A Database System with Amnesia

Martin Kersten
CWI
Amsterdam, The Netherlands
martin.kersten@cwi.nl

Lefteris Sidirourgos
CWI
Amsterdam, The Netherlands
lsidir@cwi.nl

ABSTRACT

Big Data comes with huge challenges. Its volume and velocity makes handling, curating, and analytical processing a costly affair. Even to simply "look at" the data within an a priori defined budget and with a guaranteed interactive response time might be impossible to achieve. Commonly applied scale-out approaches will hit the technology and monetary wall soon, if not done so already. Likewise, blindly rejecting data when the channels are full, or reducing the data resolution at the source, might lead to loss of valuable observations.

An army of well-educated database administrators or full software stack architects might deal with these challenges albeit at substantial cost. This calls for a mostly knobless DBMS with a fundamental change in database management.

Data rotting has been proposed as a direction to find a solution [10, 11]. For the sake of storage management and responsiveness, it lets the DBMS semi-autonomously rot away data. Rotting is based on the systems own unwillingness to keep old data as easily accessible as fresh data. This paper sheds more light on the opportunities and potential impacts of this radical departure in data management. Specifically, we study the case where a DBMS selectively forgets tuples (by marking them inactive) under various amnesia scenarios and with different implementation strategies. Our ultimate goal is to use the findings of this study to morph an existing data management engine to serve demanding big data scientific applications with well-chosen built-in data amnesia algorithms.

1. INTRODUCTION

Big Data is fueled by the ease at which data can be collected and stored for subsequent analysis. It has become so easy to hoard massive amounts of data, that most of it will be stored away and never be looked at after ingestion. The (management) costs for storing such vast amounts of data becomes a burden for research institutes and businesses alike. Costs that are not only increased because of hardware purchases and energy bills, but also because the *utility value* of the data quickly diminishes over time. Too much data, or too old data, becomes irrelevant because either one sees the same observations over and over again, or data becomes stale and of less economic or scientific value. What was relevant before, becomes irrelevant and costly to keep around or digest. Data analysts are better served by deleting or moving irrelevant data out of the critical query execution path.

Furthermore, there is an end-user misconception that we can afford to keep everything around forever in (cold) storage. This is not true and only half of the story. Although storage for fast growing databases is cheap (e.g., AWS Glacier charges $48 per TB/year in 2016) using this data becomes prohibitively more expensive over time, both money wise and by input latency (AWS Glacier data retrieval cost is $ 2.5- 30 per TB and can take up to 12 hours). Although subsequent Cloud processing power is available, it is usually placed far from the cheap secondary storage. Thus, maintaining a wisely chosen subset of active records in faster memory is needed to cut down on the cost of processing.

A naive answer to the problems of storing big data and the cost of scaling-out data management is to discard data upstream, i.e., drop most data at the source where they are produced. For example, in a scientific instrument the sensors transmit with smaller rates than what they are capable of, or business events are logged per intervals instead of continuously. But this approach suffers from losing potentially important data; valuable information might be blindly ignored. We need better ways to either *digest* data quickly or *forget* data when we can afford it without losing too much information.

Data rotting has been proposed as an alternative technical solution [10, 11]. This vision for a new generation of database systems is rather radical. It challenges common belief that a prime purpose of a (scientific) database warehouse system is to store data forever, and not to let it rot away. However, if we consider the fundamental design principal of a database system to be to retain information and to make it quickly accessible, then guaranteeing there will be no *data loss* is just a direct consequence of that initial principal. Information availability and data storage are considered interchangeable concepts, but this is not to be always the case. In the Big Data era, where data is overwhelming, we can define strategies to *forget* many data items and still *retain the information*.

For example, if you are only interested in the average value over a series of observations, then you can safely drop two tuples that together do not affect the average measured. The average information is guaranteed with a smaller database footprint. If you are interested in a profile analysis over your data, then identical tuples are not necessarily needed. Maintaining a simple count on the number of occurrences of the same observation suffices. Evidently, observations that are constrained by a Data Privacy Act should be forgotten within the legally defined time frame.

