Graphing of E-Science Data with varying user requirements

Andreas Wombacher, Robin Aly
Database Group,
University of Twente,
Enschede, The Netherlands
Email: (a.wombacher,r.aly@utwente.nl

***Abstract*—Based on our experience in the Swiss Experiment, exploring experimental, scientific data is often done in a visual way. Starting from a global overview the users are zooming in on interesting events. In case of huge data volumes special data structures have to be introduced to provide fast and easy access to the data. Since it is hard to predict on how users will work with the data a generic approach requires self-adaptation of the required special data structures. In this paper we describe the underlying NP-hard problem and present several approaches to address the problem with varying properties. The approaches are illustrated with a small example and are evaluated with a synthetic data set and user queries.**

I. Introduction

In e-science applications a lot of data are collected of which many data are quite boring. To start the scientific work on the data the users have to explore the data and identify the interesting events in the data set. This exploration is often done in a visual way, i.e., the users brows through the graphical visualization (also known as a plot) of the data set. These observations have been made in several projects of the Swiss Experiment [1].

In the Swiss Experiment, the idea is to collect data from various associated projects and provide a web based infrastructure to document, share and visualize data [1], [2]. It also includes an infrastructure component to visualize graphs of data provided by the Global Sensor Network (GSN) infrastructure[2] [3]. After a while the data volume increased significantly and visualizing data as plots resulted in network timeouts since the processing on the server side was too slow.

The main query performed is an aggregation query of calculating the average of the measurement for a certain time period providing a number of data points which are close to the number of pixels presented in the plot. Performing a little benchmark for an aggregation query over several millions of data tuples in a MySQL database indicates that the performance works fine up to a certain level and then the query response time is getting too long for web based applications. In Fig 1 the results of the benchmark are visualized where the x-axis represents the number of tuples being aggregated by the query as a single average value correlated with the query response time on the y-axis. It should be noted that the x-axis values grow exponentially. As a consequence a web based infrastructure for visualizing large amounts of data as a plot requires specialized data structures to fulfill the performance requirements.

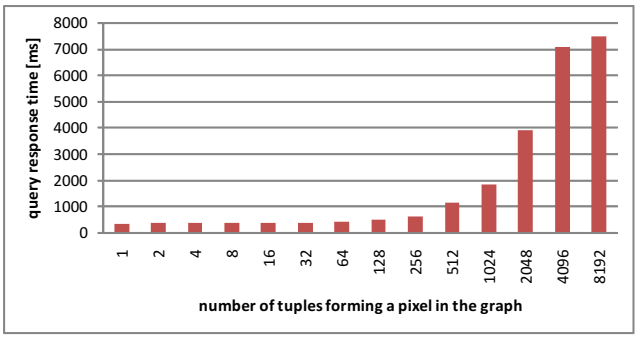

Fig. 1. Benchmark

Based on our experience in the Swiss Experiment, users are only willing to dedicate a certain amount of storage space for these additional data structures in addition to the storage cost required to persist the data itself. Further, it turns out that the characteristics of the data and the user behavior on scanning the data varies in the different associated projects and is user as well as data dependent. As a consequence, the decision which additional data structures have to be initialized depends on the user behavior represented by a set of recently performed queries and the underlying data. Thus, a generic infrastructure must adapt to these requirements.

The underlying idea in this paper is that in addition to the persistent data, pre-aggregations are calculated and stored in the database. In this paper we formalize the underlying problem and discuss several alternative approaches.

II. Problem Definition

A. Assumptions

In the following we represent the dataset D as a multiset of data elements $d \in D$ each associated with a timestamp denoted as $d.t$ and a value denoted as $d.v$. The size of the dataset D is given by the number of elements in the multiset, i.e., $|D|$. The elements in the dataset are ordered by their timestamps. For simplicity, we assume that the elements in the dataset are

[1]http://www.swiss-experiment.ch/
[2]http://sourceforge.net/apps/trac/gsn/

equally distributed over the associated time interval with a constant distance δ between two subsequent elements. Thus, for every two subsequent elements d_k and d_{k+1} the following holds: $d_k.t + \delta = d_{k+1}.t$.

B. Definition of query and plot

Based on the assumptions described in the previous subsection, the terms query and plot can be introduced next. A plot is a graphical representation of data in an image. An image has a maximum pixel resolution on the x-axis, which represents the maximum number of data elements which can be represented. Further, the aim is that the plot contains as many as possible data elements. Thus, we define a plot as follows:

Definition 1 (Plot): A plot is an image representing data elements. A plot has an x-axis resolution with n_{max} pixels. The optimal number of data elements N represented in a plot is $\frac{n_{max}}{2} < N \leq n_{max}$.

