Evaluating Query Energy Consumption in Document Stores

Duarte Duarte and Orlando Belo(✉)

ALGORITMI R&D Centre, University of Minho, 4710-057 Braga, Portugal
obelo@di.uminho.pt

Abstract. Today's system users demand fast answers when querying their own databases. Their impatience still high when waiting for the results of a query when they take more than one or two seconds to appear on the screen. However, having fast querying answers it is not the only aspect that determines the quality of a database system we are using, but also the energy consumption involved with. The development of database systems increasingly economic in terms of energy consumption has led to great technological advances in this area. Today, many of the entities that manage large data base systems pay particular attention to this issue, not only for environmental reasons but also for economic reasons, obviously. In this paper we address the issue of queries energy consumption evaluation in database systems, with particular emphasis to those that are executed in a environment of a document store. Based on the information provided by the execution of a query in MongoDB, we designed and developed a process that determines the energy consumption of queries launched in a document store, approaching different alternatives in query designing, implementation and execution.

Keywords: Document stores · NoSQL · Query processing · Energy consumption plans · Green computing · Green queries · And MongoDB

1 Introduction

For a long time, relational database management systems have dominated database systems market. Although still constitute the most common data model used in database systems, from the time intensive data processing applications (big data applications) began to be common. Thus, other data models began to gain their place in this market, having very sophisticated solution for large databases, with performance, storage and scale characteristics quite interesting when compared to the traditional relational data model—e.g., key-value, wide-column, graphs and document stores [10]. In general terms, we can say that these data models are the ones that support today the large emergence of the overall NoSQL approaches. They sustain a large number of real world big data applications, providing support for large scalable, reliable and fast databases. However, as it happens in relational database systems, NoSQL systems consume also a lot of energy in querying processing. In order to evaluate their energy consumption, we selected one of the NoSQL systems that is quite popular and highly adopted in many database systems applications: the document stores, also known as document-oriented databases [12]. The use of this type of NoSQL database is growing significantly as well is the energy concern in the IT domain. This induced us to study and develop a process to allow for evaluating the energy consumption carried out by queries executed in a document store, providing as well as some specific information for design and implement less energy consuming queries. Our idea is quite simple. The power consumption of a query alone is something that we could classify as ridiculous. However, in an environment of a document store in which thousands of queries are executed every day, at the month-end the sum of the energy consumption of all queries is no longer a ridiculous thing, on the contrary.

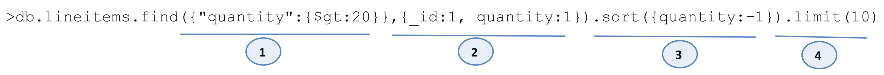
In this paper we present the work we have done for evaluating querying energy consumption in a document store management system. The main goal of this work was to establish an effective way to determine energy consumption of different querying approaches that produce the same results, pointing out the less energy-consumption one without degrading its execution time, if possible. We intend to contribute for reducing as much as possible the energy consumption of querying processes in document stores without affecting its performance on providing results. Next, we present a brief related work about energy consumption in database systems (Sect. 2), exposing the way how a document store system process a query (Sect. 3), and how we evaluated the energy consumption of a query in a document store maintained in MongoDB [11] (Sect. 4), with particular emphasis on its implementation and validation. Finally, we present the usual section of conclusions and future work (Sect. 5).

2 Related Work

In 2008, Khargharia et al. [5] warned of the need to manage well the consumption of energy and to create environmental standards for reducing power consumption in computer platforms, with a framework and a methodology for autonomous energy management in data centers. But it was only the following year, with the appearance of Claremont report [1] that concerns about power consumption in database systems were well evidenced for all stages of their development process. In that same year, also Harizopoulos et al. [4] alerted architects and database system builders about the great energy consumption the systems they built do, pointing out several relevant aspects that could improve substantially the reduce of energy consumption. Meanwhile, other initiatives were done. Note, for example, the works done by Lang and Patel [7] and Lang et al. [8], in which were designed and developed, respectively, some energy efficient data processing techniques changing performance by energy reduction, and a framework for energy-aware database query processing, augmenting query plan optimization and adding some complementary information for energy consumption prediction, and producing an energy response time profile. In the same year, Rasmussen [13] presented another approach for energy consumption estimation of database operations, and Kunjir et al. [6] demonstrated some valid alternatives to reduce the peak of energy consumption on database management systems in tasks involving complex SQL queries processing. Later, in 2014, Gonçalves et al. [3] presented another method for estimating at compile time the energy consumption of database operators integrated in a query execution plan, building up its corresponding energy consumption plan for executing the query. Later, the same authors extended their work to the data warehousing systems domain, evaluating the energy consumption of a conventional star-query launched in a data warehouse [2]. All the works referred here are only a small part of a large set of initiatives especially oriented to reduce the energy consumption in many functionalities and services of database systems. The majority of these works approached querying processing in relational databases systems. To the best of our knowledge, and based on the search we made in the literature and in the Web, there is not for now any kind of initiative covering the issue of energy consumption in NoSQL database systems or document stores in particular.

