Aggregating and Disaggregating Flexibility Objects

Siksnys, Laurynas; Khalefa, Mohamed; Pedersen, Torben Bach

Published in:
Scientific and Statistical Database Management

DOI (link to publication from Publisher):
10.1007/978-3-642-31235-9_25

Publication date:
2012

Document Version
Early version, also known as pre-print

Link to publication from Aalborg University

Citation for published version (APA):
Siksnys, L., Khalefa, M., & Pedersen, T. B. (2012). Aggregating and Disaggregating Flexibility Objects. In A. Ailamaki, & S. Bowers (Eds.), *Scientific and Statistical Database Management: 24th International Conference, SSDBM 2012, Chania, Crete, Greece, June 25-27, 2012. Proceedings* (pp. 379-396). Springer. https://doi.org/10.1007/978-3-642-31235-9_25

Aggregating and Disaggregating Flexibility Objects

Laurynas Šikšnys, Mohamed E. Khalefa, and Torben Bach Pedersen

Department of Computer Science, Aalborg University
{siksnys,mohamed,tbp}@cs.aau.dk

Abstract. Flexibility objects, objects with flexibilities in time and amount dimensions (e.g., energy or product amount), occur in many scientific and commercial domains. Managing such objects with existing DBMSs is infeasible due to the complexity, data volume, and complex functionality needed, so a new kind of *flexibility database* is needed. This paper is the first to consider flexibility databases. It formally defines the concept of flexibility objects (flex-objects), and provide a novel and efficient solution for aggregating and disaggregating flex-objects. This is important for a range of applications, including smart grid energy management. The paper considers the grouping of flex-objects, alternatives for computing aggregates, the disaggregation process, their associated requirements, as well as efficient incremental computation. Extensive experiments based on data from a real-world energy domain project show that the proposed solution provides good performance while still satisfying the strict requirements.

1 Introduction

Objects with inherent flexibilities in both the time dimension and one or more amount dimensions exist in both scientific and commercial domains, e.g., energy research or trading. Such objects are termed *flexibility objects* (in short, flex-objects). Capturing flexibilities explicitly is important for *smart grid* domain, in order to consume energy more flexibly. In the ongoing EU FP7 research project MIRABEL project [2], the aim is to increase the share of renewable energy sources (RES) such as wind and solar, by capturing energy demand and supply, and the associated flexibilities. Here, flex-objects facilitate planning and billing of energy. First, consumers (automatically) specify the flexible part of their energy consumption, e.g., charging an electric vehicle, by issuing flex-objects to their energy company. A flex-object defines how much energy is needed and when, and the tolerated flexibilities in time (e.g., between 9PM and 5AM) and energy amount (e.g. between 2 and 4 kWh). In order to reduce the planning complexity, similar flex-objects are aggregated into larger "macro" flex-objects. Then, the energy company tries to plan energy consumption (and production) such that the desired consumption, given the macro flex-objects plus a non-flexible base load, matches the forecasted production from RES (and other sources) as well as possible. During the process, the flex-objects are *instantiated* (their flexibilities are *fixed*), resulting in so-called *fix-objects*, where fixed concrete values are assigned for time and amounts, within the original flexibility intervals. Finally, the "macro" fix-objects are *disaggregated* to yield "micro" fix-objects corresponding to the instantiation of the original flex-objects issued by the consumers, which are then distributed back to the consumers. The consumers are rewarded according to their specified flexibility.

For typical real world and scientific applications (e.g., MIRABEL), there exists, hundreds of millions of flex-objects have to be managed efficiently. This is infeasible using existing DBMSs. First, due to the complexity of flex-objects, queries over flex-objects are also complex and cannot be formulated efficiently, if at all, using standard query languages. Second, the number of flex-objects can be very large and timing restrictions are tight. Third, incremental processing must be supported natively. In order to efficiently evaluates queries over flex-objects, a new tailor-made database for handling flex-objects is needed.

