A Case Study of the Aggregation Query Model in Read-Mostly NoSQL Document Stores

Diego Pasqualin, Giovanni Souza, Eduardo Luis Buratti,
Eduardo Cunha de Almeida, Marcos Didonet Del Fabro, Daniel Weingaertner
C3SL Labs, UFPR, Brazil
{dpasqualin,gvs11,elburatti,eduardo,didonet,danielw}@inf.ufpr.br

ABSTRACT

In this paper we focus on the aggregate query model implemented over NoSQL document-stores for read-mostly data bases. We discuss that the aggregate query model can be a good fit for read-mostly databases if the following design requirements are met: on-line time range queries, aggregates with predefined filters, frequent schema evolution and no ad-hoc. In our model, we present a composite object schema implementation over NoSQL document-stores, in which data associations are nested in a document under the same search key. We present the design choices to obtain a model adapted to our needs. Our schema is inspired by the star schema of Data Warehouses to reduce accessing data associations in many different documents and computing aggregates within the same composite. We present performance results of our empirical study over a 300 million records database that serves in production for the Ministry of Communications of Brazil. Results show the performance gains and penalties of our star composite schema when compared to the traditional multidimensional schema.

CCS Concepts

•Information systems → Data management systems; Database design and models; Data model extensions;

1. INTRODUCTION

NoSQL database systems appear as a popular alternative to produce BigData software for large volumes of data. With NoSQL, agile software development can benefit from several aspects, including schema-flexible databases [4] to allow software development with short release cycles, and Object-oriented (OO) database design [2] to ease (un)marshalling between objects and the storage model. For instance, in MongoDB, one well known document store, the "Binary JSON" storage format is a marshalled JSON document. Application producers only care about persisting and retrieving from JSON and let the storage marshalling to MongoDB.

IDEAS '16, July 11 - 13, 2016, Montreal, QC, Canada
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4118-9/16/07...$15.00
DOI: http://dx.doi.org/10.1145/2938503.2938546

NoSQL systems allow designing nested objects with imposed "Is-Part-Of" relationships, also known as "Composite Objects" [13]. In the storage layer, composite objects with keys and their data associations can be marshalled in hierarchical storage representation (i.e., nested documents) [12, 2]. Thus, we argue that the "Is-Part-Of" relationships indicate aggregate data, also pointed out by [13], and can be used to boost performance of read aggregate operations.

Systems with read-mostly databases can benefit from composite objects. Some types of read-mostly systems are: Business Information Services (BIS), Customer Relationship Management (CRM) and electronic library catalog. In the storage layer, these systems traditionally run atop Data Warehouses [16] where the storage model can be de-normalized and transactions are modelled as "fact tables" described by "dimension tables" (i.e., multidimensional model). Performance of the multidimensional model shines through when the storage model avoids imposing too many table join operations of complex queries with "facts" at the center of the model (i.e., star schema). However, even the multidimensional model may require further optimization for processing aggregate queries over large fact tables. The workaround for some designers is to prepare group data to represent most of the aggregate queries, like Cubes [6]. The downside is the growing complexity of the storage model that will require extra maintenance of the group data.

In this paper we discuss that read-mostly systems can leverage from composite objects to design aggregates and keep aggregate data close to the search key. We discuss the design of a composite object schema that combines the best of two worlds: the star schema from multidimensional model and the aggregation query model from OO-DBMS [12]. The main challenge to this design is to reflect the selections of most of the aggregate queries without creating extra group data. Naturally, this requires to store data associations in documents under the same search key [7]. Thus, query processing does not require accessing data associations in different documents. We also draw attention to deeply nested documents that can slow data access [12, 9]. To reduce nested data, we present the different design choices that motivated the evolution from a traditional design to a star-schema-like design. Finally, we discuss empirical results and the running conditions in which the document-oriented design performs better and worse than its relational counterpart. We use this discussion to define the design of a read-mostly database running in production for the Ministry of Communications in Brazil with more than 300 million records.

