Section 4: Week 7: How I Learned to Stop Worrying and Love Timeseries

Nate Bachmeier

TIM-7020: Database and Business Intelligence

January 19th, 2020

North Central University

How I Learned to Stop Worrying and Love Timeseries

## Background

Data volumes continue to grow exponentially on an annual basis because of IoT, Cloud, Big Data, and Mobile (ICBM). This massive emersion of records introduces challenges for organizations as they need to gain timely insights into their data. Consider the raw resource costs associated with the data management lifecycle in terms of computing, storage, and networking. For instance, the Hadron collider generates 300GB/s, making an hour of sensor data approximately 8.6Pb (Basanta-Val et al., 2017)! Other examples exist across health care (e.g., medical monitoring devices), finance (e.g., high-frequency trading), retail (e.g., click streams), and manufacturing (e.g., industrial IoT) sectors as users and devices produce these time-series data streams. Traditional big data systems addressed these issues by staging data into an object or NoSQL store, then executing batch processing models such as Map-Reduce (Barika & Garg, 2019). While this approach works today, it will continue to become economically prohibitive going forward. Instead, the programming models need to evolve towards real-time stream processing that extracts and store subsets from the feed. As the programming model changes to iterative processing paradigms, it will cascade across the technology stack, fundamentally changing downstream interactions with data management systems. For instance, training deep learning models relies on offline batch processing versus real-time only systems that will require iterative learning algorithms (Bosch, Olsson, & Brinne, 2019) (Yang et al., 2019). Other changes will take place across infrastructure and business continuity monitoring, as time-series streams are highly contextual—traffic to the eCommerce site is 1000% higher than normal, are we under attack, or did our marketing campaign go viral? There are numerous other scenarios impacted, all of which increase agility for organizations towards dynamic market conditions.

## Problem Statement

The adoption of big data started to pick up steam around 2012 (Al-Sai, Abdullah, & Husin, 2019), and shortly aftward the industry needed to transition from relational to NoSQL technologies. This data management evolution needed to happen because the programming models could no longer meet the business needs through transactional patterns, and instead pivoted towards MapReduce solutions. The efficiency challenges with MapReduce and similar batch systems comes from their blunt instrument design. An analogy exists with “counting the number of red cars in a parking structure.” A batch processing approach would hire a dozen workers and comb through each lane to report back the aggregate sum. While this strategy works for the local mall, it becomes unrealistic at Disney World. Alternatively, a stream processing approach to the problem would place a couple of attendants at each gate that modifies a local counter. The use of real-time counters is more efficient in terms of computation, storage, and communication overhead—because of the iteratively updating model that does not need to retain the source information (the car in the lot).

Similarly, the accelerating growth of high-resolution time-series data continues to drive the need for innovation into shorter retention systems that deliver the same value. These changes to the programming model will cascade into the data management system and, therefore, into the requirements of downstream consumers, such as machine learning, business continuity, enterprise resource management, and safety systems. A clear understanding of these ramifications need to take place so that compensation strategies formed to remediate deficiencies across people (e.g., training), technology (e.g., tooling), organization structure (e.g., political resistance), process (e.g., governance), and data management (e.g., security and compliance) (Al-Sai, Abdullah, & Husin, 2019).

## Relevance and Significance

Between 2013 to 2020, the world’s data has grown from 4.4 to 44ZB and is showing no signs of slowing down (Mansouri, Nadjaran, & Buyya, 2017). This emergence of big data allows organizations to become more agile and respond to highly dynamic market conditions (Knabke & Olbrich, 2018), often at the expense of competitors with immature data management strategies (Al-Sai, Abdullah, & Husin, 2019). However, as these businesses continue down this path, the volume, variety, and veracity associated with integrating across multiple data sources become economically prohibitive to gain timely insights. The cost of sensors is also decreasing at an exponential rate, which is pushing industries to measure everything models. Consider a network of restaurant supply chains that can use cloud and IoT platforms to easily monitor and predict across ordering, transporting, preparation, and customer satisfaction —these sensors report into business intelligence systems everything from temperature anomalies to the performance of wait staff (McCrea, 2019) (Ma et al., 2018). Looking toward the forefront with Industrial IoT, many modern manufacturing plants are already becoming overwhelmed with the complexity of managing these time series streams (Frodigh, 2018).