3 Querying Processing in Document Stores

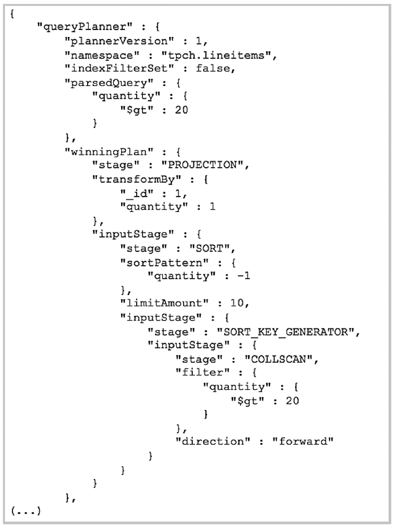
Studying the basic structure of a query in a document store and the execution plan built by the system to execute it, we get detailed information about the most elementary element used and the resources involved with, processing time and memory usage. To support this work over document stores we selected one of the most successful products in this field, which has been the preference of many NoSQL users for the last few years: MongoDB [10]. Basically, processing a query is an act of converting a search instruction for a given database written in some querying language and translating it into a set of primitive commands understandable by a system database. Additionally, processing a query also includes choosing the most efficient method to get the results corresponding to the querying instruction. In a relational database system, the queries are usually defined in SQL. However, this is not the case in non-relational databases, including document stores systems. In MongoDB queries are made in JavaScript. They have a different processing when compared to conventional relational database systems. A query in MongoDB is processed into two main steps: planning and execution. In the first step, the system aims to discover the best way to execute the query, assessing what it needs to be executed, the order for executing the various elements of the query, and how it will perform them defining a querying plan. Then, in the second step, the system performs the querying plan defined in the previous phase, using when possible the indexes defined on the collection in which the query will be executed so that its results may be obtained faster.



```
>db.lineitems.find({"quantity":{$gt:20}},{_id:1, quantity:1}).sort({quantity:-1}).limit(10)
```

Fig. 1. An example of a query in MongoDB.

To understand a little better what we done to evaluate the energy consumption of a query, it is useful to see how queries are in fact implemented in MongoDB. Queries in MongoDB return a set of documents that are contained in a given collection. For this, MongoDB provides a specific method: *db.collection.find()*. Let's see how this happens, analyzing the execution of a very simple query that is presented in Fig. 1—a query that provides us the first 10 orders recorded in a document store that have a quantity greater than 20. The query is organized in four distinct blocks, which are marked, respectively, with 1, 2, 3 and 4. The block 1 includes a predicate defining what you want to search, which is, in this case, all documents in the collection that has orders with a quantity greater than '20 '- *{"quantity": {$gt: 20}}*. Next, in block 2, a projection is set, indicating which fields should be included the results—*{_id: 1 quantity: 1}*, and in block 3 is indicated which is the field that will define the presentation order of the results and the sort criterion—*{quantity: −1}*, which is in this case a descending sort ('−1'). Finally, the block 4 includes the results cursor modification (*limit (10)*), which limits the number of documents in the result set. After defining the query we want to process, we submit it to the MongoDB's engine to be executed. The choice of an execution plan is made using a specific caching system, which maintains all the execution plans used previously that were considered viable for correspondent queries. Then the plan that was choose for the query will always be used in all future executions of the same query. However, if there is no implementation plan already established for the query, MongoDB will generate a new execution plan for it. The execution plan capable of responding to a particular query can be obtained adding the *explain()* command to the query. The explain command allows for consulting the execution plan for a given query and for obtaining statistics about its execution. The results produced by the command explain are presented in the form of a state tree. Figure 2 shows a small excerpt of an execution plan that was generated by MongoDB for the query presented previously in Fig. 1.



```
{
    "queryPlanner" : {
        "plannerVersion" : 1,
        "namespace" : "tpch.lineitems",
        "indexFilterSet" : false,
        "parsedQuery" : {
            "quantity" : {
                "$gt" : 20
            }
        },
        "winningPlan" : {
            "stage" : "PROJECTION",
            "transformBy" : {
                "_id" : 1,
                "quantity" : 1
            },
            "inputStage" : {
                "stage" : "SORT",
                "sortPattern" : {
                    "quantity" : -1
                },
                "limitAmount" : 10,
                "inputStage" : {
                    "stage" : "SORT_KEY_GENERATOR",
                    "inputStage" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
                            "quantity" : {
                                "$gt" : 20
                            }
                        },
                        "direction" : "forward"
                    }
                }
            }
        },
(...)
```