In this paper, we study the effect of *forgetting* data in the context of a given database workload setting. How much information loss would there be and how bad could it get. Given these observation,

8th Biennial Conference on Innovative Data Systems Research (CIDR '17) January 8-11, 2017 , Chaminade, California, USA.

we illustrate different strategies to retain information while forgetting tuples. To quantify the amount of acceptable information "retention" we define a number of metrics and present the simulated results.

Since the effects of data amnesia strongly depends on the data semantics, its distribution and update/query workload, we do not foresee a simple mathematical information theoretic model to come at rescue. At least, not at this stage of the research. Therefore, to gain quick insight we use a DBMS skeleton implementation. We implement a series of prototypical data distributions, query workloads, and amnesia strategies.

The simulation studies confirm our hypothesis that a careful choice of amnesia technique can keep the storage footprint within tight bounds for a given query/update workload. With the results of this study, theoretical micro models may come into view.

Finally, the question remains on what happens to forgotten data. A DBMS might be as radical as to delete all data being forgotten. A lighter and more feasible option is to *stop indexing* the forgotten data. That is to say that a complete scan will fetch all data, but a fast index-based query evaluation will skip the forgotten data. A more cost-effective option is to move forgotten data to cheap slow cold-storage. Finally, a possibly poor information retention approach would be to keep a summary, i.e., a few aggregated values (min, max, avg) of all the forgotten data. This will reduce the storage drastically but the DBMS will only be able to answer specific aggregation queries without making available any other details.

As far as we are aware, this is the first attempt to get a handle on the complications that arise from *data amnesia*. This study identifies the fundamental design choices to transform an existing system to elicit the effects of forgetting data in a real life application.

Furthermore, it uncovers a barren landscape for database research, covering new ways for data storage, indexing and (adaptive) query processing, and taking statistical sampling based approaches to a new level of effectiveness.

We next describe the setup of our Data Amnesia Simulator, its architecture, workload and information metrics. In Section 3 we introduce a snippet of the possible strategies for forgetting data in a controlled way. They are used to generate our experimental findings in Section 4. Section 5 provides our research vista on the landscape of database amnesia techniques.

2. A DATABASE AMNESIA SIMULATOR

Given the lack of theory and experience on the impact of *database amnesia* in the context of a given data distribution and workload, we developed a small-scale *database amnesia simulator*. It is solely geared at acquiring quick insights, scouting the landscape of amnesia techniques without concern of all the bells-and-whistles of a complete DBMS.

2.1 Database Architecture

The simulator is a skeleton of a columnar DBMS written in C [1]. Its schema is fixed and consists of a collection of columns. Keeping things simple, we only consider tables filled with integers in the range $\mathcal{R} = 0, \ldots,$ `DOMAIN` with a predefined distribution.

We expect that database amnesia is strongly influenced by the data distributions and query workload. For the experiments reported in this paper, we use the following distributions to reflect real life cases:

- *serial*, to model both an auto-increment key and a temporal order of tuple insertions,
- *uniform*, to model data distributions mostly found in benchmark tables such as TPC-H,
- *normal*, to model normal data distributions around the `DOMAIN` range mean with a standard deviation of 20%.
- *skewed*, taken from a Zipfian distribution to model a more realistic scenario, such as the Pareto principle (i.e., 80-20 rule) where some (random) values are dominant,

For each table T, we keep a record of *active* and *forgotten* tuples. It provides a basis for comparing query results with and without amnesia. The granularity is purposely kept to a single record since we are mostly interested in trends rather than speed. In a full-blown system the marking can be aligned with disk-blocks, file-segments, row identifiers, or value ranges. Another strategy for large production systems is to forget the tuples by either moving them to cold storage or physically remove them from the index structures used at query evaluation.

In our experiments and at any point in time, the database storage requirements in number of tuples in each table, remains constant and it is equal to `DBSIZE`. In this way we simulate a tight storage budget constraint. In a more realistic scenario, one might want to constraint the *growth* instead of the size of the database. For example, if a database starts by using half of the available RAM, do not let it grow beyond the 90% mark. This will be achieved by simply forgetting more and more tuples as you reach the upper limit. Another consideration will be bounding the processing time for the workload, but this is left for future work on data amnesia.