So, (1) if all competitive organizations need to adopt big data strategies, such as those created by managing time series sensors, and (2) those complications are quickening every business cycle, then (3) finding solutions to those challenges is both highly relevant and significant to a broad audience. One paradigm shift that addresses these needs is real-time streaming data management solutions, as these technologies focus on iteratively processing and reduce the reliance on batch processing across external storage networks (e.g., SAN, data lakes, and data warehouses). That is not to say, data lakes and warehouses seize to exist more likely stream data management acts as an essential precursor working in tandem (McKendrick, 2019).

# Literature Review

## Big Data Impacts and Challenges (2019)

According to Al-Sai et al. (2019), most industries have already adopted some form of big data strategy and are using it to create business value, typically by reviewing customer behaviors and improving marketing campaigns. Less successful companies are only storing telemetry and have yet to define how it aligns with their business goals. Goznales and Wareham (2019) would agree, that unfortunately, business intelligence cannot be gained through osmosis and needs a conscience plan with a concrete list of questions that need to address. Other businesses understand their questions but encounter external limitations such as insufficient training, high storage and tooling costs, political resistance, and compliance requirements. Some of these limitations are self-imposed and require changes at the leadership level, while others demonstrate a lack of awareness across state-of-the-art technologies that are both open-source and public cloud-based. In either scenario, the business needs to define what big data means to them and how it will enhance their competitiveness.

## Data Management Challenges for Deep Learning (2019)

If data is the new oil, then deep learning would be the engine that transforms these data points into decision processes. Bosch et al. interviewed experts across six different industries to assess the challenges associated with building predictive models by those in the field. They found that deep learning, a machine learning strategy where the compute discovers the patterns, exists across scenarios such as product recommendations, competitively pricing wind power, appraising housing, cancerous cell classification, and credit card fraud. In each use-case, having the data in the right shape, size, granularity, and accuracy is overly difficult as many problems need to span heterogeneous providers.

## Orchestrating Big Data Analysis in the Cloud (2019)

Providing data in the right shapes is “very different from traditional scientific workflows as this [requires] continuously processing heterogeneous sources and support multiple analytical tasks (Barika & Garg, 2019).” Barika and Garg provide a taxonomy of different programming models that frequently appear in big data platforms. These strategies target different paradigms such as operating across static files (e.g., Map Reduce and Hadoop), unbound sets (e.g., stream processing and Apache Storm), message queuing (e.g., Amazon Kinesis), hybrid systems (e.g., Google Dataflow and Apache Spark), micro batches (e.g., Lambda), and append-only sequences (e.g., Kappa-based). For many workloads, multiple technologies need to come together, as heterogeneous cloud systems are becoming a normal occurrence.

Consider an auto manufacturer that has factories geographically distributed across the Americas, Europe, and Asia. Each of these factories generates enormous amounts of time series data, more than would make financial sense to centralize. In these scenarios, edge processing needs to aggregate and reduce the volume to a manageable amount for centralized analytics. After selecting that subset orchestrating that data movement efficiently and reliably across multiple provider network is an open problem (Barika & Garg, 2019, p. 34). Large aspects of these challenges arise from the connecting resources across multiple vendors, such as public cloud providers (e.g., Amazon or Azure) with proprietary solutions (e.g., IBM or HP) or open-source technologies (e.g., OpenStack). Businesses would prefer to use only one consistent technology, though in many cases, this is not possible due to latency requirements (e.g., IIoT safety systems), previous investments into legacy systems (e.g., mainframes and VMWare farms), compliance constraints (e.g., HIPPA), or incompatible network protocols (e.g., multi-cast).