Fig. 2. An excerpt of a query's execution plan.

MongoDB leaves "footprints" during the execution of a query. They can be analyzed later, in order to understand the various states passed by MongoDB's engine during the execution of the query. For example, in Fig. 3 we can see the tree of corresponding states to implement the query of Fig. 1. The execution of the query was divided into five states, namely: (a), (b) the CollScan, (c) the Filter, (d) Sort, (e) Limit and, finally, (f) Projection. The CollScan is able to do research on a collection and return the documents in this collection. These documents are passed to the Filter condition and are filtered according to the query that has been set. In this case the query restricted all documents had to exceed 20 units. The remaining documents carried over to the Sort state where they will be sorted according to defined key.

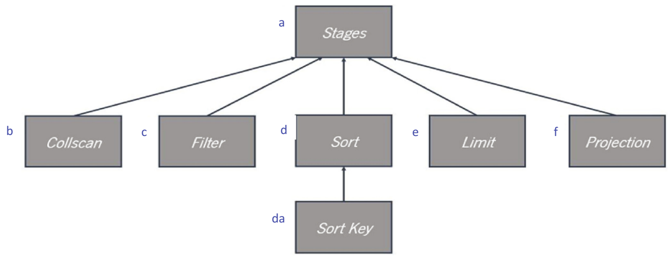


Fig. 3. The state tree of the execution of a query.

4 Defining an Energetic Comsumption Plan

The model we propose allows for a document store system administrator to compare the energy level consumption made by different queries. With this it is intended that, in addition to the definition of an execution plan for the query, something quite usual in most DBMS, you can now also possible to calculate the total energy consumption made in implementation. With this information we can improve the query or choose another to produce the same result. Let's see now, how we defined the energy measurement process of a query in MongoDB (Fig. 4a). In general terms, a query (A) is sent for execution (a) and analysis to the environment for energy consumption assessment (c). Then, using a specific meter (c), the measurement of the energy consumption is made (c1) from the time the query starts to run (b). Finishing the execution of the query also finishes the measurement of its energy consumption (c2). Then, the measurement process ends, returning a set of documents as the result of the query along with some energy consumption data gathered during the execution of the query. However, we can see a more low level view presenting the main tasks of the querying and energy measurement processes (Fig. 4b). As we can see, the process starts sending the query (A) for the energy consumption evaluation environment and preparing its execution (a). Then, the system gets the most efficient execution plan (b) for the query in the caching system and initiates the execution of the query (c) and, simultaneously, the energy meter starts measuring the energy consumption of the query (d). When the execution of the query ends, the energy measurement process is terminated, and the querying results are joined (e), integrating the documents that satisfy the query and the correspondent energy consumption data. Finally, the system returns the results (f) in a JSON file (B).