Contributions This paper is the first to introduce the vision of flex-object databases by discussing their functionality, queries, and application. As the most important operations of a flex-object database, the paper focuses on flex-object aggregation and disaggregation, which are analogous to a *roll-up* and *drill-down* queries in an OLAP database. As will be shown later, the aggregation of even just two flex-objects is non-trivial as they can be combined in many possible ways. The paper formally defines flex-objects, how to measure the flexibility of a flex-object, aggregation and disaggregation of flex-objects. Our flex-object aggregation approach takes a set of flex-objects as input and, based on the provided aggregated parameters, partitions the set of flex-objects into disjoint groups of similar flex-objects. This partitioning is performed in two steps - grid-based grouping and bin-packing. The grouping of flex-objects ensures that flex-objects in a group are sufficiently similar (in terms of chosen flexibility attributes). The bin-packing ensures that the groups themselves conform to the given (aggregate) criteria. After bin-packing, an aggregation operator is applied to merge similar flex-objects into aggregated flex-objects. In one possible scenario, when the flexibilities of the aggregated flex-objects are fixed, e.g., in the planning phase, our disaggregation approach takes the respective fix-objects as input and disaggregate them into fix-objects that correspond to the original flex-objects. Our solution inherently supports efficient incremental aggregation, which is essential to handle continuously arriving new flex-objects. An extensive set of experiments on MIRABEL data shows that our solution scales well, handling both aggregation and disaggregation efficiently.

The remainder of the paper is structured as follows. Section 2 outlines the functionality of envisioned flex-object databases. Section 3 formally defines the concept of flex-object and the problem of aggregating and disaggregating flex-objects. Section 4 describes how to aggregate several flex-objects into one. Section 5 generalizes the approach to aggregate a set of input flex-objects into a set of output flex-objects. Section 6 considers incremental aggregation. Section 7 describes the experimental evaluation, while Section 8 discusses related work. Finally, Section 9 concludes and points to future work.

2 Flex-object Databases

Although the paper focuses on two specific operations on flex-objects, aggregation and disaggregation, in this section, we provide a broader context by outlining our vision for the functionality of *flex-object databases*. A flex-object database is a database storing flex-objects, possibly of several different types and with different types of flexibilities. Thus, flex-objects must be first-class citizens in such a database and the associated complex functionality must be supported. In some application scenarios, the flex-object database will be stand-alone and focused only on flexibility management, in other scenarios, the flex-object database will be part of a larger database storing also other kinds of objects and with a mixed query workload. Support for dimensions and hierarchies is important to be able to view flex-objects at the desired level of granularity, however, the dimension hierarchies must be more complex than in current systems to support complex real-world hierarchies such as energy distribution grids. Similarly, spatio-temporal support is essential, as many flex-object applications have strong spatial and/or temporal aspects. Several options for query languages are possible. For standalone flex-object-specific applications, a JSON-style declarative language like that used by MongoDB would be effective. For more general mixed databases, an extension of SQL with specific syntax and operators for flexibility manipulation is desirable.

The storage of flex-objects is not a trivial issue. A flex-object database should be able to store massive amounts of flex-objects while still ensuring very fast response time. Due to their complex internal structures (to be detailed in the next section), flex-offers cannot be directly stored as atomic objects in standard relational databases. Here, accessing flex-objects becomes an expensive operation requiring joins and aggregations over two or more large tables. Alternatives include nested object-relational storage and or dedicated native storage, where it is important to strike the right balance between efficiency and the ability to integrate easily with other types of data. The storage issue is beyond the scope of this paper, and will be addressed in future work. We now present at the most important types of queries to be supported by a flex-object database.

Flexibility availability queries provide an overview over the *amount flexibilities* that are available at given time intervals. For example, such a query may retrieve the *min, max, and average* amounts available, or build a time series with the (expected) distribution of amounts at every time instance. Such queries are used in feasibility/risk analysis where nominal or peak values are explored, e.g., to see how much energy consumption can be increased or decreased at a given time to counter unexpected highs or lows in the RES production.

Adjustment potential queries computes the distribution of amounts that can be potentially injected into (or extracted away from) a given time interval, taking into account the amounts which are already fixed with fix-objects. Several options for the amount injection (or extraction) are possible, including adjusting amounts within amount flexibility ranges, shifting amounts within available time flexibility ranges, or a combination.

Fixing queries alter (or create) fix-objects (the plan), based on user selected amounts to inject or extract. Fixing queries are used in the analysis and planning phase, to interactively explore flexibility potentials (the first two types of queries), followed by modifying the existing plan (fix-objects) if needed.