## Patterns for Distributed Real-Time Stream Processing (2017)

As big data crosses a boundary, either into or out of a local cloud, there is an opportunity to reshape that feed through stream processing dynamically. For instance, the Hadron collider uses stream processing to sample a 300GB/s sensor feed down to a more manageable size of 300MB/s before offline processing the static files through map-reduce (Basanta-Val et al., 2017). Network administrators can use either distributed stream processing (e.g., microservice connected subsystems) or realtime parallel computing (e.g., OpenMPI or fork/join primitives). Both scenarios offer trade-offs in terms of design simplicity and scalability, though neither strategy is inherently simple and forces engineers to about low-level primitives.

For instance, Apache Storm uses a series of spouts (event emitter) and bolts (transformer) to interweave into a topology (distributed application). Under many scenarios, bolts act as combiners and filters, which is analogous to JOIN and WHERE clause in traditional SQL programming languages. Both Hamouda et al. (2017) and Schreiner et al. (2019) investigated adoption into query engines that did not leverage SQL syntax and found the learning curve to be a strong deterrent against broader usages. Instead, a domain-specific language could exist to model the distributed application as SQL-like queries, and at compile-time, generate the relevant sub-processing units. Another critical advantage of such a solution is that optimizations to the processing network could occur without introducing an additional burden on the developer.

## End-to-End Time Architecture for Analyzing and Clustering Timeseries Data (2018)

Talei et al. (2018) describe a systems architecture that electrical microgrids can use to self-regulate. Their solution centers around a semantic model of their college campus and specifies the relationship between classrooms and their buildings. Various sensor devices hydrate the model by emitting metrics (e.g., temperature, humidity, and lumens) into an open-source Kaa IoT cluster. Kaa acts as a rule engine that can publish into Apache Kafka topics that route sensor data to different consumers (e.g., individual students, faculty, and proprietary expert systems). One of these expert systems is an Apache Spark cluster that uses MLib to run classification (e.g., room state), regression (e.g., anomaly detection of sensor values), clustering (e.g., the commonality between rooms), graph analysis (e.g., differences of first and second-floor rooms), and unsupervised event association discovery (e.g., event A followed by B). The results of these stream processes flow into a reporting and control system used by the facilities team.

## The future of FinTech (2019)

The learnings from Al Akhawayn University demonstrate how several heterogeneous IoT data streams can be reshaped dynamically and then flow into the various decoupled private clouds (e.g., facilities reporting system versus student projects). These same concepts apply to numerous scenarios across the financial technology sector (FinTech). According to Das (2019), FinTech systems need to model the flow of capital markets as a mechanism to measure risk and provide financial services (e.g., investment recommendations, payment routing, fraud detection) to their customers. Many of these systems, such as high-frequency trading and market watch-dogs, need to analyze tick-by-tick every action that occurs by all participants—which introduces aggressive caching management solutions within the distributed stream processing network.

## ChronicleDB: A High-Performance Event Store (2019)

Das discusses many of the system requirements and use-cases for event systems but does not drill into the technical requirements to build such a system. Seidemann et al. (2019) discuss the inverse position with the technical write up of ChronicleDB, an event store for time-series scenarios. The database is capable of processing billions of objects on a single machine due to design principals that center around the append-only log is the system of record. They note that many similar technologies attempt to maintain both a transaction log and current state of the system view, though, for time-series big data, this is redundant and wasteful. Instead, they optimize the file structures for sequential read and write across 8mb storage blocks. When out-of-order events occur, they attempt to fix them within memory queues or relying on pointer manipulation to their Temporal Aggregated B+-tree (TAB+-tree). Their solution demonstrates methods to increase stream retention while minimizing overhead.

## TICC of Multivariate Time Series Data (2017)

Another

## Adaptive Deep Learning for Incremental Learning (2019)

Improving deep learning using shorter retentions