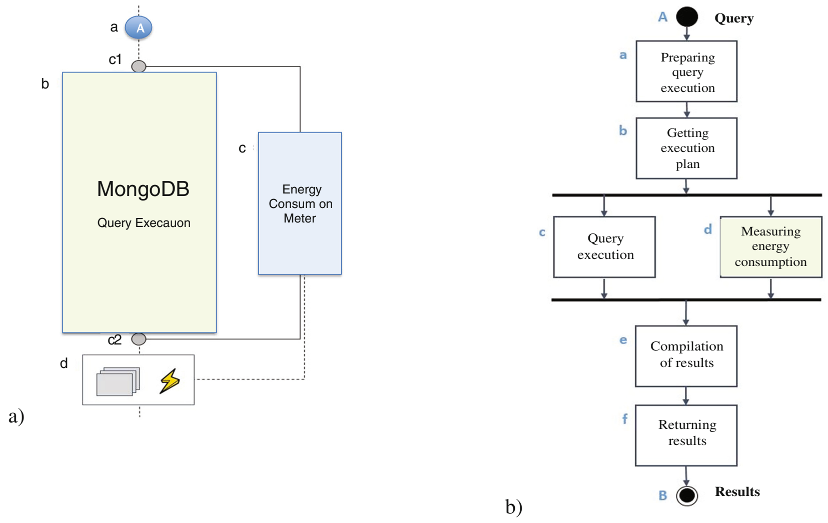


Fig. 4. A schematic view of the energy consumption process.

To support the execution and energy consumption tests we used an operational system integrating four processors Intel(R) Core(TM) i3-2100 CPU @ 3.10 GHz, each one with 8 GB of RAM, using as operating system the Ubuntu 14.04.4 LTS, and as document store management system MongoDB, version 3.2.6. The energy consumption meter was implemented in RAPL Power Meter using the driver RAPL (Running Average Power Limit) from Intel. For programming the services of the energy consumption meter we used jRAPL, a variant of the JAVA programming language for the RAPL Power Meter. Using the jRAPL, the results of the evaluation of the energy consumption of the query are a combination of three distinct values, which are related namely with the energy consumption of the: RAM, processing unit, and evaluation process programs [9].

In order to implement the energy measurement process of a query in MongoDB it was necessary to create a specific mechanism to start the measurement process, execute the query, obtain the results (related to the execution of the query and its energy consumption) and finally, finish the process. All this was done in jRAPL. Thus, to determine the energy consumed by a query with a nice accuracy level, we executed 25 times all the queries we designed for our tests, always measuring each time the energy consumption. The final energy consumption of the query was calculated using the average of all energy consumption values recorded each time the query was executed. Despite the meter provide the total value of the energy consumption of all the queries, we need to have into account that this value can be influenced by other sources that usually affect the operation of the system, and of course its energy consumption. In order to withdraw the amount of energy absorbed by these sources, we had to analyze the energy consumption of the system in an idle state, having no query or application running. Thus, the system was monitored 100 times, during 10 s each monitoring time, and then the average energy consumption value was calculate taking into account all the measurements made each time the system was monitored in idle status.

To demonstrate the utility of the consumption model we developed, and evaluate the energy consumption of a given query in a document store, we prepared ten distinct queries. Then, we grouped them two by two. Each group of two queries returns the same results, which means that the queries are equivalent, but are executed in different ways. For this, one query group was implemented using Map/Reduce (M/R), and the other was implemented using the MongoDB's aggregation functionality (Aggregation). All the queries were used over the same document store in a MongoDB system, containing a collection of 150 000 documents produced by the TPC-H [14] and converted for a format compatible with MongoDB, mapping the relational data to a document store with embedded documents containing replicated data along different documents. This is necessary due to the fact that it is not possible to join documents in MongoDB. To validate the consumption model we developed, both in terms of execution time and energy consumption, we selected a specific set of queries that were used over a document store containing a single collection ("LineItems") of 150 000 documents—Fig. 6 presents an example of one of the queries we selected, which returns a price summary report.

```
Query1 M/R

var red = function(doc, out) {
out.count_order++;
out.sum_qty += doc.quantity;
out.sum_base_price += doc.extendedprice;
out.sum_disc_price += doc.extendedprice * (1 - doc.discount);
out.sum_charge += doc.extendedprice * (1 - doc.discount) * (1 +
doc.tax);
out.avg_disc += doc.discount;};
var avg = function(out) {
out.avg_qty = out.sum_qty / out.count_order;
out.avg_price = out.sum_base_price / out.count_order;
out.avg_disc = out.avg_disc / out.count_order;
};
db.lineitems.group( {
key : { returnflag : true, linestatus : true},
cond : { "shipdate" : {$lte: 19980801}},
initial: { count_order : 0, sum_qty : 0, sum_base_price : 0,
sum_disc_price : 0, sum_charge : 0, avg_disc : 0},
reduce : red,
finalize : avg
});
```

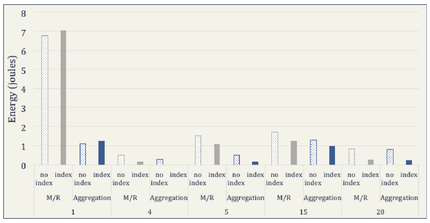
Fig. 5. An example of a query in MongoDB M/R – Query1.