Flex-object aggregation queries combine a set of flex-objects into fewer, "larger" flex-objects. In some sense, this is analogous to a *roll-up* query in an OLAP database (going from finer to coarser granularities), although considerably more complex (as will be discussed in the next sections). The aggregation usually reduces flexibilities, so it is important to quantify and minimize how much of the original flexibilities are lost due to aggregation. The aggregation of flex-objects can substantially reduce the complexity of the above-mentioned analysis queries as well as the complexity of various flex-object-based planning processes. For example, a very large number of flex-objects must be scheduled in MIRABEL. Since scheduling is NP-complete problem, it is infeasible to schedule all these flex-objects individually within the (short) available time. Instead, flex-objects can be *aggregated*, then *scheduled* (not considered in this paper), and finally *disaggregated* (see below) into fix-objects.

Flex-object disaggregation queries go the opposite way, "exploding" a large "macro" fix-object into a set of smaller "micro" fix-objects corresponding to the instantiation of the original flex-objects. This yields the refinement of the "macro" plan necessary for carrying out the plan in practice. In some sense, this is like a *drill-down* query in an OLAP database (going from coarser to finer granularities), but more complex.

Flex-object aggregation and disaggregation queries are particularly important and more challenging. In the next section, we formulate the problem of aggregating and disaggregating flex-objects.

3 Problem Formulation

We now formalize our proposed problem of aggregating and disaggregating flexibility objects. Our formalization includes (1) a definition of flex-object, (2) a measure to quantify flex-object flexibility, and (3) *aggregation* and *disaggregation* functions and their associated constraints.

The introduced flex-object is a multidimensional object capturing two aspects: (1) the *time flexibility interval*, and (2) a *data profile* with a sequence of consecutive *slices* each defined by (a) its start and end time and (b) the minimum and maximum amount bounds for one or more amount dimensions. We can formally define a flex-object as follows:

Definition 1: *A flex-object f is a tuple $f = (T(f), profile(f))$ where $T(f)$ is the start time flexibility interval and $profile(f)$ is the data profile. Here, $T(f) = [t_{es}, t_{ls}]$ where t_{es} and t_{ls} are the* earliest start time *and* latest start time*, respectively. The $profile(f) = s^{(1)}, \ldots, s^{(m)}$ where a* slice *$s^{(i)}$ is a tuple $([t_s, t_e]$, $[a^{(1)}_{min}, a^{(1)}_{max}], \ldots, [a^{(D)}_{min}, a^{(D)}_{max}])$ where $[a^{(i)}_{min}, a^{(i)}_{max}]$ is a continuous range of the amount for dimension $i = 1..D$ and $[t_s, t_e]$ is a time interval defining the* *extent of* $s^{(i)}$ *in the time dimension. Here,* $t_{es} \leq s^{(1)}.t_s \leq t_{ls}$ *and* $\forall j = 1..m$: $s^{(j)}.t_e > s^{(j)}.t_s$, $s^{(j+1)}.t_s = s^{(j)}.t_e$. *We use the terms* profile start time *to denote* $s^{(1)}.t_s$, duration of the slice *to denote* $s_{dur}(s) = s.t_e - s.t_s$, duration of profile *to denote* $p_{dur}(f) = \sum_{s \in profile(f)} s.t_e - s.t_s$, *and* latest end time *to denote* $t_{le}(f) = f.t_{ls} + p_{dur}(f)$.

For simplicity and without loss of generality, time is discretized into equal-sized units, e.g., 15 minute intervals, and we have only one amount dimension (i.e., $D = 1$). Figure 1 depicts the example of a flex-object having a profile with four slices: $s^{(1)}$, $s^{(2)}$, $s^{(3)}$, and $s^{(4)}$. Every slice is represented by a bar in the figure. The area of the light-shaded bar represents the minimum amount value (a_{min}) and the combined area of the light- and dark-shaded bars represents the maximum amount value (a_{max}). The left and the right sides of a bar represent t_e and t_s of a slice, respectively.