A query q_i is specified by a start time s_i and an end time e_i. Intuitively, a query q_i selects a multiset of data elements from a dataset D, such that, $\sigma_{q_i}(D) := \{d \in D | s_i \leq d.t \leq e_i\}$. To represent the query result in a plot the number of elements in the query result must be below the plot resolution n_{max}, which may require to aggregate the data. The aggregation factor f_i is determined as $f_i = \lfloor \frac{|\sigma_{q_i}(D)|}{n_{max}} \rfloor$. The aim is to have the same semantics for each point in the plot, thus, when aggregating we have to make sure that the first data point in the plot represents the first f_i elements of the query result $\sigma_{q_i}(D)$. This can be achieved using an offset of the aggregation, which is determined as $o_i = s_i \bmod f_i$ for a query q_i.

Definition 2 (Aggregation Query): A query q_i is specified by a start time s_i and an end time e_i. To meet the requirement of having a maximum amount of data elements in the result set an aggregation query has an associated factor f_i and offset o_i as given above. The set of all queries is $\mathcal{Q} = \bigcup_i q_i$.

An aggregation query can be translated into the following SQL statement

```
select d.v from D
where s_i ≤ d.t ≤ e_i group by d.t − o_i div f_i;
```

An aggregation query can be either answered directly on the raw dataset D or on a pre-aggregation level A_j with associated aggregation factor f_j^a and offset o_j^a. Let's assume in addition to the dataset D we have an associated pre-aggregation level, that is a pre-calculated dataset A_j based on dataset D where each element in A_j is an average of the f_j^a subsequent elements in dataset D. We define as follows

Definition 3 (pre-aggregation level): A pre-aggregation level is a dataset A_j based on dataset D with a factor f_j^a and an offset o_j^a where for all data elements $d' \in A_j$ holds that

$$d'.v := \frac{1}{f_j^a} \sum_{d \in \{d \in D | \lfloor \frac{d'.t - o_j^a}{f_j^a} \rfloor = \lfloor \frac{d.t - o_j^a}{f_j^a} \rfloor\}} d.v$$

. The set of all pre-aggregation levels is $\mathcal{A} = \bigcup_j A_j$.

The costs of storing a pre-aggregation level $cost(A_j)$ relative to the size of the data set is the number of tuples n the pre-aggregation level. The number of tuples is the reciprocal of the associated factor f_j^a times the size of the dataset, thus the relative costs are $cost(A_j) = 1/f_j^a$.

It should be noted that the query result selected by the original query may deviate from the query result determined on a pre-aggregation level. In particular the following relation holds:

$$|\sigma_{q_i}(D)| - f_i < \lfloor \frac{|\sigma_{q_i}(D)|}{f_i} \rfloor * f_i \leq |\sigma_{q_i}(D)|$$

This deviation is caused by the fact that the number of data elements selected by a queries's start and end time may not be a multiple of the required factor to fulfill the optimality criteria of a plot. However, we accept this marginal deviation.

C. Re-use of Pre-Aggregates

Instead of answering the queries directly on the raw dataset, the system may provide several pre-aggregtion levels to answer user queries fast. Using a pre-aggregation means re-writing the original query by changing the used dataset and adapting the required aggregation factor and offset. A query is answered the fastest if the additionally required aggregation is as small as possible. Thus, for a given query the required factor is f_i and the required offset is o_i. An aggregation level A_j with factor f_j^a and offset o_j^a can be used for answering the query if there exists integer constants c and c' such that

$$\begin{aligned} f_i &= c * f_j^a \\ o_i &= o_j^a - c' * f_j^a \end{aligned}$$

The pre-aggregation A_j with the lowest integer constant c for solving the above equations guarantees the fastest query response time. The cost for answering the query is proportional to the number of tuples which are selected from the dataset to answer the query, i.e.,

$$cost(q_i, A_j) = c * n_i$$

where c is the integer solution of the above equation system, and n_i is the number of data points provided for plotting.

D. Optimization goal

The aim is to provide pre-aggregations for plotting the user queries, while only devoting a certain percentage of the storage space to pre-aggregations. Thus, the question is to find the optimal combination of pre-aggregations for a specific set of user queries and a specific storage constraint. In particular, the optimization goal is to (i) minimize the sum of tuples selected by all queries while (ii) not using more than a percentage ε of the dataset for storing pre-aggregations.

In the following we will use a simplified problem statement since the explanation and visualization of approaches gets too complicated. In particular, we are not using offsets in the further discussion, however, all approaches can be extended by handling offsets.