2.2 Query Workload

Forgetting data can be harmful for it leads to loss of information. However, it also strongly depends on the application. If we are only interested in aggregated summaries over scientific data, then missing a few tuples may not be too bad. The error introduced vanishes behind the noise encountered by taking the observations. Contrary, if the data is about unique standing payments, then forgetting such information would be a big inconvenience. Therefore, ideally, knowledge about all queries and their frequency to be ran against a database would make it possible to identify if and how long a tuple is active before it can be safely forgotten. Collecting such statistics is a good start to assess what data amnesia an application can afford.

For the simulator in this work, we only focus on typical database benchmark query templates. Given the unbounded space of SELECT-PROJECT-JOIN queries we carve out a well understood subspace for our simulation. The base line for our experiments are simple range queries over a database table, controlled by a *selectivity factor* $\mathcal{S}$. A selectivity factor $\mathcal{S} = 1.0$ would expose all forgotten tuples as an imprecision of the result set. In other words, if a range query requests all tuples, then the answer will be incomplete exactly as much as the number of forgotten tuples. Conversely, a range query with a small selectivity factor $\mathcal{S} = 0.01$ is less susceptible to forgotten tuples. There is a smaller chance a forgotten tuple to be part of the query range predicate, especially if the amnesia strategies are picked correctly.

The second query group involves simple aggregations over subranges, e.g., the average (AVG). Aggregations are more robust against forgotten tuples. For example, any pair of tuples with antipodal values around the average if removed won't change the outcome. Moreover, the probability of a forgotten tuple to greatly distort the average value depends on the standard deviation of the value distribution.

Clearly, data distribution and query format can lead to different forms of amnesia and different levels of information loss. Next, we

[1]The simulator code is available from the authors

define a number of metrics to measure how much information is retained after a batch of tuples has been forgotten.

2.3 Information Precision Metrics

Database amnesia leads to incomplete result sets and approximated aggregate values. In our analysis, the simulator collects the following metrics to quantify the information loss, or expressing it in a positive way, the *query precision*, after inserting $\mathcal{F}$ new tuples and forgetting $\mathcal{F}$ other tuples to keep the size of the database constant.

- $R_{\mathcal{F}}(Q)$ number of tuples in query result Q,
- $M_{\mathcal{F}}(Q)$ number of tuples missed in query result Q,
- $P_{\mathcal{F}}(Q)$ is the query precision such that $P_{\mathcal{F}}(Q) = R_{\mathcal{F}}(Q)/(R_{\mathcal{F}}(Q) + M_{\mathcal{F}}(Q))$
- and E is the error margin defined as $E = \text{avg}(R_{\mathcal{F}}(Q))/\text{avg}(R_{\mathcal{F}}(Q) + M_{\mathcal{F}}(Q))$ computed over an entire batch of Q queries.

Observe that the simulator only marks tuples as either *active* or *forgotten*, which gives us the opportunity to precisely calculate the query precision. Furthermore, we assume that all queries only consider the data domain as stored in the table. Thus, any range query will produce a result set based on the values seen so far, albeit it may attempt to retrieve forgotten tuples.

In our experiments we assume a query dominant environment, where a batch of queries is followed by a batch of updates, immediately followed by applying an amnesia algorithm to guarantee that the database is always of `DBSIZE`. The metrics are reported by averaging over a batch of 1000 individual queries fired against the incomplete database.

3. DATA AMNESIA

An amnesia strategy can be based on many application factors and features of a DBMS. Here we focus on amnesia as a controlled random process and by studying the effects of learning which tuples are of interest.

3.1 Temporal Biased Amnesia

A natural first dimension is to consider the order in which tuples have been added to the database. This creates a time-line over which a sliding buffer of size `DBSIZE` defines the active tuples. Much like a *FIFO* strategy works for buffer management. Keeping this buffer at the head of the time line only shows results based on fresh data. Streaming database applications are good examples for this kind of amnesia, where all you can see is what's in the stream buffer. We refer to this scenario as the *FIFO-amnesia* algorithm.