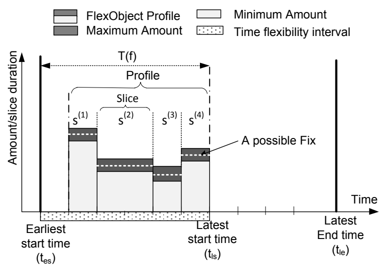


Fig. 1. A generic flex-object

We distinguish two types of flexibility associated with f. The *time flexibility*, $tf(f)$, is the difference between the latest and earliest start time. Similarly, the *amount flexibility*, $af(s)$, is the sum of amount flexibility of all slices in the profile of f, i.e., $af(f) = \sum_{s \in profile(f)} (s.t_e - s.t_s) \cdot (\sum_{j=1}^{D} s.a_{max}^{(j)} - \sum_{j=1}^{D} s.a_{min}^{(j)})$. Based on these notations, the *total flexibility* of f is defined as follows:

Definition 2: *The total flexibility of an flex-object* f *is the product of the time flexibility and the amount flexibility, i.e.,* $flex(f) = tf(f) \cdot af(s)$.

Consider a flex-object f=$([2, 7], s^{(1)}, s^{(2)})$ where $s^{(1)} = ([0, 1], [10, 20])$ and $s^{(2)} = ([1, 4], [6, 10])$. The time flexibility of f is equal to $7 - 2 = 5$. The amount flexibility $af(f)$ is equal to $(1 - 0)(20 - 10) + (4 - 1)(10 - 6) = 22$. Hence, the total flexibility of f is equal to 110.

A flex-object with time and profile flexibilities equal to zero is called a *fix-object.* In this case, the fix-object $f = ([t_{es}, t_{ls}], s^{(1)}, \ldots, s^{(m)})$ is such that $t_{es} = t_{ls}$ and $s.a_{min}^{(d)} = s.a_{max}^{(d)}$ $\forall s \in profile(f), \forall d = 1..D$.

Definition 3: *An* instance *(instantiation) of a flex-object* $f = ([t_{es}, t_{ls}], s^{(1)}, \ldots, s^{(m)})$ *is a fix-object* $f_x = ([t_s, t_s], s_x^{(1)}, \ldots, s_x^{(m)})$ *such that* $t_{es} \leq t_s \leq t_{ls}$ *and* $\forall i = 1..m, d = 1..D : s^{(i)}.a_{min}^{(d)} \leq s_x^{(i)}.a_{min}^{(d)} = s_x^{(i)}.a_{max}^{(d)} \leq s^{(i)}.a_{max}^{(d)}$. *We refer to* t_s *as the* start time *of the flex-object* f.

There is an infinite number of possible instances (fix-objects) of a flex-object. One possible instance is shown as the dotted line in Figure 1. We can now define *aggregation* and *disaggregation* as follows:

Definition 4: *Let* AGG *be an aggregate function which takes a set of flex-objects F and produces a set of flex-objects A. Here, every $f_a \in A$ is called an* aggregated flex-object*, and $|A| \leq |F|$.*

Definition 5: *Let* DISAGG *be a function which takes a set of A instances and produces a set of F instances. We denote these sets of fix-objects as A_X and F_X, respectively, and assume that $A = AGG(F)$, $\forall f \in F \Leftrightarrow \exists f^x \in F_X$ and $\forall f_a \in A \Leftrightarrow \exists f_a^x \in A_X$. Moreover, to ensure the balance of amounts at aggregated and non-aggregated levels, for all time units $T = 0, 1, 2, \cdots$ and all dimensions $d = 1..D$, the following equality must hold:*

$$\sum_{t=0}^{T} \left[s.a_{min}^{(d)} | \forall f_a^x \in A_X, \forall s \in profile(f_a^x), s.t_e \leq t \right] =$$

$$\sum_{t=0}^{T} \left[s.a_{min}^{(d)} | \forall f_x \in F_X, \forall s \in profile(f_x), s.t_e \leq t \right].$$

Evaluation of the functions AGG and $DISAGG$ is called flex-object *aggregation* and *disaggregation*, respectively. Due to the amount balance requirement, disaggregation is, however, not always possible for any arbitrary AGG function. Depending on whether disaggregation is possible or not for all instances of aggregated flex-objects, we identify two types of flex-object aggregation: *conservative* and *greedy*, respectively. Aggregated flex-objects resulting from greedy aggregation might define more time and amount flexibilities compared to the original flex-objects. Obviously, instances of such flex-objects might not be disaggregated using $DISAGG$. Nevertheless, this type of aggregation is still important in feasibility/risk analysis where extreme amount values are explored (see *flexibility availability queries* in Section 2). On the contrary, aggregated flex-objects resulting from conservative aggregation always define less (or equal) flexibilities compared to the original flex-objects. Consequently, it is always possible to disaggregate instances of such flex-objects using $DISAGG$. Conservative aggregation is important in use-cases where planning is involved, e.g., in MIRABEL. The flex-object database has to support both types of aggregation, however, in this paper, we focus on conservative aggregation only.