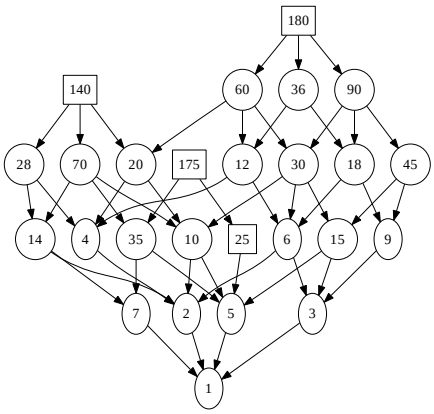

Fig. 2. Complete Graph

III. Example

As a running example to explain and illustrate the different approaches we use the following example. We assume a set of user queries with associated factors 25, 140, 175, 180. Further, we assume that the owner of the application is willing to devote only 5% of the raw dataset for pre-aggreations.

Based on this basic setup 25 pre-aggregation levels are in general possible, which are all integer dividers of at least one of the query factors. All possible pre-aggregation levels are depicted in Fig 2. A circle in the graph represents a possible pre-aggregation level. A square in the graph represents a possible pre-aggregation level, which is also a factor associated to a query. A directed edge indicates that the target node is a integer factor of the source node. The presented graph shows all relations between the different queries and pre-aggregation levels. The aim is to find a set of pre-aggregation levels, which have the lowest query costs while fulfilling the storage space constraint.

To be able to calculate the query and storage costs, we assume that a plot has a maximum of 10 data points. Thus for a query with an associated factor of 175 answered based on the raw dataset has a query cost of 10*175=1750, which is 10 data points in the plot each aggregating 175 data points in the raw data set. Since no pre-aggregation has been instantiated the storage cost is 0.

If a pre-aggreation factor of 25 has been instantiated than the query costs is 10*7=70, since the query with factor 175 can be answered by aggregating 7 data points of the pre-aggregation level of 25. The storage cost is 4%, which is the reciprocal of 25, i.e., $1/25 = 0.04 = 4\%$.

IV. Approaches

In the following several possible approaches are investigated and their advantages and disadvantages are discussed. The results of are summarized in Tab I.

A. Optimal Solution

The optimal solution can be determined by an exhaustive search, i.e., for each element of the power set of pre-aggregation levels it is determined whether the storage constraint is fulfilled and if so the query costs are determined. Finally the set of pre-aggregation levels is determined having the lowest query costs. Performing the exhaustive search determines pre-aggregation levels of 140, 175, and 180 as the optimal solution with a query cost of 280 and storage costs of 1.84%. The resulting solution is depicted in Fig 3, where circles represent instantiated pre-aggregation levels. A white square represents a factor which is an instantiated pre-aggregation level AND is associated with a query. A grey square represents a factor associated with a query, but is not instantiated as a pre-aggregation level. A directed edge indicates that the target node is a integer factor of the source node.

The advantage of this solution is that it provides the optimal solution. The disadvantage is that it does not scale since the number of possible combinations of pre-aggregation levels is exponential with the set of factors derived from the factors associated with the queries.

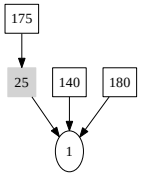

Fig. 3. Optimal Solution

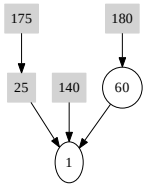

Fig. 4. Fixed Factors Solution

B. Fixed Factors Approach

The approach is based on the observation that people are often in specific queries, like e.g. the data of a day or a week or a year. In these cases a fixed set of pre-aggregates can be specified and the algorithm in this approach makes a selection of this fixed set based on the available queries. In particular, the algorithm iterates over all possible pre-aggregates (line 3 of Alg 1). It checks in line 4 whether there is still sufficient storage space available before iterating over all queries to investigate whether adding this pre-aggregate is beneficial to any query (line 7). If this is the case the pre-aggregate is added to the result (line 11) and the used storage is updated (line 12). The determined set of pre-aggregates is returned in line 13. As fixed pre-aggregates we suggest to use the following time aggregates: 10 min, 15 min, hour, 4 hours, day, week, and month.

Applying this approach to the example results in a single pre-aggregate of factor 60 which corresponds in our case to the hourly average (see Fig 4). The resulting query cost is 3430 and the used storage is 1.67%. The query cost is quite far away from the optimal query cost. The reason for that is that the used queries do not follow the assumptions made in the approach. They are not queries going along the lines of usual time related factors. The benefit of the approach is that it can be easily calculated and the complexity of the algorithm is independent of the factors used in the queries, but only dependent on the number of queries to be considered.

