Twoogle: Searching Twitter With MongoDB Queries

Wolfram Wingerath[1] Felix Gessert[2] Norbert Ritter[3]

Abstract: Modern real-time databases follow the same collection-based querying semantics as traditional database systems. Targeting interactive workloads, real-time databases do not only deliver a query's result upon request, but also produce a continuous stream of informational updates thereafter. In theory, building interactive, reactive, or collaborative applications should thus be simple with collection-based real-time queries as they bridge the gap between traditional database queries over static collections and continuous queries over dynamic data streams. In practice, though, building real-time applications is still considered challenging, since most real-time databases today provide only poor scalability, confusing interfaces for real-time data access, and reduced query expressiveness in comparison to their pull-based counterparts. In this demo, we illustrate that scalability, query expressiveness, and simplicity can go hand-in-hand for modern real-time databases. To this end, we present the social media search app Twoogle which is built on top of Baqend's real-time query API.

Keywords: Real-Time Databases, NoSQL, Scalability, Query Expressiveness, Real-Time Queries, Push-Based Data Access, Self-Maintaining Queries, Event Stream Queries, Search

1 Introduction

Traditional databases are optimized for pull-based queries that make information available only as a response to an explicit client request. While this access pattern is adequate for mostly static domains, it requires inefficient and slow workarounds (e.g. periodic polling) when data is in flow and clients need to stay up-to-date at all times. **Real-time databases** such as Firebase, Meteor, RethinkDB, and Parse address interactive and reactive applications by proactively pushing new information to their clients in realtime, i.e. as soon as new data has been committed to storage. But current implementations are either unscalable or avoid the complex queries that make them so appealing in theory: For example, the original Firebase only supports sorting by a single attribute, while Parse does not support sorted real-time queries at all and RethinkDB only supports sorted real-time queries with limit, but not with an offset clause [WGW^{+}18]. In consequence, developers often find themselves evaluating complex performance trade-offs or compensating for missing query expressiveness when building reactive applications on top of a state-of-the-art real-time database.

[1] Baqend, Stresemannstraße 23, 22769 Hamburg, ww@baqend.com
[2] Baqend, Stresemannstraße 23, 22769 Hamburg, fg@baqend.com
[3] Universität Hamburg, DBIS, Vogt-Kölln-Straße 30, 22527 Hamburg, ritter@informatik.uni-hamburg.de

doi:10.18420/btw2019-37

In this demo, we illustrate how the real-time database in production at Baqend[4] [Win18] combines the query expressiveness of a common NoSQL document store [GWFR16], a great ease-of-development, and push-based data delivery. To this end, we present **Twoogle**, an interactive real-time social media application.

2 Collection-Based Real-Time Queries

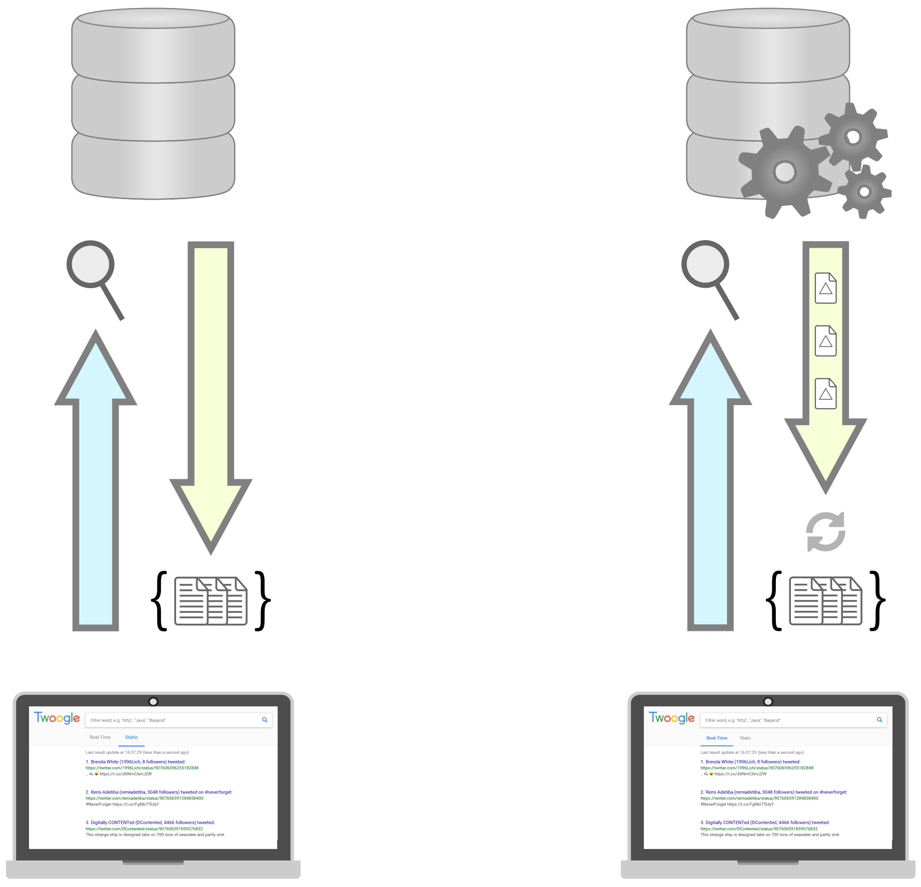
There are two distinct types of real-time database queries that differ in the way they expose data to the application: **Event stream queries** simply present the raw change events to the application, so that it can maintain an up-to-date copy of the result or apply custom business logic. **Self-maintaining queries** are more abstract as they do the result maintenance in a transparent manner and provide the client with the complete (updated) query result on every change. Using the latter, reactive behavior can be implemented without any notion of data streams or change events built into the business logic. Some static applications can be transformed into real-time applications simply by switching the underlying query mechanism – without even touching application code.

