agile data-driven decisions.

# Business Intelligence Tomorrow (2019)

To understand the landscape of future Business Intelligence systems, one needs to look at the challenges of today (Harper, 2019). Harper proposes investment areas into (1) metadata management, semantic understanding, data catalog, data modeling, (2) Natural Language Processing, and (3) Edge computing.

The first aspect deals with the Data Lifecycle Management of ICBM data and its operationalization. Storage prices have decreased significantly, which has led to numerous businesses collecting vast pools of unstructured dark data. One of the principal inhabitants for these businesses is a lack of Information Governance, which includes classification, controls, identification, and monitoring (Ajis & Baharin, 2019). As these organizational systems improve the usability and discoverability of specific data subsets, then data scientists can begin exploring the data and coming up with operational insights.

The second aspect deals with the interaction of users into the data management system. Many of these data repositories rely on query languages, such as Structured Query Language (SQL), to store and retrieve records. These languages introduce a barrier to entry challenges for users of the system, as they need to learn tedious syntax. Instead, Natural Language Processing (NLP) can convert business questions directly into data-driven solutions. When the NLP algorithm is made aware of proprietary object models, it can derive entities, verbs, and other relationships. These capabilities led to a democratization of self-service scenarios across all levels of the organization.

The third aspect is the inclusion of edge computing in data processing architectures. Micro-clouds of IoT and mobile devices are generating vast collections of sensor and machine-to-machine data. Centrally processing these feeds could involve significant network I/O, or is economically prohibitive to move. Instead, a transformation of these ‘high volume/low quality’ feeds into ‘low volume/high quality’ aggregations needs to take place. For instance, one hundred temperature sensors could report one hundred individual measurements or the median of their aggregate value.

# Big Data Quality: A Survey (2018)

Big data is high-volume, high-velocity, and high-variety information that can produce high-value, assuming it's high-quality (Taleb et al., 2018). According to ISO 25012, the critical measurement of quality is its fitness for us. Taleb et al. measure this fitness against intrinsic, contextual, accessibility, and representational dimensions. They propose a Process-Driven Analytic Pipeline that contains multiple quality gates. Each quality gate further refines the information into a more standardized and curated representation of itself. For instance, unstructured data becomes annotated, reformatted, and entities extracted. This curation process improves the consumption of data into analytical solutions, enabling more precise estimations.

Their approach is similar to the National Institute of Standards and Technology (NIST) reference model for data lifecycle management, which includes phases Collection, Preparation, Analysis, and Action (Mazumdar, Seybold, Kritikos, & Verginadis, 2019). NIST also proposes spending effort upfront to clean data sets. For instance, extreme outliers might skew the prediction results, and by pre-emptively removing them, a more accurate model defined.

# Business Intelligence for Enterprise Systems: A Survey (2017)

After data curation, experts use machine learning to perform statistical analysis (Duan & Xy, 2017). These algorithms come in two forms, supervised and unsupervised, based on the availability of the predicted value. Supervised algorithms, such as linear regression and decision trees, use examples to build an inference model. Unsupervised algorithms, such as itemset mining and k-clustering, seek out correlations within groups of similar records. In either scenario, the data become split for parameter training (70%) and accuracy validation (30%). Then the model is exposed through RESTful endpoints and consumed by other enterprise applications. Through the use of web service abstraction layers, a clear separation of duties occurs, such that development teams can leverage the predictive capabilities without specialized data science training.

Duan and Xy also describe future challenges for business intelligence in the enterprise environment. They call out the need for outlier detection, management of graph data, and a strategy for edge computation. Graph databases enable the discovery of complex relationships that would not be possible in traditional relational stores (Kronmueller, Chang, Hu, & and Desoky, 2018). In a conventional relational store, graph data is tedious to store, because the table structures are globally indexed. Meanwhile, native graph stores can maintain local indexes for the edges to quickly traverse to associated nodes.

# A Survey on Graph Database Management Techniques for Huge Unstructured Data (2018)

Many solutions have attempted to provide graph abstractions on top of relational databases or other traditional technologies. These solutions tend to be inefficient due to the overhead of layering technology (Patil, Kiran, Kavya, & Naresh, 2018). Patil et al. also point to adoption challenges caused in-memory storage models limiting the scalability and inconsistency of programming interfaces. They acknowledge that Gremlin and SPARQL are gaining consensus as to the leading query languages, but have a long road to industry standard tooling.