The main contributions of this papers are:

- we present an empirical study of a document oriented database design based on the aggregate query model in OO-DBMS [12] and the star schema of Data Warehouses.
- we discuss the design requirements that motivate the exploration of NoSQL document stores. We give special attention to the design of a composite object schema for read operations to avoid deeply nested documents and keep aggregate data under the same document.
- we discuss pros and cons of such document oriented design compared to the multidimensional relational designs when upon the same workload. For this, we present empirical results comparing both designs for a data base that serves in production for the Ministry of Communications in Brazil.

This paper is organized as follows: Section 2 discusses related work. Section 3 presents the case study and the design requirements. Section 4 presents the design challenges of document-stores for the read-mostly database and the proposed Document-Oriented design. Section 5 presents empirical results and we conclude in Section 6.

2. RELATED WORK

In the abstract level, there are different data representations that can be used to design aggregate data schema, including *Entry per Aggregate Object (EAO)* and *Entry per Top-Level-Field (ETF)* [2]. These data representations are the basis of the NoSQL Abstract Model (NoAM) [3]. The NoAM is particularly handy when designing databases over NoSQL, specially when the storage model has not yet been decided. For instance, the software producer can switch over the NoSQL back-end from document store to a graph store based on EAO. Unfortunately, storage and performance impacts of such flexibility can vary substantially from one data representation to the other. Thus, designers are likely to choose a specific data representation when the database workload trend is already known.

The aggregate query model for OO-DBMS [13] presents a possible composite object schema to be implemented in document-stores, although the paper does not discuss any performance issues nor empirical results. Similarly, more recent work on NoSQL document stores describe "best practices" on how to compute aggregation data [5, 10], but none of them discusses how to benefit from composite objects to boost performance. They only pinpoint possible implementation methods based on batch processing in MapReduce or Pipeline[1]. [10] presents an empirical investigation of NoSQL document-stores running aggregates for a specific use case in log analysis. However, the document-store design is taken for granted and no discussion of pros and cons against another possible design is presented.

For query processing over composite objects, [17] presents a comprehensive cost model for forward and reverse path traversal. We do not discuss cost models in this paper, however, it inspired the proposed schema optimization to reduce the need for path traversal.

[1]MongoDB Pre-aggregated Reports available online at: https://docs.mongodb.org/ecosystem/use-cases/pre-aggregated-reports/

3. CASE STUDY AND DESIGN REQUIREMENTS

In this section, we present the case study that motivated the implementation of a BIS over NoSQL document-stores. Over the last years, the Brazilian government has been investing in nationwide projects to improve connectivity within government institutions and provide better digital inclusion for the Brazilian citizens. There are three main ongoing projects: i) *GESAC* that aims to bring internet connection to remote areas where commercial support is not available (like in the middle of the Amazon Forest), ii) *Digital Cities* that consists of a high speed fiber network connecting government buildings in small towns, while also providing free broadband internet connection in public spots, and iii) *Telecenters* that are public LAN houses with free internet access and open courses for digital inclusion.

With the increasing investments to accomplish these initiatives, the government is also acting to assess whether the investments are actually implemented. Therefore, every point of presence (*Gesac* or *Digital City* router, and computers on *Telecenters*) is being monitored, with network usage collected in a five minute interval, for a current number of more than 10,000 devices. Network usage, together with some other metrics, not presented in this paper, can provide evidences on how the infrastructure is being used for planning future upgrades and reacting to possible outages.

The BIS developed to address this amount of data, called SIMMC[2], was commissioned with the following requirements: on-line time range queries, aggregates for predefined filters (to fill predefined charts), no ad-hoc and agile development for frequent software releases. The latter requirement is important because of the frequent addition of new points of presence with different attributes.

The first version of the SIMMC was developed with Data warehousing concepts: Extract, Transform and Load (ETL) with overnight batch and multidimensional relational schema. Figure 1 illustrates the data flow through SIMMC system, starting in the *Collecting System*, where data is acquired from computers and routers, going to an API that receives data and inserts it into the *Storage System*. The *Storage System* is a relational database with a Stating Area (SA) that gets consolidated by an overnight batch process.

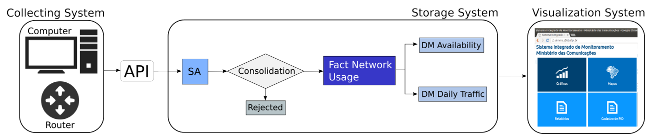


Figure 1: Data flow for SIMMC system.