For query interfaces based on callback functions, the transition between static and real-time behavior is particularly smooth. When a query is executed asynchronously, the main thread of execution does not wait for the result to return. Instead, a callback function is provided at query time to specify how the result should be processed. When it is received, the callback function is invoked with the result as an argument.
If the query is executed as a static ad hoc query (see Figure 1a), the callback function will only be called once. If the query is executed as a self-maintaining real-time query (see Figure 1b), on the other hand, the callback function will be invoked once when the initial result is returned and then again every time when the underlying data has changed; thus, the application effectively behaves as though a new static query was executed whenever it would return a different result than the last invocation. As the only application requirement, every callback function has to be implemented in idempotent fashion, so that an invocation overrides all effects of the previous invocation. For illustration, consider a search app or website where a callback function renders the result of a user-defined query. If the callback function is executed in the context of a self-maintaining query, it has to remove any result representations that were generated earlier or else the search result might grow indefinitely or might otherwise reflect the real-time updates incorrectly.

3 Demo Outline

As a showcase application, we implemented a social media search app named **Twoogle**[5] [Win17] that provides an interface for searching a database of Twitter messages that is

[4] Baqend: `https://www.baqend.com`.
[5] To see a running version of Twoogle, visit `https://twoogle.info`.



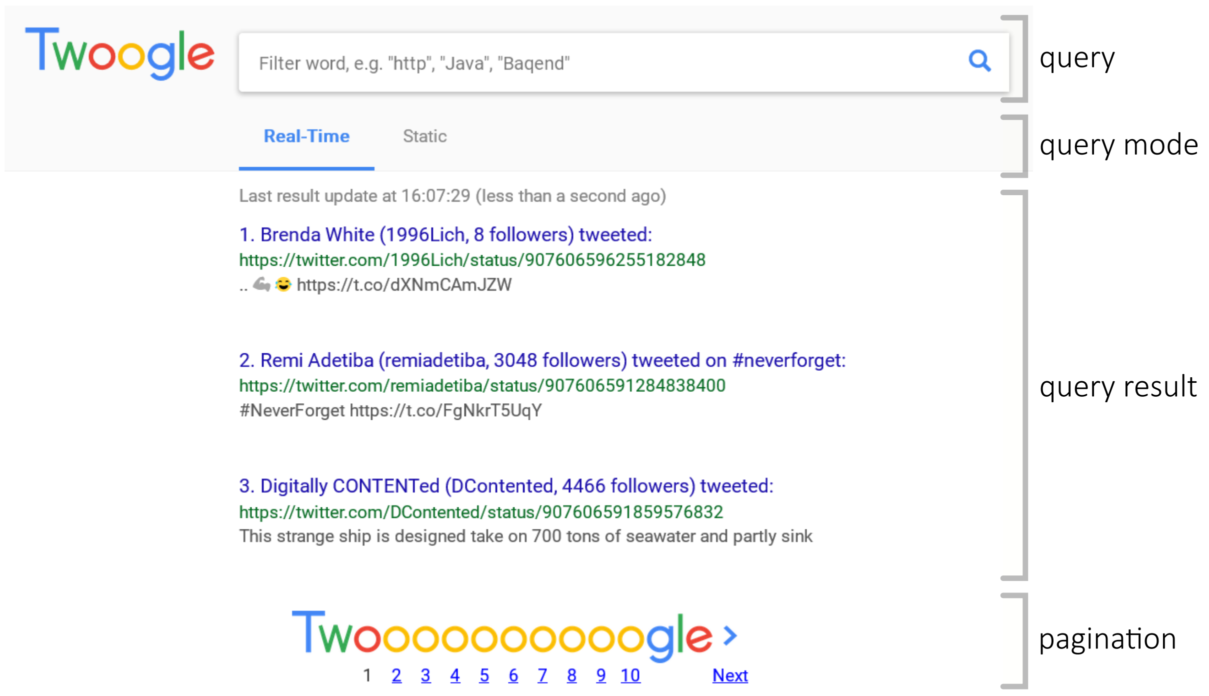
(a) A static ad hoc query produces a single result that represents database state at query time.