Next, the authors reviewed seven different native graph databases and compared their strengths and weaknesses. Patil et al. claim that Neo4j is the most popular graph database in enterprises. It uses a proprietary language, called *Cypher,* to calculate statistics, traverse relationships, and find bipartite and connected components (Kronmuller, Chang, Hu, & Desoky, 2018). Kronmuller et al. would agree with Patil et al. that *Cypher* is easy to learn and apply to real-world problems.

Another technology reviewed is the Titan Graph, which provides a consistent abstraction layer across different NoSQL stores. Administrators can host a Rexster service to expose RESTful interfaces that accept Gremlin queries. Similar to Cypher, Gremlin can perform many relationships and path operations, such as the shortest path. Titan typically accompanies Apache TinkerPop, a Java-based implementation that can scale to billions of nodes across clusters of horizontal-scaled servers.

Patil et al. conclude their discussion with a review of active research topics within Graph Database research, such as online compression, feature mining, subgraph comparisons, and biology applications. Biology research uses these systems for representing protein-protein interfaces, which can contain enormous relationships between different treatments. Until recently, it has been challenging for smaller organizations to manage big data graphs due to cost restrictions. However, now that public cloud providers are natively offering on-demand pay-for-use economics, they are becoming available to businesses of all shapes and sizes.

# Data Storage Management in Cloud Environments: Taxonomy and Direction (2017)

Storage as a Service (StaaS) is enabling businesses to make agile use of highly reliable, scalable, and flexible resource pools (Mansouri, Nadjaran, & Buyya, 2017). According to the CAP theorem, a distributed store can't provide more than two guarantees between consistency, availability, and partition tolerance (Gilbert & Lynch, 2002). This constraint causes network administrators to choose between different optimization points offered by Relational (SQL), Not Only SQL (NoSQL), and NewSQL solutions. Relational stores focus on consistency over availability, in contrast to NoSQL, which targets availability through eventual consistency models. Between these two extremes is NewSQL, which provides ACID transactions across a storage partition, not the entire data set. Another optimization dimension comes from the trade-off of Online Transaction Processing (OLTP) versus Online Analytic Processing (OLAP). OLTP needs to handle updates occurring many times per second, compared to OLAP, which is predominately read-only. As the number of changes decreases, so does the number of entity-level locks that block concurrent operations.

Mansouri et al. state that business intelligence comes as structured, semi-structured, and unstructured formats. A semi-structured entity might use a standardized envelope and then application-specific extensions (e.g., property bags). Unstructured objects are opaque byte strings such as audio and video. To hold these objects, business intelligence stores need to expose specialized abstractions, such as block and file access.

# Data Lakes and Data Warehouses, Working Tandom (2019)

Enterprises environments are adopting data lakes as part of their data management workflows. One survey estimated that 38% of organizations had already provisioned an instance, and another 15% are evaluating rollout strategies (McKendrick, 2019). This new technology raises questions about the positioning of data lakes against traditional OLAP data warehouses. According to McKendrick, a data lake strategy should complement, not replace, the enterprise data warehouse. The distinction comes from the storage abstraction layer, with lakes optimized for unstructured and semi-structured content versus warehouses requiring highly structured content. The literature refers to this difference as ‘schema on reading versus schema on writing patterns.’

According to Garda, a complementary strategy might start with all data sources flowing into the data lake. Next, information governance policies need to classify and annotate the data for discoverability and exploration. Then data processing workflows can *extract* aspects of the unstructured records, *transform* into schematized facts, and *load* into the data warehouse. Afterward, business professionals can use familiar tooling and query languages to examine Key Performance Indicators (KPIs) and other aggregate data (Garda, 2019).