Approach	Optimal	Fixed Factors	Min Multiple	GCD	GCD extended	LIP
example query costs	280	3430	350	1890	1490	280
example storage costs	1.84%	1.67%	3.41%	4.71%	4.56%	1.84%
pro	optimal solution	independent of query factors	storage boundary by construction	independent of query factors	independent of query factors	optimal solution
con	does not scale	assumptions on queries	empirical	depends on order of factor selection; empirical	empirical	scaling unclear

TABLE I
SUMMARY OF EXAMPLE SOLUTIONS

input : none
output: set of pre-aggregation levels
1 $storage = 0$;
2 $factors = \emptyset$;
3 **forall the** $A_j \in \mathcal{A}$ **do**
4 **if** $\frac{1}{f_j^a} + storage \leq \varepsilon$ **then**
5 $flag = false$;
6 **forall the** $q_i \in \mathcal{Q}$ **do**
7 **if** $f_i \bmod f_j^a = 0$ **then**
8 $flag = true$;
9 break;
10 **if** *flag* **then**
11 $factors.add(f_j^a)$;
12 $storage = storage + \frac{1}{f_j^a}$;
13 **return** factors;

Algorithm 1: Fixed Factors Algorithm

C. Minimum Multiples Approach

Continuing the basic idea of avoiding the combinatorial complexity of pre-aggregates, we propose an approach which ensures the storage constraint by limiting the choices of possible pre-aggregation levels. In particular, for a storage constraint of ε of the dataset, an upper bound of the storage constraint can be guaranteed if the series $\sum_{i=1}^{n} \frac{1}{a} * (\frac{1}{b})^i \leq \varepsilon$ fulfills the inequality. The closed formulation of the series is $\sum_{i=1}^{n} \frac{1}{a} * (\frac{1}{b})^i = \frac{1}{a} * \frac{b^n - 1}{b^n - b}$.

In this approach we use the above stated inequality to first calculate a minimum value for the a variable by simplifying the inequality, i.e., $\varepsilon \geq \frac{1}{a} * \frac{b^n-1}{b^n-b} \geq \frac{1}{a}$, thus $a \geq \frac{1}{\varepsilon}$ (see line 4 in Alg 2). Using this value as a start, we determine in lines 7-17 the factor producing the minimum query costs. In case we do not find a new factor, the current factors are returned (line 19). Otherwise, the process is repeated while the depth counter is increased, the currently known minimal cost and the factors are used further on.

In the subsequent iterations, the a variable in the equation is known (available as the second element in the factors list in Alg 2). To determine the start for searching the following pre-aggregation level the b variable has to be determined. The b variable can be calculated by re-writing the inequality as $b \geq \frac{a\varepsilon}{a\varepsilon - 1 + \frac{1}{b^n}}$. Since for the right hand side of the inequality it applies that $\frac{a\varepsilon}{a\varepsilon-1} \frac{a\varepsilon}{a\varepsilon-1+\frac{1}{b^n}}$, we are on the safe side if value b fulfills the following condition: $b \geq \frac{a\varepsilon}{a\varepsilon-1}$. We use this inequality in line 6 of Alg 2. In fact, the actual used values are usually higher than the calculated b values, which offers some potential for optimization of the algorithm.

input : depth, lowestCost, factors
output: set of pre-aggregation levels
1 PrevCost = Integer.MAX_VALUE;
2 PrevFactors = NULL;
3 **if** *depth==0* **then**
4 $start = \frac{1}{\varepsilon}$;
5 **else**
6 $start = factors(2) * \left(\left\lceil \frac{factors(2)*\varepsilon}{factors(2)*\varepsilon - 1} \right\rceil \right)^{depth}$;
7 **for** $j = start$ **to** max_factor **do**
8 c=0;
9 factors.add(j);
10 **forall the** $q_i \in \mathcal{Q}$ **do**
11 **for** $k = factors.size()$ **to** 1 **do**
12 **if** $f_j \bmod factors(k) = 0$ **then**
13 $c = c + f_i / factor(k)$;
14 break;
15 **if** $c < PrevCost$ **then**
16 PrevFactors = factors.clone();
17 PrevCost = c ;
18 **if** *PrevFactors = NULL* **then**
19 **return** factors;
20 **else**
21 **return** MinimumMultiples(depth+1, PrevCost, factors);

Algorithm 2: Minimum Multiples Algorithm

Applying this algorithm to the example results in pre-aggregation levels 35 and 180 and a query cost of 350 with a storage cost of 3.41%. The main advantage of this approach is that the storage constrained is guaranteed by the limitation of considered pre-aggregation level combinations. However, this is also the main disadvantage, since for each depth there can

only be a single pre-aggregate. With regard to the example, the a variable is determined as the first selected pre-aggregate, which is $a = 35$. As a consequence, the determined b value is $b = 3$. Thus, the next pre-aggregate has to be at least of factor $35 * 3 = 105$. The algorithm selects 180. In the subsequent iteration, the next pre-aggregate must have at least a value of $35 * 3^2 = 315$ which is higher than the maximum query factor of 180. Thus, it is not possible to add the pre-aggregate 140 to the solution although it would be beneficial and sufficient storage space is available, since for every depth there can only be a single pre-aggregate. The complexity of the approach is linear with number of queries to be considered and logarithmic with the maximum factor of the queries.

