Clinical Decision Support at Scale: Making Fast Decisions on Big Data

Angelo Kastroulis, ALM
Optum, Raleigh, NC

Abstract

We introduce a Scala-based execution engine for Clinical Quality Language (CQL) that leverages the scalability of Apache Kafka for fast computations on arbitrarily large sets of data. We discuss several key components to the system; the partitioner, the interactive query store, the storage format, and the computational engine that compiles and executes CQL rules.

Introduction

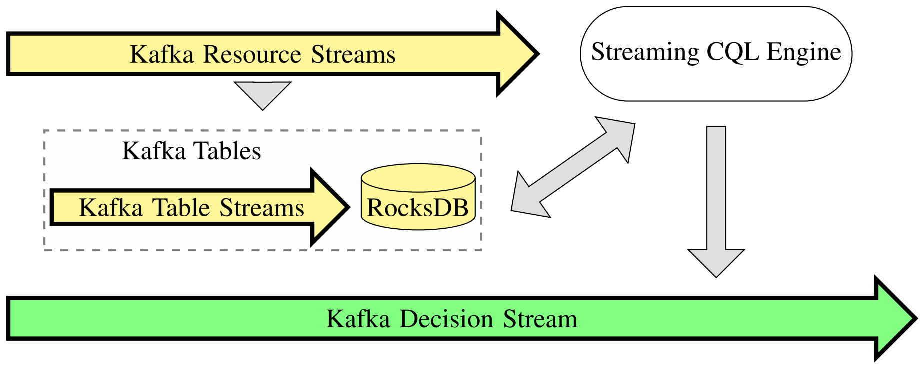
Clinical decision support (CDS) at the time of care requires extremely low latency rule execution. In the world of Big Data, this is particularly challenging. Big Data has been characterized in terms of 4 Vs–volume, velocity, variety, and veracity[1]. Many solutions exist for executing rules in terms of conventional computer science (e.g., Drools and DMN). Health IT-specific methods of encoding rules such as Clinical Quality Language (CQL) statements have also emerged. These solutions still suffer from the difficulty in balancing data velocity and volume.

Rule-based approaches (such as RETE) require that rules and data fit into working memory. As rule sets and data increase, the amount of memory required increases disproportionately (referred to as “memory explosion”)[2]. Eventually, the working set becomes sufficiently large, and the system breaks down. Similarly, small, fast scripting interpreters fail when data sets become exceedingly large[3, 4].

Technologies in the Big Data ecosystem such as Spark, Kafka, and Cassandra execute computations on large sets of data by partitioning the data and moving the computation to the data[3, 4]. Data movement has become the new bottleneck, and ongoing research shows promising techniques to improve query performance [5].

Key Contributions

CQL has emerged as a widely adopted standard for Quality Measure computation, but also is gaining momentum in clinical decision support. In this discussion, we explore a CQL execution engine that leverages Apache Kafka to scale fast computations on arbitrarily large sets of data. We discuss several key components to the system; the *partitioner*, the *interactive query store*, the *storage format*, and the *computational engine*.



Data Partitioning The unit of scalability in Kafka is the partition[3]. A unit of execution is divided among partitions. Many partitions may exist on a single server, or they could be spread horizontally among servers. Therefore, to execute clinical rules effectively (avoiding cross-network shuffling), we implement a custom partitioning scheme to guarantee patient data remains on the same partitions on each machine.

Interactive Queries Querying data requires carefully balancing the decision boundary of index probing and scanning[5]; therefore, a performant system must seek to minimize data movement.

RocksDB is a fast, durable key-value store often used as a backing store for various technologies such as Redis-on-flash [6] and Kafka's interactive query engine. Kafka uses RocksDB to index and arrange data on the server, while the underlying stream manages the partitions across the network. Since the partitioner has already co-located a patient's data, we can index and make it available for interactive queries. We implement partial key range queries so that a patient's resources are located sequentially and retrieved quickly. Querying the patient portion of a composite key composed of the patient's identifier and the resource's identifier allows efficient range queries.

Storage Format Binary formats for serialization significantly outperform JSON and other representations of data[7]. Apache AVRO provides a convenient method for schema exchange and description (even along with the message), while the data itself is in a compact binary format. Such a scheme provides an optimal mix of performance and schema management. We use AVRO versions of FHIR templates to improve serialization/deserialization performance over other FHIR-based schemes (XML and JSON).

CQL Execution Engine Within the engine, itself, we introduce a Scala-based execution engine that leverages Kafka internals optimized for computational performance. It includes the capability to compile rules to native jar files rather than reading ELM XML. The execution engine can be leveraged separately for smaller workloads and performs well compared to other open-sourced solutions (such as OpenCDS).

The engine's parser subsystem converts CQL or ELM into a compact Abstract Syntax Tree, compiling that to a jar file. The execution subsystem executes the rules based on a pluggable architecture accommodating various data sources and formats (such as Kafka and AVRO). While we implement CQL, the engine could execute any arbitrary representation, such as DMN, with the inclusion of an appropriate parser.

Conclusion

The presented approach shows significant improvement in the performance of the execution of CQL on individual workloads, but also offers fine-grained control over how to scale the system over datasets of various sizes by leveraging the horizontal scalability and mechanisms of Apache Kafka. The novel combination of health standards and proven technologies for Big Data allow each to bring their respective strengths to the design.

References

1. Bellazzi R. Big data and biomedical informatics: a challenging opportunity. Yearbook of medical informatics. 2014;23(01):08–13.
2. Chen Y, Bordbar B. Dress: A rule engine on spark for event stream processing. In: Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies; 2016. p. 46–51.
3. Kreps J, Narkhede N, Rao J, et al. Kafka: A distributed messaging system for log processing. In: Proceedings of the NetDB. vol. 11; 2011. p. 1–7.
4. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I, et al. Spark: Cluster computing with working sets. HotCloud. 2010;10(10-10):95.
5. Kastroulis A. Towards Learned Access Path Selection: Using Artificial Intelligence to Determine the Decision Boundary of Scan vs Index Probes in Data Systems. Harvard University; 2019.
6. Ouaknine K, Agra O, Guz Z. Optimization of rocksdb for redis on flash. In: Proceedings of the International Conference on Compute and Data Analysis; 2017. p. 155–161.
7. Maeda K. Performance evaluation of object serialization libraries in XML, JSON and binary formats. In: 2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP). IEEE; 2012. p. 177–182.