# Building novel capabilities to enable business intelligence agility (2018)

A challenge with this approach is the data warehouse can only address previously stated business questions. To onboard new decision models, requires engineering staff to configure custom ETL workflows. This additional work reduces agility and the organization’s ability to pivot towards dynamic market opportunities (Harper, 2019). One solution is to duplicate a subset of the lake into in-memory databases, as they are more responsive to exploration (Knabke & Olbrich, 2018). Technologies such as ElasticSearch can efficiently hold vast datasets of unstructured documents. The store creates a reverse index to associate terms (values) with specific records and can fetch those items in near real-time. User’s of the system issue queries through an Apache Lucene interface, in the form of keywords and Boolean modifiers. For instance, to search for ‘Java and not coffee’ would become ‘java -coffee.’ This simplified syntax has a low barrier to entry and promotes self-service scenarios.

It can be expensive to operate large scale in-memory systems at scale, and this leads to information governance questions. Many organizations choose to only retain a few days or weeks of information in these memory stores. Some implementations support a tiered storage architecture that pages the Least Recently Used (LRU) objects into slower and cheaper storage. However, it can be confusing for the end-user to determine if hot or cold storage will service the query. These inconsistent performance characteristics can degrade the user experience and discourage broader adoption.

Alternatively, the process that loads from the data lake into in-memory databases can (1) register the source location inside of a data catalog and (2) annotate the in-memory items to contain the data catalog address (Zambetti, Pinto, & Pezzotta, 2019). This procedural change extends the exploration into more advanced scenarios without sacrificing performance consistency. Others have suggested using encoding and compression schemes to improve the efficiency of resource utilization (Taleb, Serhani, & Dessouli, 2018). While there are strengths to both technical and procedural changes, reviewing the information governance policy and removing specific content might be more straightforward.

# Bringing SQL databases to key-based NoSQL databases (2019)

The disruption of ICBM data requires businesses to rethink their approach towards scalability with a transition from traditional Relational stores to NoSQL. These new systems leverage distinct access patterns that introduce a learning curve for engineering teams (Schreiner, Duarte, & Santos Mello, 2019). Schreiner et al. present a framework that is capable of automatically translating specific queries at runtime, minimizing the modifications to existing source code. They accomplish this through a canonical model that maps the hierarchical document structure to the legacy table schema. These models drive an algebraic decomposition process that converts the SQL statement into various primitive operations. These primitives perform Create, Read, Update, and Delete (CRUD) actions, along with specific table join scenarios.

Schreiner et al. provide performance comparisons of their tooling against alternative implementations. Their solution was faster, though aspects of the test can be misleading. For instance, Amazon’s SimpleDB is slower due to not locally caching the schema and doubling network transactions. Also, their dataset could use Mongo’s BulkGetItems function, which is a *product-specific* feature in a *product-agnostic* framework. Other configurations could have different performance characteristics and not be representative of these specific results. Despite these nuances, their tool is an impressive innovation that removes a tight dependency from the storage technology stack.

# Document-Oriented Data Schema for Relational Migration to NoSQL (2017)

Schreiner et al. discussed the importance of constructing the canonical model, though they did not provide any guidance on best-practices or process. Hamouda et al. provide various ‘rules of thumb’ that engineers can follow during the storage conversion. They state that the critical step is to identify the hierarchical structure and entity boundaries. Relational storage models blur these lines through table normalization. However, join operations do not natively exist in NoSQL implementations, so these dimensional aspects need to become part of the native entity document (Hamouda & Zainol, 2017). For instance, a relation called *students* references a *demographic* table. A document-oriented schema would collapse these properties into a child array of the student document.

The article continues with a combination of Unified Modeling Language (UML) and Entity-Relationship diagrams. While they covered many core competencies, there was little discussion around many-to-many relationships. Consider the model for *students* and their *class memberships* and the challenges around these associations. In these scenarios, a naïve solution would be to repeat portions of the dataset inside of both objects. However, this approach can be complex to propagate changes reliably across multiple objects. Specific systems, such as Amazon’s DynamoDB, offer a transactional commit mode that encapsulates updating groups of records at the same time. There are limitations to transactions across NoSQL and NewSQL systems, such as alignment at partition boundaries (e.g., Microsoft SQL Azure) or small collections of items within the operation (e.g., Amazon’s DynamoDB) (Mansouri, Nadjaran, & Buyya, 2017).

# Implementation of a performance-optimized database join operation on FPGA-GPU (2017)

A relational join attempts to combine two or more data sets using a par