The biggest table to ingest data stores network traffic. This table, called *fact network usage*, is the central piece to build aggregate data into Data Marts for fast generation of charts and reports in the *Visualization System*. The *fact network usage* accounts for more than 300 million records and is composed by attributes grouped as dimensions and metrics. The dimensions are *id_point* (point identifier), *id_city* (location identifier), *macaddr* and *ip_address* (device identi-

[2]SIMMC, translated from Portuguese to "Monitoring System of the Ministry of Communication", available online at http://simmc.c3sl.ufpr.br/

fier[3]), and finally *contact_date* and *collect_date* (the date/-time the collect agent contacted our server and retrieved the data from the device, respectively). As metrics, we have *download_bytes*, *download_packages*, *upload_bytes* and *upload_packages*, accounting for the amount of bytes and packets transferred in and out of the device during the interval $[collect\_time, collect\_time - 5minutes]$. Indexes are present for *collect_date*, *collect_time* and *id_point*, the most common search keys for aggregations in our queries.

However, performance degraded overtime and combined to the addition of new points of presence, we had to constantly evolve the database schema. This became a major problem to deal with in the multidimensional model. Therefore, we decided to explore one of the benefits of NoSQL document-stores: they allow agile schema evolution, although schema evolution is not the focus of this paper (see [14] for a discussion of schema evolution in NoSQL). When switching a BIS to NoSQL, we found ourselves with performance problems to run aggregate queries in documents, specially for our large volume database. The goal of this paper is to understand the performance gains and penalties of a NoSQL document-store design running the SIMMC database.

4. DOCUMENT-ORIENTED DESIGN

In the SIMMC, each monitored device submits network monitoring data as a record within a five minutes interval. In the multidimensional model, we transform this record to fit into fact (traffic information) and dimension tables (device description). We recall that the biggest challenge in the SIMMC is the *fact network usage* to motivate the discussion in this section. In the document model, different alternatives are possible to transform and ingest the 300 million records, but with performance considerations that need to be taken into account. The designer could follow a naive approach and store all records in a single document, but this would result in several concurrency contention issues [15]. Another naive approach is to convert each record to a document in a one-to-one fashion. However, it would lead to performance penalties in the storage level with 300 million separate documents.

When data is nested under different documents, query processing requires traversing from a document to another [11, 12]. This processing is called path traversal in OO-DBMS. Path traversal can be done in forward or reverse way. Queries perform forward path traversal when filters are known. The benefit of forward path traversal is that data associations can be directly fetched in the order of the hierarchy. However, database schema with deeply nested associations can slow down data access since many path traversal steps may be required to access a single object, thus creating concurrency problems in situations of mixed workloads, such as failed transactions [15]. When query filters are unknown the database system performs the reverse path traversal with high performance cost (see [17, 11] for a comparison between forward and reverse path traversal).

To sum up, the main challenges to our design are: (1) the amount of data can grow significantly as network devices are asked to submit data within a tight time window, and (2) the majority of queries submitted require the aggregation of massive amounts of data.

[3]As we monitor different type of devices, we need particular identifiers for each one. They are de-normalized here because of performance concerns.

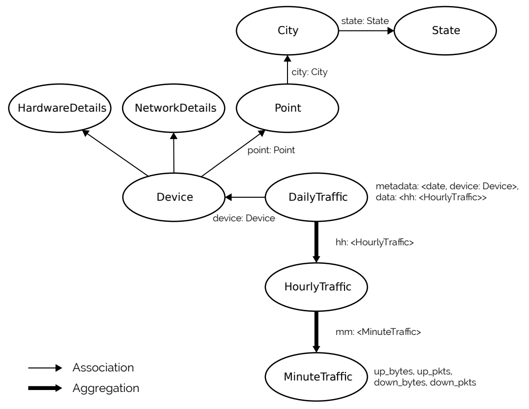


Figure 2: Composite object schema for daily network traffic data.