The following requirements for the aggregation originate from the MIRABEL use-case, but they are also important for general flex-object aggregation:

Compression and flexibility trade-off requirement. It must be possible to control the trade-off between (1) the number of aggregated flex-objects and (2) the flexibility loss, i.e., difference between the total flex-object flexibility (see Definition 2) before and after aggregation.

Aggregate constraint requirement. Every aggregated flex-object $f_a \in AGG(F)$ must satisfy a user-defined so-called *aggregate constraint* C, which is satisfied if the value of a certain flex-object attribute, e.g., *total maximum amount*, is within the given bounds. For example, Such constraints can ensure that aggregated flex-objects are "properly shaped" to meet energy market rules and power grid constraints.

Incremental update requirement. Flex-object updates (addition/removal) should be processed efficiently and cause minimal changes to the set of aggregated flex-objects. Thus functionality is vital in scenarios, e.g., MIRABEL, where addition/removal of flex-objects are very frequent.

In the following sections, we present a technique to perform flex-object aggregation and disaggregation while satisfying all requirements.

4 Aggregation and Disaggregation

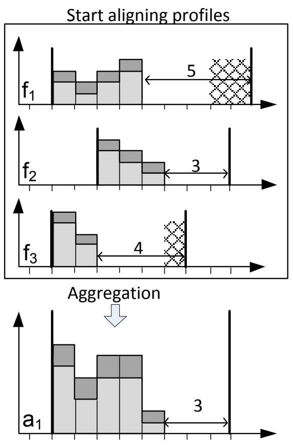
In this section, we first propose a basic N-to-1 flex-object aggregation algorithm and explain how to generalize for it a large set of flex-objects. Additionally, we explain how disaggregation can be performed.

According to the flex-object definition, the profile start time ($s^{(1)}.t_s$) of a flex-object f is not pre-determined, but must be between the earliest start time $f.t_{es}$ and the latest start time $f.t_{ls}$. Hence, the aggregation of even two flex-objects is not straightforward. Consider aggregating two flex-objects f_1 and f_2 with time flexibility values equal to six and eight, respectively. Thus, we have 48 ($6*8$) different profile start time combinations, each of them realizing a different aggregated flex-object. Three possible profile start time parameter combinations are shown in Figure 2(a-c).

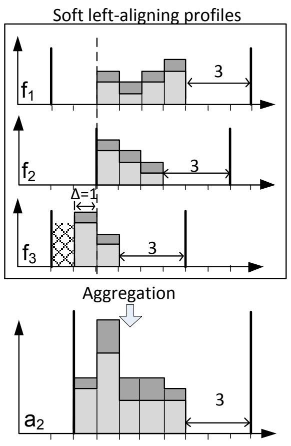
In general, to aggregate a set of flex-objects F into a single aggregated flex-object f_a, we follow these three steps:

1. Choose a profile start time value $f.s^{(1)}.t_s = s_f \ \forall f \in F$ such that $f.t_{es} \leq s_f \leq f.t_{ls}$. Later, we will refer to this choice of profile start time as *profile alignment*.
2. Set the time flexibility interval for f_a such that $f_a.t_{es} = min_{f \in F}(s_f)$ and $f_a.t_{ls} = f_a.t_{es} + min_{f \in F}(f.t_{ls} - s_f)$.
3. Build a profile for f_a by summing the corresponding amounts for each slice across all profiles.

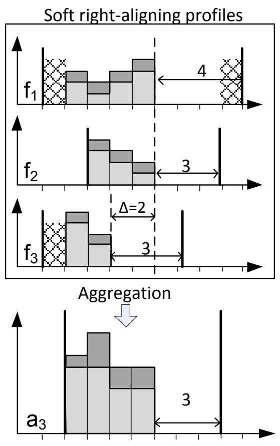
There are many ways to align profiles (by choosing the constants $s_{f_1}, s_{f_2}, \ldots, s_{f_{|F|}}$). Each of these alignments determine where amounts from individual flex-objects are concentrated within the profile of f_a. We focus only on the three most important alignment options: *start-alignment*, *soft left-alignment*, and *soft right-alignment*. Start-alignment spreads out amounts throughout the time extent of all individual flex-objects, making larger amounts available as early as possible. On the contrary, soft left-/right-alignment builds shorter profiles with amounts concentrated early (left) or late (right) in the profile. In the context of MIRABEL, start-alignment is suitable for the near real-time balancing of electricity, where energy has to be available as early as possible; and soft left/right alignment allows the consumption of anticipated wind production peaks with steep rises (left-alignment) or falls (right-alignment). The three alignment options are illustrated in Figure 2. Here, the crossed area in the figure represents the amount of time flexibility that is lost due to aggregation when different profile alignment options are used. The alignment option are elaborated below:



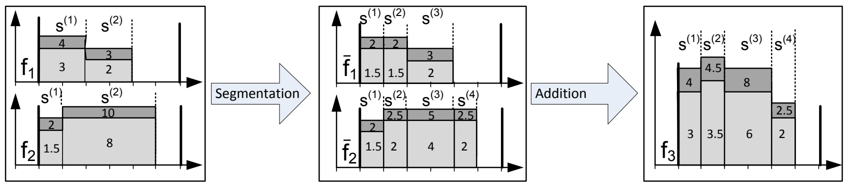
(a) Start-alignment



(b) Soft left-alignment



(c) Soft right-alignment



(d) Aggregating profiles

Fig. 2. Profile alignment and aggregation

Start-alignment. We set s_{f_1}, s_{f_2}, $\ldots$, $s_{f_{|F|}}$ so that $\forall f \in F : s_f = f.t_{es}$. This ensures that profiles are aligned at their respective *earliest start time* values (see f_1 and f_2 in Figure 2(a)).

Soft left-alignment. We set s_{f_1}, s_{f_2},..., $s_{f_{|F|}}$ so that $\forall f \in F : s_f = min(f.t_{ls} - min_{g \in F}(g.t_{ls} - g.t_{es}), max_{g \in F}(g.t_{es}))$. Figure 2(b) illustrates the effect of soft left-alignment. Here, f_1 and f_2 are left-aligned, meaning that their profile start times are equal. However, the profile of f_3 cannot be left-aligned with respect to the profiles of f_1 and f_2 as that would shorten the remaining time flexibility range of the aggregate. f_3 lacks one time unit ($\Delta = 1$) for its profile to left-align.

Soft right-alignment. We set $s_{f_1}, s_{f_2}, ..., s_{f_{|F|}}$ so that $\forall f \in F : s_f = min(f.t_{ls} - min_{g \in F}(g.t_{ls} - g.t_{es}), max_{g \in F}(g.t_{es} + p_{dur}(g)) - p_{dur}(f)$. Figure 2(c) illustrates the effect of soft right-alignment. Here, f_1 and f_2 are right-aligned, meaning that their profiles align at the right hand side (i.e., have equal $t_{es} + p_{dur}$ values). However, the profile of f_3 cannot be right-aligned with respect to the profiles of f_1 and f_2 as this would shorted the remaining time flexibility range of the aggregate.

After the alignment, the time flexibility interval is computed for the aggregated flex-object. As illustrated in Figure 2(a-c), for all three alignment options, the time flexibility of f_a is equal to that of the flex-object with the smallest time flexibility in the set F, i.e., $f_a.t_{ls} - f_a.t_{es} = min_{f \in F}(f.t_{ls} - f.t_{es})$. However, other types of alignment, e.g., *hard left* or *hard right* where all profiles are forced to align at the left or right hand side, might reduce the time flexibility of the aggregated flex-object.

Finally, the minimum and maximum amounts of adjacent slices in the aligned profiles are summed to construct the profile of the aggregated flex-object. If adjacent slices at any time unit have different durations, those slices are partitioned to unify their durations. During the partitioning, minimum and maximum amounts are distributed proportionally to the duration of each divided slice. This step is called *segmentation*. The segmentation step reduces the amount flexibility, $af(f)$, as it imposes more restrictions on the amount for each divided segment. For example, consider the slice $s^{(1)}$ of f_1 in Figure 2(d), which illustrates the segmentation for two flex-objects. Originally, the minimum amount is 3 and the maximum amount is 4 over two time units. Thus, we can supply one amount unit in the first time unit, and three units in the second time unit. However, this supply is not acceptable after dividing the slice into two equal-sized slices $s^{(1)}$ and $s^{(2)}$ with minimum and maximum amount of 1.5 and 2, respectively. After the segmentation, the addition of profiles is performed. During the addition, $a_{min}^{(1)}$ and $a_{max}^{(1)}$ amounts are added for every corresponded profile slice.

