Baleen Analytics

PAT HELLAND

**LARGE-SCALE
FILTERING OF
DATA PROVIDES
SERENDIPITOUS
SURPRISES**

Baleen is a substance found in the mouths of the largest whales in the ocean. Similar to fingernails in its construction, baleen is used as a giant filter that allows these whales to scoop up vast amounts of seawater, while keeping the nutritious krill and other sea life as food. Krill are small crustaceans similar to shrimp. I think of them as “shrimpy” shrimp, yet they provide a large portion of the food consumed by the largest animals on earth.

Like these whales, data analytics is increasingly hoovering up anything it can find without regard to shape, form, or schema. By ingesting anything and everything, oblivious to its provenance or hygiene, we are finding patterns and insights that weren’t available before. This is greatly accommodated by keeping the data in JSON, XML, Avro, or other semi-structured forms.

This article explores the implications of this “late-bound” schema for both data analytics and for messaging between services and microservices. It seems that a pretty good understanding among many different sources allows more

*It's Not Your Grandmother's Database Anymore

flexibility and interconnectivity. Increasingly, flexibility dominates perfection.

DEVOLVING TO THE FUTURE

In the mid-'90s, I helped build the architecture for the world's first transactional N-tier application platform. It was a fantastic team and one of the highlights of my career. Application developers created components and interfaces to them. Components were deployed across multiple servers with dynamic activation and load balancing as work flowed across clients, servers, and multiple databases with well-defined transactional behavior. RPCs (remote procedure calls) were based on the method calls in the interfaces, which were defined in advance.

We had many raucous and lively debates about the tradeoffs between early- and late-bound interfaces. *Early-bound* interfaces are defined in advance. They specify the set of methods, access-control policies, and all that is needed to invoke a persnickety service that must have everything perfect to process a remote method call. Early-bound interfaces are kept in some form of shared repository where they may be sucked up by potential clients that will ensure perfect compliance. In many ways, this mirrored how components are used to compose monolithic single-process applications. Many of us thought that early binding was the ultimate evolutionary step for interfaces, providing the most intimate semantics for interchange.

In contrast, *late-bound* interfaces are far more loosey-goosey. A target service may document the approximate

In many ways,
the difference
between early-
and late-bound
interfaces
is one of vision
and goals.

formats required in the semi-structured messages sent to a service. The service will take this information and figure out what it can, well, figure out. Usually, this works pretty well. Frequently, the target service is gradually enhanced to do a better job of scratching its head to make the most of the barely understandable garbage it receives from callers. Both SOAP (Simple Object Access Protocol) in its earliest incarnation and REST (Representational State Transformation) are examples of late-bound interface mechanisms.

In many ways, the difference between early- and late-bound interfaces is one of vision and goals. Early-bound offers a crispness and clarity of purpose. You are much more confident of the semantic tightness of the caller and the callee. This makes a lot of sense in a relatively small world where the combinatorics of interaction are relatively small; it can assist in debugging the interaction of the two components. On the other hand, when the set of called and calling components gets large, this same semantic tightness causes big challenges as components evolve over time. When the called and calling components are in different departments or even companies, the friction imposed on changes can be overwhelming.

Early distributed systems were evolving forward from single-server mainframes and minicomputers. The vision was an extension of the centralized machine to leverage the compute capabilities of multiple computers in a single data center under a single trust domain for one organization. Over the years, this has evolved toward a cacophony of loosely coupled services in different trust

environments, each of which is moving forward at its own pace. *The ability to successfully interoperate has proven to be more valuable than crisp and clear semantics as components work together.* While I didn't see this coming at first, it is obvious in the rear-view mirror.

SQL AND ITS DDL

SQL is the dominant relational database language. Relational systems are brilliant, powerful, and used to hold and process a great deal of essential data, especially the data used to run companies. Relational algebra (the theoretical backbone of SQL) needs to have its schema well defined to describe how the data is structured into tables, rows, and columns. The schema is captured with DDL (Data Definition Language) in the database.

SQL's schema is early-bound. It is declared and may be transactionally updated. If the DDL is changed in the database using a transaction, a new DDL is used to define the behavior of queries after the change. The old DDL was in effect before the change. At any point in time, there is a single structure for the DDL that defines the shape and form of the database and how to query it.

SQL has excelled at OLTP (online transaction processing). This supports active online updates from thousands of concurrent users. Many systems also excel at OLAP (online analytical processing), in which complex and frequently large queries are processed against large collections of data. Systems are getting better at processing OLAP queries against rapidly changing OLTP data, but it is a difficult challenge.

One approach to meet this challenge is to move data

out of the OLTP system to somewhere where it may be more easily processed without interfering with the online transaction work. A common technique is to process the transaction system's logs as they flow to the storage used by the database. In this fashion, a slightly lagging analytical system may be kept. The shape and form of the schema for that slightly old data are captured in the DDL as of the point in time when the updates were performed. To understand this data properly, you must understand the point-in-time schema.