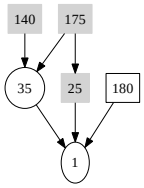

Fig. 5. Minimum Multiple Solution

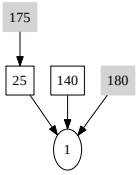

Fig. 6. Greatest Common Divisor Solution

D. Greatest Common Divisor Approach

Although in the following approach we are avoiding to explore all pre-aggregate combinations, but try to get some more flexibility in the choice of the possible pre-aggregates. The basic idea behind this approach is that the factors shared between queries the most are most beneficial for being instantiated as a pre-aggregate. In particular, we are considering the Greatest Common Dividers (GCD) between two queries since the higher the common factor the more beneficial for the query costs. Further, we expect that there is a minimum integer factor between two pre-aggregation levels to ensure that there is a significant benefit of introducing the higher pre-aggregate..

The approach is performed in the following steps described in Alg 3

1) calculate the histogram of all Greatest Common Divisors (GCD) of pairs of queries (lines 3-6)
2) select the bin, i.e., the factor, in the histogram with the highest count (line 11-14)
3) if the factor is not violating the storage constraint (line 16), add the selected factor to the list of pre-aggregation levels (line 18) and adjust the remaining storage capacity (line 17)
4) remove all bins A from the histogram which are in relation to the previously selected bin B such that for a minimum factor f between two bins the following holds: $(A \bmod B = 0 \wedge A/B < f) \vee B \bmod A = 0 \wedge B/A < f)$ (lines 19-21)
5) repeat from 2 until no new bins can be added

The presented approach does not specify how bins with equal count in the histogram should be dealt with. As a consequence, the result of the approach depends on the order in which factors with the same count are considered. This is a clear disadvantage since different implementations of the same approach applied to the same set of queries may produce different results. A further disadvantage is that the result is empirical and we can not provide any assessment of the possible deviation of the query costs from the optimal one.

```
   input : none
   output: set of pre-aggregation levels
1  factors = ∅;
2  map = ∅ ;          /* Relation gcd → count */
3  forall the q_i ∈ Q do
4      forall the q_j ∈ Q do
5          gcd = GCD(f_i, f_j);
6          map(gcd) = map(gcd) + 1;
7  stor = 0;
8  max_count = 0;
9  max_gcd = 0;
10 while map ≠ ∅ do
11     forall the (gcd → count) ∈ map do
12         if count > max_count then
13             max_count = count;
14             max_gcd = gcd;
15     map.remove(max_gcd → max_count);
16     if 1/max_gcd + stor < ε then
17         stor = stor + 1/max_gcd;
18         factors.add(max_gcd);
19     forall the (gcd → count) ∈ map do
20         if (max_gcd mod gcd = 0 ∧ max_gcd/gcd <
           f) ∨ (gcd mod max_gcd =
           0 ∧ gcd/max_gcd < f) then
21             map.remove(gcd → count);
22 return factors;
```

Algorithm 3: GCD Algorithm

For the running example the result is depicted in Fig 6. The instantiated pre-aggregations are 25 and 140 resulting in a query cost of 1890 and a storage cost of 4.71%. The solution is reasonable compared to the optimal and the minimum multiple approach. The disadvantage of the approach is that in case of the histogram having several factors with the same count the selection of a factor depends on the implementation and the order of inserting the data in the histogram. Thus, the algorithm is non-deterministic. Further, this is an empirical approach and it is not possible to provide any generic estimate on the quality of the derived solution. The advantage is that it is independent of the factors used in the query and only dependent on the number of considered queries. Thus, the approach scales very well with the factors being used. Thus, the complexity of the approach depends quadratic on the number of considered queries.

E. Extended Greatest Common Divisor Approach

To address the non-determinism disadvantage of the Greatest Common Divisor Approach a selection criteria is introduced in case there are several bins in the histogram with the same count. In case of equal counts, the approach checks for the set of bins with equal counts whether already accepted pre-aggregations levels reduce the benefit of the particular bin and select the bin which has the highest benefit. The benefit is defined as the highest new aggregation achieved by introducing this pre-aggregate. The benefit of a bin is calculated by determining the highest integer quotient of an accepted pre-aggregate. In case there is no integer quotient, then the bin value is used as the benefit measure. Also in this case there might be non-determinism, however, this is much less likely and much less influential on the result.