In this section, we describe our design to efficiently keep aggregate data in NoSQL document-stores for read-mostly databases. As document-stores implement OO functionalities, our design was originally inspired by the traditional O2 database language [9] and the Aggregation Query Model. We show that the design choices evolved the model to better handle path traversal operations and aggregate data.

4.1 Traditional composite modeling

We present a first object model proposal following traditional document-oriented design practices, where parts of the model contains aggregation relationships. The convenience of such design is that it can be embedded in NoSQL documents under the same search key, avoiding performance penalties in the storage level.

The object model (also called object schema) is defined as a directed labeled graph $G = (V, A)$, where V is the set of vertices, which represent the classes; and A is the set of arcs, representing the relationships (aggregations or compositions). Our object schema is illustrated in Figure 2. Aggregation relationships are differentiated by boldface arcs. Each node is labeled with a class name and a set of attributes.

The class *DailyTraffic* is the root of a composite object. In a document oriented design, it is possible to store all the components of a composite object into a single document. This enables objects to be loaded without requiring path traversal operations (i.e., navigation through the object relationships) to fetch all its components, reducing the cost of queries over the aggregation hierarchy. Figure 3 shows an instance of the composite object stored in JSON.

However, when considering relationships between objects that span through multiple documents, i.e., association relationships, it is necessary to load them separately. This means that, depending on the query, a significant amount of path traversal operations is required to span through different documents. For instance, consider the query written in Object Query Language (OQL) illustrated in Figure 4: its goal is to "*retrieve the total amount of uploaded Bytes for an entire state department*" .

The corresponding path traversal graph indicates the need of 5 path traversal operations. In addition, this query performs reverse path traversal on all steps for two reasons: (1) it does not perform any filtering (selection operations) on

```
{   metadata: {
      device: "10.151.52.1",
      date: "2015-05-07"
    },
    data: {
      00: {
        00: {
          up_B: 50672, up_pkt: 348,
          dl_B: 140372, dl_pkt: 329
        }, ...
        55: {...}
      }, ...
      23: {...}
    }}
```

Figure 3: JSON representation of a document for the network traffic data.

```
SELECT D.date, SUM(
    SELECT SUM( SELECT M.up_bytes From M in H )
    From H in D.data
) FROM
     D in DailyTraffic, DV in D.device,
     P in DV.point, C in P.city, S. in C.state
WHERE S.name = "Sao Paulo";
```

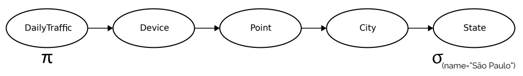


Figure 4: Upload data for "Sao Paulo" state department.

the root class, and (2) the number of accesses to the associated class through path traversal from the root class is greater than the original number of associated objects.

4.2 Star-based composite modeling

In this section we present an alternative object model that minimizes the path traversal operations. We model the inter-document associations inspired by traditional Data Warehouse "Star-schemas". While it may not be the most intuitive modeling technique, we link some of the classes directly on the root class, as shown in Figure 5.

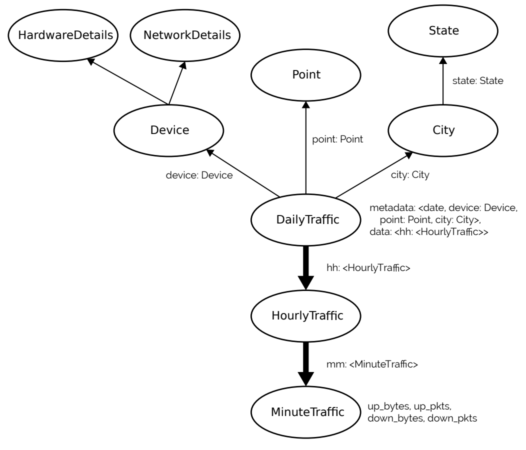


Figure 5: Star-based Composite Object Schema

The implementation of such change in the document is straightforward, since it is only necessary to encode the inter-document associations in the document metadata. The benefit is similar to what is expected from "Star-schemas": we reduce the number of path traversals. Figure 6 shows the query over the star-based composite to answer the same question presented in the previous section.

```
SELECT D.date, SUM(
    SELECT SUM( SELECT M.up_bytes FROM in H )
    FROM H in D.data
) FROM D in DailyTraffic,
       C in D.metadata.city, S. in C.state
WHERE S.name = "Sao Paulo";
```

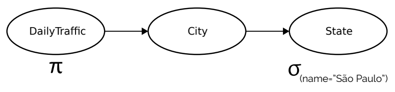


Figure 6: Upload data for "Sao Paulo" state department in the star-based composite.