(b) A self-maintaining query yields a sequence of results, each reflecting the latest update.

Fig. 1: While a static ad hoc query is *pull-based*, a self-maintaining (real-time) query *pushes* updates to the client and presents a new result on every change.

permanently updated. A continuous process running on the Baqend application server receives live tweet messages as they are created by users anywhere on the world, each of which is inserted into the `Tweet` database collection. Twoogle users can then search all tweets in the continuously evolving database collection by typing a filter expression into the web interface. The currently active real-time search is canceled whenever the search query is updated.

The Twoogle user interface (UI) is depicted in Figure 2. Similar to other search engines, Twoogle displays the result of a user-defined filter query, newest matches first. The result is structured in pages, so that users can retrieve older messages by selecting a later page. A feature that is probably unique to Twoogle is the ability to toggle the **mode of query execution**: Users can choose between push-based real-time queries and pull-based static queries by clicking the corresponding button. When the query is executed in pull-based fashion, the search result is generated on page load and whenever the user modifies the query, for example by typing into the search field or navigating to another page. With a



Last result update at 16:07:29 (less than a second ago)

1. Brenda White (1996Lich, 8 followers) tweeted:
https://twitter.com/1996Lich/status/907606596255182848
.. https://t.co/dXNmCAmJZW

2. Remi Adetiba (remiadetiba, 3048 followers) tweeted on #neverforget:
https://twitter.com/remiadetiba/status/907606591284838400
#NeverForget https://t.co/FgNkrT5UqY

3. Digitally CONTENTed (DContented, 4466 followers) tweeted:
https://twitter.com/DContented/status/907606591859576832
This strange ship is designed take on 700 tons of seawater and partly sink

Fig. 2: Twoogle is a social media application that enables user-defined real-time queries over live Twitter messages.

self-maintaining query, in contrast, the user-defined result also refreshes itself whenever a new matching tweet is inserted into the `Tweet` collection or when a matching tweet is updated or leaves the result.

As part of the demonstration, we will show how self-maintaining queries turn a static version of Twoogle into a real-time version with only three simple lines of code. We will further demonstrate that event stream queries impose additional complexity as they require the developer to handle individual change events, but at the same time also enable more complex query patterns (e.g. real-time aggregations) and sophisticated notification mechanisms.

4 Future Directions

Baqend's real-time query engine currently supports MongoDB semantics for sorted filter queries with `limit` and `offset`, including query operators for content-based filtering through regular expressions (`$regex`), comparisons (e.g. `$eq`, `$ne`, `$gt`, `$gte`), logical combination of filter expressions (e.g. `$and`, `$or`, `$not`), evaluating matching conditions over array values (e.g. `$in`, `$elemMatch`, `$all`, `$size`), and various others (e.g. `$exists`, `$mod`). The current real-time query engine does not support aggregations or joins; prototypical implementations of full-text search and geo queries are being evaluated, but have not been rolled out into production at the time of writing. Extending the Baqend real-time query engine to these query types is work in progress and may therefore be addressed in future publications.

References

[GWFR16] Felix Gessert, Wolfram Wingerath, Steffen Friedrich, and Norbert Ritter. NoSQL Database Systems: A Survey and Decision Guidance. *Computer Science - Research and Development*, 2016.

[WGW$^+$18] Wolfram Wingerath, Felix Gessert, Erik Witt, Steffen Friedrich, and Norbert Ritter. Real-Time Data Management for Big Data. In *Proceedings of the 21th International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018*. OpenProceedings.org, 2018.

[Win17] Wolfram Wingerath. Going Real-Time Has Just Become Easy: Baqend Real-Time Queries Hit Public Beta. *Baqend Tech Blog*, September 2017. Accessed: 2017-09-26.

[Win18] Wolfram Wingerath. *Scalable Push-Based Real-Time Queries on Top of Pull-Based Databases*. PhD thesis, University of Hamburg, 2018.