Applying this approach to the running example results in pre-aggregation levels of 25 and 180 with a query cost of 1490 and a storage cost of 4.56% (see Fig 7). The difference between the approach described above and the GCD approach is that the pre-aggregation level of 140 is replaced by the pre-aggregation level of 180. This reduces the query cost marginally. The advantages and disadvantages are the same as discussed before except that the non-determinism has been significantly reduced.

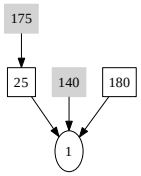

Fig. 7. Extended Greatest Common Divisor Solution

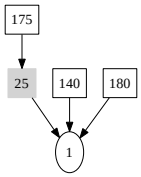

Fig. 8. Linear Integer Programming Solution

F. Linear Integer Programming Approach

The final approach proposed in this paper is to translate the problem into a Linear Integer Programming (LIP) problem. Based on such a problem description we can facilitate existing algorithms and tools to solve our problem. Further, in case the LIP solver finds a solution, we know we have determined the optimal solution. The challenge is to formulate the problem in the right way.

Our problem can be described similar to the Lockbox problem as introduced in [4], which is NP-hard. For the set of queries $\mathcal{Q} = \bigcup_{i=1}^{n} q_i$ and the set of possible pre-aggregation levels $\mathcal{A} = \bigcup_{j=1}^{m} A_j$ we define the following variables:

- $x_{i,j}$ is set to one if query q_i is answered by pre-aggregation level A_j, otherwise is set to zero
- y_j is set to one if pre-aggregation level A_j is part of the solution, otherwise is set to zero
- $L_{i,j}$ is the costs for executing query q_i with pre-aggregation level A_j; the cost is either f_i/f_j^a if $f_i \mod f_j^a = 0$, or a high constant otherwise indicating that the query factor f_i is not an integer multiple of the pre-aggregate f_j^a
- $I = \{1, \ldots, n\}$ is the set of indexes of all queries $q_i \in \mathcal{Q}$
- $J = \{1, \ldots, m\}$ is the set of indexes of all pre-aggregation levels $A_j \in \mathcal{A}$

As can be seen from above definitions, there is one variable y_j indicating whether a possible pre-aggregate belongs to the solution. Further there is one variable $x_{i,j}$ indicating whether the query q_i will be answered by pre-aggregate A_j in the optimal solution. The linear integer program (ILP) can be described as:

$$\begin{aligned} \text{minimize} \quad & \sum_{i \in I} \sum_{j \in J} L_{i,j} x_{i,j} \\ \text{subject to} \quad & \sum_{j \in J} x_{i,j} = 1 && \text{for all } i \in I \\ & \sum_{i \in I} x_{i,j} \leq |\mathcal{Q}| y_j && \text{for all } j \in J \\ & \sum_{j \in J} \frac{1}{f_j^a} y_j < 1 + \varepsilon && (1) \end{aligned}$$

The first line specifies the optimization function, which is the minimization of the query execution costs of all queries in $\mathcal{Q}$. The second line indicates that each query is answered by exactly one pre-aggregation level. The third line ensures that if a query uses a pre-aggregation level (indicated by $x_{i,j}$) then this pre-aggregation level must also be maintained (indicated by y_j). The last line represent the constraint on the storage cost, i.e., the sum of all storage costs must be below the threshold of $1 + \varepsilon$. The constraint threshold is bigger than one since the solution of the LIP must contain the raw data to ensure that for each query a pre-aggregate can be determined. Since the raw data have a storage cost of 100%, thus 1, this must be added to the storage constraint.

To solve the LIP we use the open source LPsolve solver [3]. The solution of the LIP approach (see Fig 8) consists of the pre-aggregates 140, 175, 180 with a query cost of 280 and a storage consumption of 1.84 %. It is the same as the optimal solution. The advantage of the approach is that if a soolution can be determined it is optimal. The disadvantage of the solution is that not always a solution can be derived. In particular, with increasing numbers of variables the risk of not finding a solution increases.

The complexity of the approach consist of creating the LIP problem description and solving the LIP problem. Since we use an off the self solver we can not make a statement about the complexity, since the solver provides various algorithms.

V. Evaluation

The different approaches have been introduced with rather small queries. To get a better feeling for the quality of the results of the various approaches, we specified a small set of queries on a dataset of 3 years of environmental data. The queries we chose have the associated factors 241884, 3600, 86400, and 172800. We keep the storage constraint as before with 5% of the dataset. The corresponding graph of possible

[3] http://sourceforge.net/projects/lpsolve/

pre-aggregates consists of 117 nodes indicating a search space of 2^{117} for finding the optimal solution in an exhaustive search. The calculation of the optimal result by an exhaustive search (see Sect IV-A) did not terminate and therefore we can not provide any results. All results are summarized in Tab II providing the query costs, the storage costs, the execution time for finding the solution as well as the set of pre-aggregates found by the approach.