4.3 Integrating aggregated data

Operations to summarize data is commonplace in read-mostly databases to support insight reports about business processes. This operation potentially requires the aggregation of a massive amount of data. A common strategy used in data warehouses is to keep pre-calculated aggregation tables [8] with summarized facts, constructed by either eliminating dimensionality or by associating the facts with rolled-up dimensions. The benefit of this approach is dramatically positive for queries on large data sets with performance improvements in orders of magnitude [1].

In our case, the network traffic data is modeled in such a way that aggregate data can be easily stored in the composite object itself. For instance, in the class *DailyTraffic*, it is possible to store the pre-aggregated sum of *HourlyTraffic* (with a "many-to-one" relationship to *DailyTraffic*). With this design, queries do not need to traverse the entire *DailyTraffic* to compute aggregate data for the "daily" granularity.

The maintenance of aggregated data is also easy on our model. Since all the pre-aggregated data is contained in the same document, when an update or insertion happens.

5. EXPERIMENTAL RESULTS

In this section we present an empirical investigation of our document oriented design and a discussion about storage footprint and performance, compared to the multidimensional relational model. A snapshot of the SIMMC database was taken for each model and the experiments run on the same machine, a Xeon E5506@2.13GHz processor with 12GB of RAM and Debian 8.3 GNU/Linux OS. The queries were implemented in Node.js[4] and the backends for the relational store is PostgreSQL 9.4 and for the document-store is MongoDB 3.2, with default WiredTiger storage engine.

5.1 Storage Footprint

In a document oriented database the name of the attributes (i.e., metadata) need to be repeated in every document. This fact can represent a substantial overhead on storage footprint. Given that our database has around 300 million records with eleven attributes, each record stores 114 bytes of metadata which uses approximately 32GB of storage. Although the cost for hard disks are continuously dropping, it is important to notice that the extra space for metadata impacts on the efficiency of the cache system, since less data can fit in memory at a given time.

[4]https://nodejs.org/en/

The first effort commonly used to mitigate this issue is to shorten the attribute names, preferably to something that can still be meaningful to the developer. Reducing the attribute names by just a few bytes can save a considerable amount of space as the number of records grow. For instance, in a naive "one record-to-one document" design of the *fact network usage* table, simply shortening our attributes *upload* and *download* to *up* and *down* would reduce the overhead by 37%, from 32GB to 20GB. However, the real gain comes from the star-based composite schema, reaching an even better storage footprint than its relational counterpart.

A single document on the composite object model aggregates information of an entire day, while the relational model stores one record for each five minute interval. This lower granularity level allows us to avoid repetition on a ratio of 288 : 1 for every dimension value on the de-normalized relational model, namely: id_point, device, location and collect_date. Now these attributes are stored a single time on each document. As a result, despite the repetition of attribute names, the document-oriented model has achieved a considerable lower storage footprint. We compare disk space usage for both MongoDB and PostgresSQL, grouped by data itself and B-tree indexes (which are the same on both models). Considering only data, PostgreSQL consumed 37GB of disk space against 11.5GB for MongoDB. As for indexes, MongoDB was remarkably more efficient, using mere 10MB of disk space compared to 12GB on PostgreSQL. This difference is due mainly to the lower data granularity on the non-relational schema, but can be also attributed to an efficient data compression engine implemented in MongoDB[5]. Consequently, having a smaller amount of records, it was nine times faster to create indexes on MongoDB.

5.2 Performance

The performance measurement between the two models was designed to test common queries and aggregations used to generate reports and charts for the visualization system. We consider the impact of indexes and different levels of granularity for the aggregation.

Figure 7 shows the response time for a query that returns all stored data within one week. Results are presented in logarithmic scale and both sequential and index scan benefit from the composite model, specially due to the smaller amount of documents to be fetched from MongoDB than records from PostgreSQL.