In database amnesia we want to go a step further. The tuples retained in the database are spread over a larger segment of the time line and tuples are removed using a randomized process. For example, after each update batch we uniformly select tuples to be removed. This approach is similar to the reservoir sampling technique [19]. At any round of amnesia, a tuple has the same probability to be forgotten, but older tuples have been a candidate to be forgotten multiple times. We refer to this scenario as the *Uniform-amnesia* algorithm and it serves as an easy to understand baseline.

A refinement is to consider roughly two amnesia classes: *retrograde* and *anterograde* amnesia. In retrograde amnesia one can't recall old memories, thus translated to database amnesia, older tuples are more easily forgotten from the database. *FIFO-amnesia* is an example of retrograde amnesia. Contrary, in *anterograde* amnesia, one can not accumulate new memories easily. We implement this kind of amnesia by choosing randomly mostly recently added tuples to be forgotten. This strategy prioritize historical data, and a new piece of information is only remembered if it appears too often.

3.2 Query Based Amnesia

An alternative for these randomized algorithms is to take the interest of past queries into account. For example, a tuple that appears often in a query result might be considered more important and should not be forgotten easily. To study this behavior we extend the tables with the frequency of access for each tuple and after each batch of inserts, tuples are forgotten with probability analogous to their frequency.

Care should be taken not to drop most recently added tuples, which would result in an anterograde amnesia behavior. For that, we use a high water mark approach, where tuples are forgotten when they are not frequently accessed but also been part of the database long enough. We refer to this approach as rot.

A totally opposite approach would be to forget data that has been used too frequently. The motivation for this policy would run as follows. Assuming a database that shifts around data, by transforming and summarizing its context. If a tuple has been accessed too many times, then its role should be reconsidered. In other words, no data should continue to appear in a result set, if that data has not been curated, analyzed, or consumed in any other way.

3.3 Spatial Biased Amnesia

Another way to model the amnesia processes is to mimic nature more closely using a forgetting algorithm fit with a bias towards areas already "infected with mold" because of lack of freshness. It aligns also with the observation that hardware errors on magnetic disks are spatially highly correlated, usually caused by disk inactivity due to lack of interest for the data stored on those areas. This amnesia strategy is labeled as *area* based. It is implemented by keeping a list of areas of forgotten tuples, say K and set n to a value between $1, \ldots, K + 1$. If $n = K + 1$, then we start new mold for a tuple by randomly selecting a new active starting point. Otherwise, we look into the database tiling and extend the n-th area of forgotten tuples in either direction.

4. EVALUATION

In this section we provide a snippet of the experiments around implementation of the amnesia algorithms which deemed relevant. They provide an outlook on a fast and adventurous landscape of algorithm research.

The simulator spans over a sizable search space. The arrival order of tuples forms the basis for retrograde and anterograde amnesia, the data data distributions and query workload influence the query precision, and the various amnesia algorithms can be extended in multiple directions.

4.1 Data amnesia map.

A key parameter in our first rounds of experiments is to keep the database footprint fixed. This size could be as large as the main memory available from a virtual machine in the Cloud. By limiting the database footprint we restrict access to remote file storage, which is slow and expensive. In this setting, the remote file store can be consider the cold storage.

Our first goal is to visualize which portion of the database is retained over time and under different amnesia strategies. Figure 1 illustrates the distribution of still active tuples after a sequence of 10 update batches under all amnesia algorithms except the rot amnesia. The brighter the colored area is, the more tuples are still