All queries follow the model presented in Fig. 5, M/R and another in Aggregation and then were tested and analyzed their execution plans and energy consumption. As mentioned above, in MongoDB resort to explain command. However, explain returns a JSON document and to be more easily interpretable JSON was converted to a visual level. However, the explain command is only valid in the aggregation model as M/R does not. In terms of consumption, for each query is returned a list that discriminate three types of consumption, namely memory consumption, CPU consumption and package execution consumption. In Fig. 6a and b, respectively, we can see the results for the three types of consumption of Query 1 for the case of use of aggregate framework and Map/Reduce respectively. With results equal to these and for all queries used for testing was possible to reach a representative chart of all queries executions (Fig. 7).

a)
```
Power consumption of dram: 0.00538629999937256 power consumption of cpu:
0.510784899999998 power consumption of package: 0.661329599999953 time:402
```

b)
```
Power consumption of dram: 0.0290893000004871 power consumption of cpu:
3.13843540000016 power consumption of package: 3.94761509999989 time:2186
```

Fig. 6. Consumption results for Query 1 using aggregation (a) and using M/R (b).

a) 

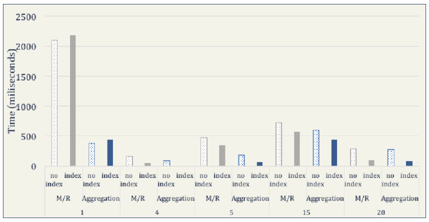
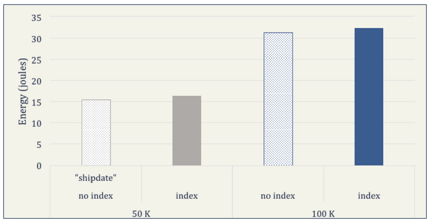
b) 

Fig. 7. Energy consumption (a) and execution time (b) of all queries.

Figure 7 presents the energy consumption of all the queries used in this work. In this figure, we can see that all the queries that have a lower energy consumption, except query 1—the case of Map/Reduce—which had a great energy consumption. Furthermore, the use of indexes in the queries also allowed for saving energy, except in the case of query 1, just as before. As expected the use of indexes has clear advantages in having better query execution time (except again for test 1), but would be expected to their use would a disadvantage. It is something that occurs with the use of indices inserts become slower and more costly the energy level. To verify this property we used two tests, one of 50 000 insertions and another 100 000. These two tests were carried out for two cases as and without indexes in the collection, before insertion already in the collection 100 000 documents—all the tests were performed 25 times each.

a) 

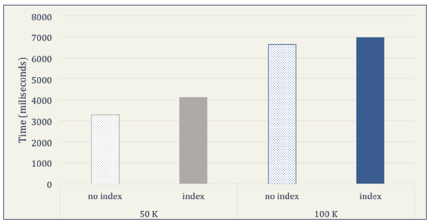
b) 

Fig. 8. Energy consumption (a) and execution time (b) of insert operations.

Regarding the execution time, it seems that the use of indexes has some impact, as expected. More specifically, the impact we measured was about 25.6% and 4.5%, respectively, for the execution of 50 000 and 100 000 insert operations in the document store. This is due to the work that is required to add these new documents to the data structure (a tree) that contains the correspondent indexes. In Fig. 8, we can see that the use of indexes has an impact on the energy level for insert operations as it happened for the execution time. This impact is about 5.6% and 3.4%, respectively, for the same insert operations cases done before −50 000 and 100 000 inserts. With this, we verified the advantage acquired when we done the inserts using a document store—an associated negative weight. With regard to the energy consumption, despite the higher consumption this is not very significant. Regarding the execution time for a large number of insert operations, it has a reduced impact. In fact this impact is more relevant in the document store if compared to the one of common search queries, as it happens usually in any database. However, in a regular day of operation, in a database system insert operations are much lower than the number of search operations.

5 Conclusions

Today, in all branches of activity, companies are discussing and promoting energy consumption measures and actions in order to have more efficient energy consumers. In information technology such concern is rising each passing day, with particular emphasis in databases. See, for instance, the case of Google. The Google's search engine makes a day about 3.5 billion searches, and an abysmal number of 1.2 thousand of billions searches per year, worldwide. Thus, if there is a way to save a tiny amount of energy, even small, for each of these searches, it will be possible to transform that saving in a huge amount of energy at the end of a year of activity. With this, companies with a similar magnitude to Google can reduce significantly their energy costs and contribute to the reduction of their ecological footprint—in the case of Google this is already a concern.