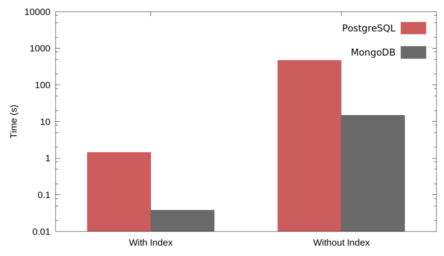


Figure 7: Query on attribute date.

In a more in-depth experiment we perform a progressive increase in the selectivity factor to analyze the impact of a table and collection scan (see Figure 9). The x axis shows the selectivity factor, while y axis represents the query latency compared to a full table scan. The baseline represents the hypothetical scenario where latency increases at the same ratio as the selectivity factor.

It is noticeable that the composite model implemented on MongoDB follows the baseline performance, while the relational model in PostgreSQL degrades at the very beginning. First of all, by using composite objects we have fewer records, consequently reducing the time to fetch the requested data. Besides, when selectivity factor is greater than 10% PostgreSQL chooses to do a table scan, while MongoDB still relies on indexes.

For the next experiment we have selected four different aggregation keys to assess performance between the two models: i) "PoP,day,min", to aggregate metrics (bytes and packets transferred in/out of the device) for every device that belongs to a PoP (Point of Presence) for each five minute interval within a day, ii) "PoP,day" to sum up metrics for every PoP in one day interval, iii) "PoP" to sum up metrics for each monitored PoP, and finally iv) "city" to group metrics by the cities where "PoPs" are located. All queries touch the entire database. Aggregation in MongoDB was performed via MapReduce framework[6], while for PostgreSQL we used the standard "GROUP BY" statement.

Figure 8a displays the results, this time with the relational model running faster by a considerable margin for every aggregation key. Results might be concerning depending on the frequency that aggregations need to be generated and whether it is possible to perform incremental aggregation, meaning that new data can just be merged into existent table/collection. As this is our case, we created functions on PostgreSQL to mimic MongoDB built-in incremental aggregation and ran the tests simulating ingestion of one day worth of data. Figure 8b shows the results, and whilst MongoDB is still slower, the difference is less relevant as we can easily update results faster than they arrive (5-minute interval bursts), assuring a real-time view for the SIMMC system.

6. CONCLUSION

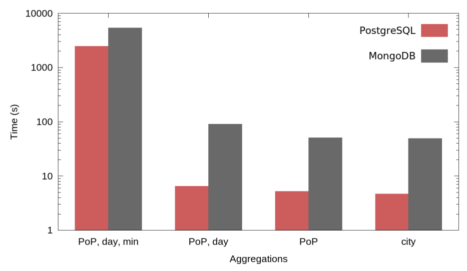
In this paper we discussed that read-mostly systems can leverage from composite objects to design aggregate data. We present a document-oriented design combining the best of two worlds: composite objects from OO-DBMS and star schema from Data Warehouses. We showed that our document oriented design can present significant performance gains if specific design requirements are met. We also showed that our design saved storage space when compared to the traditional multidimensional relational design for BIS. Our results come from an open-source BIS database serving in production for the Ministry of Communications of Brazil with more than 300 million records. Future work is still needed to explore other optimization features, namely sharding, and also improve our data ingestion procedure as NoSQL presents many possibilities, such as MapReduce and Pipelining.

7. ACKNOWLEDGMENTS

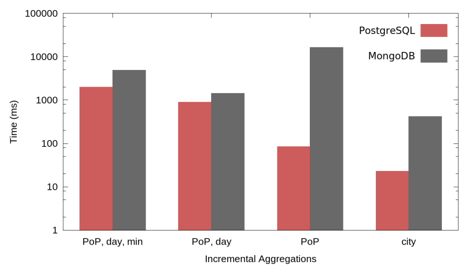
This work was partially funded by the SIMMC Project: Monitoring System of the Ministry of Communication of Brazil, the FNDE Telecenter Project of the Ministry of Education of Brazil and CNPq grant 441944/2014-0.