The Fixed Factors approach (see Sect IV-B) produces very fast really poor results. The resulting query costs is not acceptable. Thus, we have the impression that the assumed constraints on the factors used in the queries are too limiting for the kind of queries used in this evaluation.

The Minimum Multiple approach (see Sect IV-C) is the slowest of all approaches but produces very low query costs. This approach produces the highest number of pre-aggregates. This is due to the design of the algorithm. For the evaluation queries, the a variable is set to $a = 36$ and the b variable can be determined as $b = 3$. As a consequence, the maximum number of pre-aggregates generated by this approach can be calculated knowing the maximum query factor, which is 241884 as $\left\lfloor \frac{\ln(241884)-\ln(a)}{\ln(b)} \right\rfloor = 8$. For the evaluation query the algorithm makes use of the all eight pre-aggregatres. The high processing time, can be reduced by slightly modifying the algorithm in Alg 2 by constraining the upper bound of the for loop in line 7. However, since performance is not the main evaluation criteria we kept the original version of the algorithm.

The Greatest Common Divisor approach (see Sect IV-D) and the Extended Greatest Common Divisor approach (see Sect IV-E) are fast wrt processing time and produce almost an optimal result.

The Linear Integer Programming approach (see Sect IV-F) results in a problem description with $4 * 117 = 585$ variables and 122 constraints. Surprisingly the LIP solver produces fast a result with minimum query costs.

We are aware that we have to perform a more elaborate evaluation and investigate in more detail the conditions under which the different approaches perform well or not. However, based on this initial evaluation, we can conclude that the Fixed Factors seems not be applicable in a generic case. Further, we see that Minimum Multiples, the Greatest Common Divisor, and the LIP approaches are promising. In particular, we find the LIP results very promising and will investigate in future work specific conditions under which the approach will fail. However, these four approaches have a boot strapping problem. That is, if a dataset is made available and no queries have been performed by the user on the dataset, we have to guess the most likely user queries to boot strap/ initialize the system to support that user queries can be answered with low query costs.

VI. Alternative Approaches

In addition to the approaches discussed in Sect IV, we explored alternative approaches.

A. Map Reduce

Since we are dealing with large datasets, the idea is not to provide pre-aggregates but to use the distributed computing power of a cluster to answer the queries directly from the raw data. A computing principle supporting the automated distribution of tasks and aggregation of partial results in Map Reduce. Therefore, we stored the data in a HBase on a cluster and performed the queries using a Map Reduce infrastructure. This resulted in very slow query executions with query processing times in the order of magnitude of minutes. Since this is too far away from the targeted scenario, we abandoned this approach.

B. Aggressive Caching

We also had the idea of using a caching infrastructure instead of pre-aggregates. This makes a lot of sense if we are in an environment with standing queries, i.e., the same queries executed periodically. However, if you are dealing with a dataset of sensor data which is still evolving it turned out that the maintenance of the cached fragments and their relations is quite complicated. Especially if a new query must be answered by re-using several cached results. We think that this overhead out weights the potential benefits and therefore abandoned also this approach.

C. Haar Wavelets

Haar wavelets have been proposed e.g. in [5] as a method for storing data and implicitly providing pre-aggregates. In case of Haar wavelets the raw data are no longer explicitly stored, but the transformed data. The transformation is invertible and therefore all raw data can still be derived. The transformation uses a sequence of 2^n elements of the raw dataset and provides n pre-aggregates with factors $2, \ldots, 2^n$. In case a query requires a factor not being directly one of these factors the required factor can be constructed as a linear combination of the available factors. Like for example a factor of 25, which has the binary representation of 11001_2, can be constructed by $\frac{2^4 * f_4^a + 2^3 * f_3^a + 2^0 * f_0^a}{25}$. However, to construct the raw data represented in the formula as f_0^a, all Haar wavelet data of the complete time span is required. As a consequence, there is no benefit for query costs if the factor is not a power of two. Therefore, we also do not further consider this case.

VII. Related Work

Besides the related work discussed in the previous section, there is quite some related work on approximate query evaluation like e.g. applied in stream query processing. The idea here is that with limited processing effort an approximate aggregation query result can be achieved. These approaches are designed for stream processing and are not really helpful for offline processing as discussed in this paper.