In this work we presented and described a tool we implemented for helping document stores administrators to assess the energy consumption of the different queries that are usually launched in these systems. The way two different queries producing equal results interact with the engine of a document store may cause a significant difference in the energy consumed by each one of them. Modifying a query to optimize its energy consumption could not be always possible, since such improvement can worse its own execution time. Thus, sometimes the differences in the implementation of a query do not allow for any kind of improvement in terms of energy consumption, due to some kind of application or operational requisites. However, this was something that was not observed in this work. We verified that the fastest queries were also the most energy efficient, which is a very encouraging result. Despite this, more extensive tests must be done to reinforce these results. Finally, we need also to do all the tests performed in this study using a different document store management system, as well as a different application context with other operational conditions, in order to verify the findings we got here—shorter execution time, lower energy consumption. However, this cannot be generalized at the moment to all document stores management systems, since they all have different kinds of implementations.

Acknowledgments. This work has been supported by COMPETE: POCI-01-0145-FEDER-007 043 and FCT—Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/ 00319/2013.

References

1. Agrawal, R., Ailamaki, A., Bernstein, P., Brewer, E., Carey, M., Chaudhuri, S., Doan, A., Florescu, D., Franklin, M., Garcia-Molina, H., Gehrke, J., Gruenwald, L., Hass, L., Halevy, A., Hellerstein, J., Ioannidis, Y., Korth, H., Kossman, D., Madden, S., Magoulas, R., Ooi, B., O'Reilly, T., Ramakrishnan, R., Sarawagi, S., Stonebraker, M., Szalay, A., Weikum, G.: The claremont report on database research. Commun. ACM **52**(6), 56–65 (2009)
2. Belo, O., Gonçalves, R., Saraiva, J.: Establishing energy consumption plans for green star-queries in data warehousing systems. In: Proceedings of the 2015 IEEE International Conference on Green Computing and Communications (GreenCom 2015), Sydney, Australia, pp. 11–13 (2015)
3. Gonçalves, R., Saraiva, J., Belo, O.: Defining energy consumption plans for data querying processes. In: Proceedings of the 4th IEEE International Conference on Sustainable Computing and Communications (SustainCom 2014), Sydney, Australia, 3–5 Dec 2014
4. Harizopoulos, S., Shah, M., Meza, J., Ranganathan, P.: Energy efficiency: the new holy grail of data management systems research. In: CIDR (2009)
5. Khargharia, B., Hariri, S., Yousif, M.: Autonomic power and performance management for computing systems. In: Proceedings of 2006 IEEE International Conference on Autonomic Computing, pp. 145–154 (2006)
6. Kunjir, M., Birwa, P., Haritsa, J.: Peak power plays in database engines. In: Proceedings of the 15th International Conference on Extending Database Technology, pp. 444–455 (2012)
7. Lang, W., Patel, J.M.: Towards eco-friendly database management systems. In: CIDR (2009)
8. Lang, W., Kandhan, R., Patel, J.M.: Rethinking query processing for energy efficiency: slowing down to win the race. IEEE Data Eng. Bull. **34**, 12–23 (2011)
9. Liu, K., Pinto, G., Liu, Y.: Data-oriented characterization of application-level energy optimization. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pp. 316–331 (2015)
10. Moniruzzaman, A., Hossain, S.: Nosql database: new era of databases for big data analytics-classification, characteristics and comparison. Int. J. Database Theory Appl. **6**(4), 1–14 (2013)
11. MongoDB: MongoDB. https://www.mongodb.com (2016). Accessed 19 Aug 2016
12. Person. L.: World NoSQL Market—opportunities and forecasts, 2013–2020 (2015)
13. Rasmussen, N.: A.p.c. determining total cost of ownership for data centers and network room infrastructure. www.apcmedia.com/salestools/CMRP-5T9PQG/CMRP-5T9PQG_R4_EN.pdf(2011). Accessed from 3 Sept 2016
14. TPC-H: TPC-H, An ad-hoc decision support benchmark. http://www.tpc.org/tpch/(2016). Accessed from 19 Aug 2016