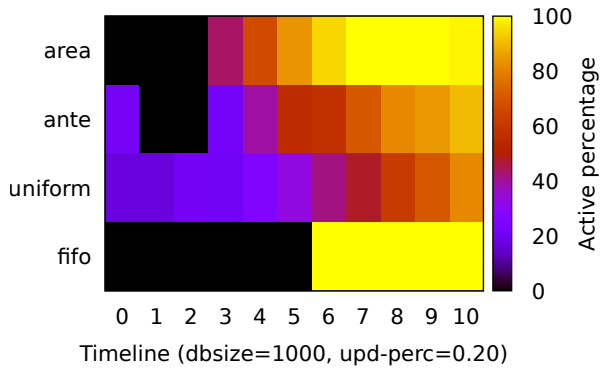

Figure 1: Database amnesia map after 10 batches of updates

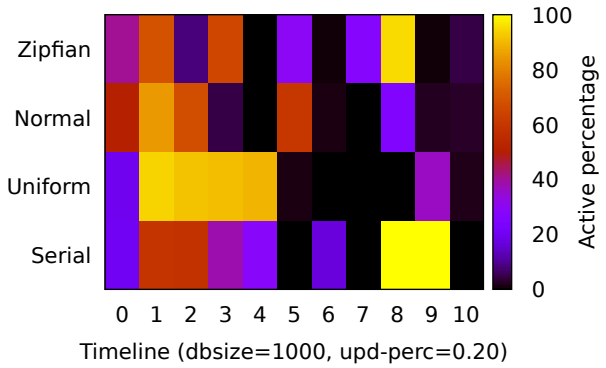

Figure 2: Database rot map after 10 batches of updates

accessible after a long update run followed by range queries and aggregate calculations. For the four different amnesia strategies of Figure 1, the data distribution plays no role, only the relative position of each tuple in the database storage space. A fifo amnesia, will only highlight the latest tuples, since all old data have been forgotten. The uniform amnesia strategy, as expected, produces a uniform coloring which is brighter at the end because the newer the tuples, the less opportunities they had to been forgotten. The anterograde amnesia strategy, retains most of the data at point 0 (initial data of the database), and then forgets all updates, starting from the oldest ones. If we were to continue the update batches, the black hole would increase to include more recent updates. Finally, the area amnesia strategy, which chooses at random places to start a hole and expand them, shows an affect witch resembles a uniform-fifo combination. Naturally, the oldest the data the more holes they will contain, resulting to a fifo effect, but the newer the data the more uniform will be.

The rot amnesia strategy, depends on how fresh are the data. Freshness is measured by the frequency of appearing in a result. Since all range and aggregate queries are the same in our experiments, the data distribution is the differential factor for rotting. Figure 2 shows the different effect of rotting for serial, uniform, normal, and zipfian distributed datasets. Figure 2 illustrates that the data distribution in combination with the amnesia has a strong impact on what you retain from the past.

The observed behavior exhibits similarities to common buffer schemes and database samples. A critical issue, however, not addressed as such in database sampling, is that we assume a stream of updates where we can learn from the past use and, hopefully, it acts as a good predictor of the future. Or, perhaps we don't need to learn if the query load does not require it from a precision perspective.

4.2 Range query precision

The next question is to gain insight in the effect of the amnesia policy on the query precision. The baseline experiment is to consider its effect on range queries. However, here we have to consider the query distribution as well. If the user is mostly interested in the recently inserted data then a FIFO style amnesia suffice. For this round of experiments we assume that the query workload addresses all tuples ever inserted into the database. This provides an upper bound on the accurracy under an amnesia strategy.

We hypothesized that the amnesia algorithm would have a sizable impact on precision and is influenced by both the data distribution, volatility and query load. The volatility captures the amount of data being forgotten at each intermediate stage. We experimented with both low (10%) and high update volatility (80%). Furthermore, we used a uniform distribution of the queries over all data being inserted.

Figure 3 illustrates the results from range queries with a Normal and Zipfian data distribution. The range query generator selects a candidate value v from all active tuples and constructs the range where attr >= v- 0.01 * RANGE and attr < v + 0.01 * RANGE where RANGE is in the range 0 to the maximum value seen up to the latest update batch.

As expected the precision drops quickly over time as more and more information is forgotten. Surprisingly, though, the area rotting behaves differently. It is biased to increasing an area, which means that a smaller fragment of range queries is affected. The data distributions has some effect but converges to the same values in the long run.

Overall, the area and anti- policies seem to retain precision better. Increasing the selectivity factor does not improve the precision, because it affects the complete database, active and forgotten.