[5]MongoDB WiredTiger white-paper, available online at: https://docs.mongodb.org/v3.0/core/wiredtiger/

[6]https://docs.mongodb.org/manual/core/map-reduce/



(a) Aggregation.



(b) Incremental aggregation.

Figure 8: Aggregation time between PostgreSQL and MongoDB.

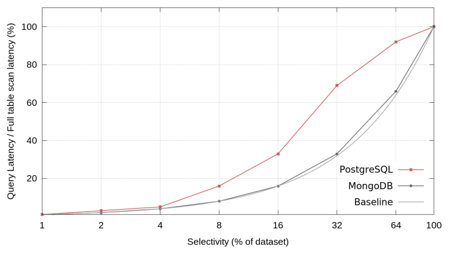


Figure 9: Selectivity factor between PostgreSQL and MongoDB.

8. REFERENCES

[1] C. Adamson. *Mastering Data Warehouse Aggregates: Solutions for Star Schema Performance*. Wiley, 2006.

[2] F. Bugiotti, L. Cabibbo, P. Atzeni, and R. Torlone. Database design for nosql systems. In *Conceptual Modeling - 33rd International Conference, ER*, pages 223–231, 2014.

[3] F. Bugiotti, L. Cabibbo, P. Atzeni, and R. Torlone. How I learned to stop worrying and love nosql databases. In *23rd Italian Symposium on Advanced Database Systems, SEBD*, pages 216–223, 2015.

[4] T. Cerqueus, E. C. de Almeida, and S. Scherzinger. Safely managing data variety in big data software development. In *1st IEEE/ACM International Workshop on Big Data Software Engineering, BIGDSE@ICSE*, pages 4–10, 2015.

[5] K. Chodorov. *MongoDB: The Definitive Guide*. O'Reilly, 2013.

[6] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Min. Knowl. Discov.*, 1(1):29–53, jan 1997.

[7] I. Katsov. Data modeling at scale: Mongodb + mongoid, callbacks, and denormalizing data for efficiency. http://blog.michaelhamrah.com/2011/08/data-modeling-at-scale-mongodb-mongoid-callbacks-and-denormalizing-data-for-efficiency/, 2011.

[8] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*, chapter Building the Data Warehouse, pages 331–370. 2nd edition, 2002.

[9] C. Lécluse and P. Richard. The o2 database programming language. In *Proceedings of the 15th International Conference on Very Large Data Bases*, VLDB '89, pages 423–432, 1989.

[10] K. Mahmood, T. Risch, and M. Zhu. Utilizing a nosql data store for scalable log analysis. In *Proceedings of the 19th International Database Engineering & Applications Symposium*, pages 49–55, 2015.

[11] G. Mitchell, S. B. Zdonik, and U. Dayal. Optimization of object-oriented queries: Problems and approaches. In *NATO ASI OODBS*, pages 119–146, 1993.

[12] H. Pirahesh, B. Mitschang, N. Südkamp, and B. G. Lindsay. Composite-object views in relational DBMS: an implementation perspective. In *4th International Conference on Extending Database Technology, EDBT'94*, pages 23–30, 1994.

[13] J. W. Rahayu, D. Taniar, and X. Lu. Aggregation query model for oodbms. In *Proceedings of the Fortieth International Conference on Tools Pacific: Objects for Internet, Mobile and Embedded Applications*, CRPIT, pages 143–150, 2002.

[14] S. Scherzinger, T. Cerqueus, and E. C. de Almeida. Controvol: A framework for controlled schema evolution in nosql application development. In *31st IEEE International Conference on Data Engineering, ICDE, 2015*, pages 1464–1467, 2015.

[15] S. Scherzinger, E. C. De Almeida, F. Ickert, and M. D. Del Fabro. On the necessity of model checking nosql database schemas when building saas applications. In *Proceedings of the 2013 International Workshop on Testing the Cloud*, TTC 2013, pages 1–6, 2013.

[16] M. Stonebraker and U. Çetintemel. "one size fits all": an idea whose time has come and gone. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 2–11. IEEE, 2005.

[17] D. Taniar. Forward vs. reverse traversal in path expression query processing. In *TOOLS 1998: 28th International Conference on Technology of Object-Oriented Languages and Systems*, pages 127–140, 1998.