Interpreting the dynamic SQL as a point-in-time SQL leads to capturing the image of the data and its schema together. As time goes on, you have many sets of data, each with its own schema, and you want to analyze these together. It is not uncommon to project this data as semi-structured objects in JSON derived from the database data.¹ This is especially convenient since the semantics of the projected semi-structured objects contain their own schemas, which are meaningful as of the time the data was modified. They are self-describing. This is what I call *descriptive metadata*.^{2,4}

MISMATCHED AND MIX-MATCHED

Typically, a SQL database is not the only source of data. Nowadays, there are sensors, camera feeds, event streams, uploads from who knows where, change data from partners, analyses of customer trends, newsfeeds, and much, much more. There is an onslaught of new data as a company rolls out more servers to hold the data, either virtual or physical. All of this data has some form of metadata describing what it means. The metadata may

As machine learning has advanced, it is getting more practical to find patterns by examining the raw data.

be on the objects or events, or it may be on a data set or event stream. Still, without metadata, it's likely you'll fire up some software to discern patterns and then attach metadata.

As machine learning has advanced, it is getting more practical to find patterns by examining the raw data. Semi-structured data objects generally have a hierarchy, and each node in the hierarchy is labeled. Even when different objects have different metadata, and come from different sources, we are finding more and more success by comparing and contrasting these different objects and labeling similar things. Semantically, this can be thought of as decorating the individual objects with new attributes as patterns are detected. While learning, this is an iterative process where the patterns emerge after correlating attributes added to new data with attributes on old data. After a while, the machine learning becomes proficient enough to do pretty well as new data arrives.

ELT VS. ETL: GETTING LOADED FIRST

ETL (extract, transform, and load) is a venerable mechanism for downloading data from a system with a well-established schema and converting the data to a target schema. Sometimes this loses some of the knowledge because the schema mappings aren't perfect.^{3,4} For decades, this has been the tried-and-true mechanism for sharing data across independent systems.

ELT (extract, load, and transform) is somewhat different. In this scheme, the target system sucks up the data even without the schema. Typically, there's some subset of the needed schema in the semi-structured object, but not

always. By examining the data in the object and looking for patterns, the system can assign an interpretation.

ELT, with its deferred metadata assignment, has advantages and disadvantages. Like late binding on procedure calls across services, ELT is much more adaptive and accepting of differences and evolution. Also, like late binding, it may not perfectly match the semantics but rather give “good enough” answers.

SHREDDING DOCUMENTS TO INCREASE YOUR UNDERSTANDING

When you think about shredding documents, you think about ensuring the paper is in tiny pieces that are impossible to understand. In complex analytic environments, it’s the other way around. You can find attributes of data with the same metadata as those in other documents. By finding commonality, or pretty good commonality, you can group together the values of these attributes from many documents. Separating a document into its constituent attributes and storing them separately is called *shredding*.

Sometimes, all the attributes in a document are shredded into these groupings. Other times, some attributes don’t fit well with others and are left in their original document. Depending on the content of the document, the shredded representation may be complete enough to reconstitute the original document if needed. In this case, you can discard the original without losing data.

By putting these related attributes next to each other, it is possible to view them as a column and create super-fast, column-oriented analytics. When this happens, the system

is scooping up data, organizing it by inferred relation, and creating fast analytic capacity. As more data is ingested, it is reorganized to support fast queries. Additional information is continuously merged into the old. Now, very fast analytics can be performed over the shredded input documents.

DATA LAKES, DATA PONDS, AND DATA OCEANS

Data lakes are centralized repositories for structured, unstructured, and semi-structured data. For many companies, public clouds make data lakes possible. These clouds require a lot of technology to build and operate. The collections of disparate data can be big, ginormous, or relatively small. Nowadays, it is not too complex to gather the data, name it, and provide read access. What *is* complex, independent of size, is reasoning about all the different schema and data contained within.

We are progressing beyond expecting that everything is in a single database. Rather, we see an out-of-tune chorus of different representations, sources, and metadata. It is no longer possible to expect a tight understanding of all the details and nuances of the combined data. Increasingly, getting “pretty good” insights across these many disparate sources is valuable. Indeed, frequently it is vital.

Blue whales are believed to be the largest animals to have ever lived on Earth. Some humongous blue whales have been known to engulf up to 220 tons of krill-laden seawater at a time and filter it through the baleen in their mouths. In this way, they can keep only the best stuff to be swallowed and processed, and to enrich the ocean water with the output.

In the newly emerging data lakes, data ponds, and data oceans, it is incredibly valuable to have systems that ingest, analyze, and label data with meaningful (or partly meaningful) metadata. Recent advances in pattern matching and machine learning are now creating surprisingly useful output. Just like blue whales.

References

1. Helland, P. 2005. Data on the outside vs. data on the inside. *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*; <http://cidrdb.org/cidr2005/papers/P12.pdf>.
2. Helland, P. 2011. If you have too much data, then “good enough” is good enough. *acmqueue* 9(5); <https://queue.acm.org/detail.cfm?id=1988603>.
3. Helland, P. 2016. The power of babble. *acmqueue* 14(4); <https://queue.acm.org/detail.cfm?id=3003188>.
4. Helland, P. 2019. Extract, shoehorn, and load. *acmqueue* 17(2); <https://queue.acm.org/detail.cfm?id=3339880>.

Pat Helland has been implementing transaction systems, databases, application platforms, distributed systems, fault-tolerant systems, and messaging systems since 1978. For recreation, he occasionally writes technical papers. He works at Salesforce. Pat’s blog is at pathelland.substack.com.

Copyright © 2020 held by owner/author. Publication rights licensed to ACM.