4.3 Aggregate query precision

Aggregate operations have become more important in data analytics then the joins. They are a challenge for any database system, because mostly the subset against which the aggregate operator is ran is not a priori known. Fortunately, they also are a kind of mini summaries whose precise value is not always needed. This gives space to forget tuples and only keep track of the error bounds and influence on the variance.

In the Simulator we studied their behavior with and without a range predicate over the underlying table. The former reflects the maximum information loss to be expected. The latter would show the effect in daily life, where the focus of aggregation can be directed to a specific part of the database, e.g. the fresh data.

Aggregate queries were expected to be less influenced by the amnesia algorithms. To study this, we increased the experimental run length and study the query SELECT AVG(a) FROM t. To our surprise the differences were marginal and the graphs came out similar to Figure 3. This hints towards a simple mathematical model to determine the precision, i.e. how many update batches have been processed.

4.4 The scope of amnesia algorithms

Evidently the base line experiments just provide a glimpse of the query precision in a database amnesia setting. It calls for a much broader analysis of the dependencies between data distribution and query workload. Better application specific amnesia algorithms is another area for innovative research. The simulator, a relatively small C-program, can be readily extended to serve this quest.

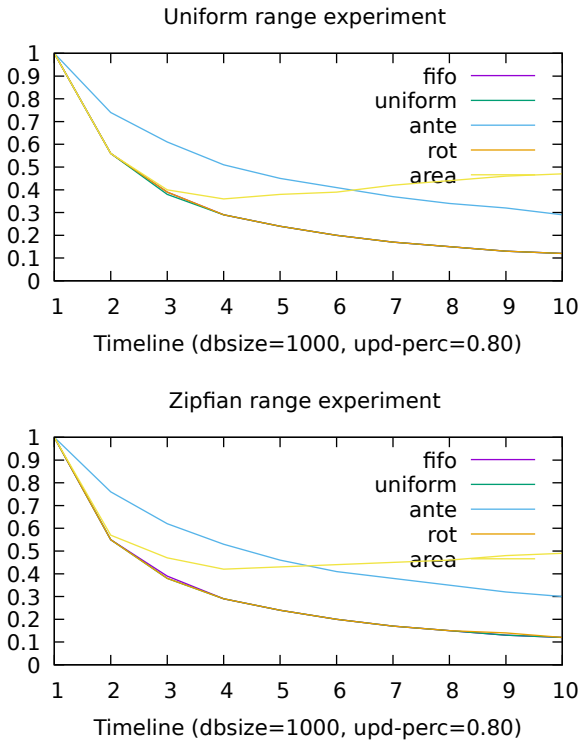

Figure 3: Range query precision ($v \in 0 .. max$)

To illustrate, the query patterns semantics can be exploited further. For example, the average query could be used to identify pairs of tuples to be forgotten instead of a single one. It would retain the precision as long as possible.

Alternatively, amnesia may be aligned with the data distribution of present and past. That is, we attempt to forget tuples that do not change the data distribution for all active records. Keeping the two distributions aligned as much as possible is what database sampling techniques often aim for [7]. However, in our case the data distribution evolves as more and more tuples are ingested (and forgotten). This means that the data distribution might change.

In a full fledged system, auxiliary data structures are also a good candidate for being (partially) forgotten. For example, indices improve the query processing, but also consume quite some space. They can be easily dropped, and recreated upon need, to reduce the storage footprint. This technique is already heavily used in MonetDB without the user turning performance knobs.

Data compression can be called upon to postpone the decisions to forget data. And once needed, how to ensure the least loss of information. A refinement is to consider partial indices, such as Block-Range-Indices.

After a query has been executed we know both its interest in the database portion and the cost of the relational algebra components. An alternative is giving preference to ditching tuples that cause an explosion in either processing time or intermediate storage requirements.

Instead of user defined partitioning schemes, it might be worth to study amnesia in the context of adaptive partitioning. Each partition can then be tuned to provide the best precision for a subset of the workload.