There are various open source systems for managing data of e-science applications like e.g. [3]. Most of those provide a graphing functionality, which is often implemented in a straight forward way. A very interesting approach has been described in [6]. The authors describe a caching based approach,

Approach	Optimal	Fixed Factors	Min Multiple	GCD	GCD extended	LIP
query costs	-	2420790	40	50	50	40
storage costs	-	0.48%	2.84%	2.81%	2.81%	0.03%
proces. time [ms]	-	5	3762	11	11	52
pre-aggreates	-	240, 1440	36, 3600, 5103, 15309, 45927, 86400, 137781, 172800, 241884	36, 3600, 86400, 241884	36,3600, 86400, 241884	3600, 86400, 172800, 241884

TABLE II
SUMMARY OF EVALUATION SOLUTIONS

however the user is limited to a fixed set of pre-aggregates. Thus, the proposed solution is ore limited as the ones discussed in this paper.

In other domains like e.g. monitoring infrastructure similar problems of graphing the observed data are known. A often used tool in this context is based on a round robin database [7]. This means that the users of the system indicate which pre-aggregates they want to have and for how long they would like to have access to these data. In case data expires, it is overwritten and not accessible anymore. The zooming functions available here are limited to the level of stored pre-aggregate.

Recently Google fusion tables [8][9] getting a lot of attention. There are basic visualization capabilities provided by the service. However, so far it seems the emphasis is more on the sharing and integration of the data rather than the visualization.

Please note that the topic addressed in this paper is very different from data aggregation in sensor networks as addressed e.g. in [10], since these approaches do not preserve the original data.

VIII. CONCLUSION

To present in a web application data in a graph requires fast aggregation of the data. In this paper, we formalize the problem and investigate several alternatives. The most promising one so far seems to be the Linear Integer Programing approach since it produces an optimal solution although we expect that in large scenarios the underlying algorithm will not provide a solution. Therefore the Greatest Common Divisor and the Minimum Multiples approaches are also promising. However, these approaches are empirical and no statement about the quality of the result can be made.

In future work we will extend the presented approaches by considering offsets as described in the problem definition. Although this does not require a structural change of the approaches, it increases the search space significantly.

IX. ACKNOWLEDGMENT

This work and the different approaches have been inspired by the following students of the Univerisity of Twente: Vincent van Donselaar, Yassin El Moutaouakkil, Guillaume Sellier, Henry Been, Edwin Keijl, Edwin Smulders, Renske Vermolen, van Gennep, Hijmans, van der Molen, van Toll, Tjeerd Boerman, Joost Wolfswinkel, Maarten Fonville

REFERENCES

[1] H. Jeung, S. Sarni, I. K. Paparrizos, S. Sathe, K. Aberer, N. Dawes, T. G. Papaioannou, and M. Lehning, "Effective metadata management in federated sensor networks," in *SUTC/UMC*. IEEE Computer Society, 2010, pp. 107–114. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/SUTC.2010.29

[2] A. V. Singh, A. Wombacher, and K. Aberer, "Personalized information access in a wiki using structured tagging," in *OTM Workshops (1)*, ser. Lecture Notes in Computer Science, R. Meersman, Z. Tari, and P. Herrero, Eds., vol. 4805. Springer, 2007, pp. 427–436. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-76888-3_65

[3] K. Aberer, M. Hauswirth, and A. Salehi, "Infrastructure for data processing in large-scale interconnected sensor networks," *Mobile Data Management, 2007 International Conference on*, pp. 198–205, May 2007.

[4] A. Holder, Ed., *Mathematical Programming Glossary*. http://glossary.computing.society.informs.org: INFORMS Computing Society, 2006–08, originally authored by Harvey J. Greenberg, 1999-2006.

[5] C. Shahabi, M. Jahangiri, and F. B. Kashani, "ProDA: An end-to-end wavelet-based OLAP system for massive datasets," *IEEE Computer*, vol. 41, no. 4, pp. 69–77, 2008. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/MC.2008.130

[6] M. Keller and J. Beutel, "Demo abstract: Efficient data retrieval for interactive browsing of large sensor network data sets," in *Proceedings of the 10th International Conference on Information Processing in Sensor Networks*, 2011, pp. 139–140.

[7] T. Oetiker, "MRTG: The multi router traffic grapher," in *Proceedings of the 12th Conference on Systems Administration (LISA-98)*. Berkeley, CA: USENIX Association, Dec. 6–11 1998, pp. 141–148.

[8] H. Gonzalez, A. Y. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley, W. Shen, and J. Goldberg-Kidon, "Google fusion tables: web-centered data management and collaboration," in *Proceedings of the 2010 international conference on Management of data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 1061–1066.

[9] H. Gonzalez, A. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley, and W. Shen, "Google fusion tables: data management, integration and collaboration in the cloud," in *Proceedings of the 1st ACM symposium on Cloud computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 175–180.

[10] R. Rajagopalan and P. K. Varshney, "Data-aggregation techniques in sensor networks: A survey," *IEEE Communications Surveys and Tutorials*, vol. 8, no. 1-4, pp. 48–63, 2006.