5. RELATED WORK

Backing-up and removing portions of a database has traditionally been the responsibility of a DBA. The technological barrier between him and the application owners can be large, which easily leads to calling for more capital investments or increasing the Cloud services to run HDFS-like applications. A partitioning of the database by tuple arrival age also helps in removing old data. The data amnesia algorithms introduced in this paper are closely tight with the DBMS itself and address the core question: what to retain and for how long?

In our simulator we marked tuples as (in)active to obtain a quantifiable measure. In a real system, we could move the inactive tuples to another level of the storage hierarchy, often called *cold* storage [12, 13, 14]. However, our view on the role of forgotten data is different from that of cold data in general [4]. Cold data can still appear in the result set, but it is rather slow to fetch and process. Aging/cold data research focus aims at maximizing the hot data in the primary store, i.e., optimizing the database buffers. In a database with amnesia, however, data is *forgotten* and will never show up in query results, unless the user takes the action and recover a backup version of the database from cold storage explicitly.

Closely related to our work is also recent research on sample-enabled approximate query processing for interactive query sessions [1, 16, 3]. In this line of research, according to the time and accuracy constraints placed by the user, samples are drawn from the entire dataset such that those constrains are met. These designs assume access to the *entire* dataset or at least a priori constructed statistics over the entire dataset. A sample can be re-drawn and the query answer re-shaped if the results are not satisfactory. However, this is not the case for data amnesia. We are interested in strategies that assume there is no reference to the original and complete view of information, thus our approach is even more fuzzy than sampling-based techniques.

Semantic database integrity creates another challenge for amnesia strategies. For example, foreign key relationships put a hard boundary on what we can forget. Should forgetting a key value be forbidden unless it is not referenced any more? Or should we cascade by forgetting all related tuples? These questions can only be answered by taken into account the given semantics involved per application.

Recent studies [6, 2] use neurological inspired models of the human short term memory system to assess the recall precision in the context of forgetting data. The results show that amnesia algorithms based on “human forgetting inspired heuristics” can be an effective tool for shrinking and managing the database. In our simulator we stick to the probabilistic features of the database and the characteristics of a given query load. However, it is conceivable that modern AI learning techniques can provide hooks to improve the amnesia algorithms.

A special, but highly relevant approach is to counter the forgetting information process by turning portions of the database into summaries. They can take the form of traditional compression schemes, or for the more adventurous, replacing portions of the database by micro-models [15]. Database summaries and materialized views are a potential good medicine crafted by the user against amnesia [17]. However, the first concern should be to understand how the amnesia affects the quality of the query workload before diving into these semantic rich models.

Finally, we wish to combine data and application domain specific knowledge to create well tailored amnesia strategies. For example, in a database with historical weather information, data from areas that have constant weather patterns can be forgotten in a few weeks time, where for areas that exhibit strange meteorological phenom-

ena the data should be kept for longer periods. Not to mention the need for database systems that provide techniques for controlled forgetting private information. Snapchat [18] is the proof of such need. Relevant early work in this context has been studied from the perspective of *vacuuming* the database with a strong focus on the tuple's age in a temporal database context [9, 5].

6. CONCLUSION

In this paper we propose *data amnesia* as a new feature for a modern DBMS. The amnesia algorithms are considered an integral part of a DBMS kernel with limited tuning knobs, because they are there to enable the DBMS to perform best within the resource bounds given. Amnesia addresses the ever expanding data sizes in business and scientific application, which may become too voluminous for interactive processing or their Cloud-based parallel processing too expensive.

Database amnesia forces the DBA and the users to seriously consider the cost of keeping data available forever. The price of more information retention may not outweighs the added return on investment in storage and processing power. This means that a proper choice of the data amnesia policy is required, or a timely action should be taken to compress the data into meaningful summaries.

As far as we know, this paper is the first experimental study of the impact on the quality of results when a database ignores tuples because it "suffers from amnesia". It has been shown through a straightforward simulation that a careful balance of forgetting strategies given a query load, can improve the storage footprint without significant loss in information.

We merely scratched the surface of *data amnesia* and already encountered wide vistas of possible research directions. On our shortlist are similar studies to understand the impact of scale and taking into account processing time. The migration of *forgotten* data to cheap storage or removal from indexes in the context of a real DBMS is next. To no one's surprise, our preferred choice of a DBMS for extending our work will be MonetDB [8].

7. REFERENCES

[1] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, 2013.

[2] G. S. Bahr and S. L. Wood. The big data between your ears: Human inspired heuristics for forgetting in databases. In *2015 IEEE International Conference on Multimedia & Expo Workshops, ICME Workshops 2015, Turin, Italy, June 29 - July 3, 2015*, pages 1–6. IEEE Computer Society, 2015.

[3] J. Bisbal, J. Grimson, and D. Bell. A formal framework for database sampling. *Information and Software Technology*, 47(12):819 – 828, 2005.

[4] J. DeBrabant, A. Pavlo, S. Tu, M. Stonebraker, and S. Zdonik. Anti-caching: A New Approach to Database Management System Architecture. *Proc. VLDB Endow.*, 6(14):1942–1953, Sept. 2013.

[5] A. F. Dia, Z. Kazi-Aoul, and A. Boly. Extension de C-SPARQL pour l'échantillonnage de flux de graphes RDF. In C. de Runz and B. Crémilleux, editors, *16ème Journées Francophones Extraction et Gestion des Connaissances, EGC 2016, 18-22 Janvier 2016, Reims, France*, volume E-30 of *RNTI*, pages 159–170. Hermann-Éditions, 2016.

[6] S. T. Freedman and J. A. Adams. Filtering data based on human-inspired forgetting. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 41(6):1544–1555, 2011.

[7] V. Ganti, M. L. Lee, and R. Ramakrishnan. ICICLES: Self-Tuning Samples for Approximate Query Answering. In *Proc. of the 26th VLDB*, 2000.

[8] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten. Monetdb: Two decades of research in column-oriented database architectures. *IEEE Data Eng. Bull.*, 35(1):40–45, 2012.

[9] C. S. Jensen. Vacuuming. In *The TSQL2 Temporal Query Language*, pages 447–460. 1995.

[10] M. Kersten. Big Data Space Fungus. In *Proc. of the 7th CIDR*, 2015.

[11] M. Kersten. Keynote: DataFungi, from Rotting Data to Purified Information. In *Proc. of the 32nd ICDE*, 2016.

[12] J. Levandoski and P. Larson. Identifying Hot and Cold Data in Main-Memory Databases. In *Proc. of the 29th ICDE*, 2013.

[13] L. Ma, J. Arulraj, S. Zhao, A. Pavlo, S. R. Dulloor, M. J. Giardino, J. Parkhurst, J. L. Gardner, K. Doshi, and S. Zdonik. Larger-than-memory Data Management on Modern Storage Hardware for In-memory OLTP Database Systems. In *Proc. of the 12th DaMoN*.

[14] C. Meyer, M. Boissier, A. Michaud, J. O. Vollmer, K. Taylor, D. Schwalb, M. Uflacker, and K. Roedszus. Dynamic and transparent data tiering for in-memory databases in mixed workload environments. In R. Bordawekar, T. Lahiri, B. Gedik, and C. A. Lang, editors, *International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures - ADMS 2015, Kohala Coast, Hawaii, USA, August 31, 2015.*, pages 37–48, 2015.

[15] H. Mühleisen, M. L. Kersten, and S. Manegold. Capturing the laws of (data) nature. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*. www.cidrdb.org, 2015.

[16] L. Sidirourgos, M. Kersten, and P. Boncz. SciBORQ: Scientific data management with Bounds On Runtime and Quality. In *Proc. of the 5th CIDR*, 2011.

[17] J. Skyt, C. S. Jensen, and T. B. Pedersen. Specification-based data reduction in dimensional data warehouses. *Inf. Syst.*, 33(1):36–63, 2008.

[18] Snapchat. https://www.snapchat.com/.

[19] J. S. Vitter. Random Sampling with a Reservoir. *ACM Transactions on Mathematical Software*, 11(1), 1985.