It is always possible to disaggregate a flex-object produced by this aggregation approach. Consider the following disaggregation procedure. For a given instance f_a^x of flex-object f_a, we produce the set of fix-objects $\{f_1^x, f_2^x, ..., f_{|F|}^x\}$ such that $\forall i = 1..|F| : f_i^x.t_{es} = f_i^x.t_{ls} = s_{f_i} + (f_a^x.t_{es} - f_a.t_{es})$. It is always possible to fix the *start time* of every $f_i^x, i = 1..|F|$ because the time flexibility range of the aggregate f_a is computed conservatively, and the aligned profiles of $f \in F$ can always be shifted within this range (see Figure 2). Also, the amount values from every slice $s_a^x \in profile(f_a^x)$ are distributed proportionally to the respective slices of $f_i^x, i = 1..|F|$ so that minimum and maximum amount constraints are respected for every $f \in F$. This can always be achieved, and consequently, for any instance of f_a, it is always possible to build instances of flex-objects from F. Moreover, the newly built fix-objects will collectively define a total amount which is equal to that of the initial fix-object f_x^a.

To summarize, the N-to-1 aggregation approach can be used to aggregate flex-objects in F. However, the time (and total) flexibility loss depends on the flex-object with smallest time flexibility in the set F. Due to this issue, much of the flexibility will be lost when aggregating flex-objects with distinct time flexibilities. To address this, we will now propose an N-to-M aggregation approach.

5 N-to-M Aggregation

As discussed in Section 4, aggregating "non-similar" flex-objects results in an unnecessary loss of time flexibility. This loss can be avoided, and the profile alignments can be better enforced, by carefully grouping flex-objects and thus ensuring that their time flexibility intervals overlap substantially. We now describe an (N-to-M) approach to aggregate a set of flex-objects, $\mathcal{F}$, to a set of aggregated flex-objects, $\mathcal{A}$, while satisfying all requirements (see Section 3). The algorithm consists of three phases: *grouping*, *bin-packing*, and *N-to-1 aggregation*:

Grouping phase. We partition the input set $\mathcal{F}$ into disjoint groups of similar flex-objects. Based on the application scenario, the user specifies which attributes to use in the grouping. For example, the user may choose the earliest start time, latest start time, time flexibility, and/or amount flexibility as grouping attributes. Two flex-objects are grouped together if the values of the specified attributes differ no more than a user-specified threshold. The thresholds and the associated grouping attributes are called *grouping parameters*. As shown later, the choice of grouping parameters yields a trade-off between compression and flexibility loss. In the example in Figure 3, flex-objects $f_1, f_2, ...,$ and f_5 are assembled in two groups g_1 and g_2 during the grouping phase.

Bin-packing phase. This phase enforces the aggregate constraint (see Section 3). Each group g produced in the *grouping* phase is either passed to the next phase (if g satisfies the constraint already) or further partitioned into the minimum number of bins (groups) such that the constraint $w_{min} \leq w(b) \leq w_{max}$ is satisfied by each bin b. Here, $w(b)$ is a weight function, e.g., $w(b) = |b|$, and w_{max} and w_{min} are the upper and lower bounds. We refer to w_{min}, w_{max}, and w as *bin-packing parameters*. By adjusting these parameters, groups with a bounded number of flex-objects or a bounded total amount can be built. Note that it may be impossible to satisfy a constraint for certain groups. For example, consider a group with a single flex-object, while we impose a lower bound of two flex-objects in all groups. These groups are discarded from the output (see g_{22} in Figure 3), and, depending on the application, these flex-objects can be either: (1) excluded from the N-to-M aggregation output, or (2) aggregated with another instance of the N-to-M aggregation with less constraining grouping or bin-packing parameters.

N-to-1 Aggregation phase. We assemble the output set $\mathcal{A}$ by applying N-to-1 aggregation (see Section 4) for each resulting group g. The alignment option is specified as the *aggregation parameter*. Every aggregated flex-objects satisfies the aggregate constraint enforced in the bin-packing phase.

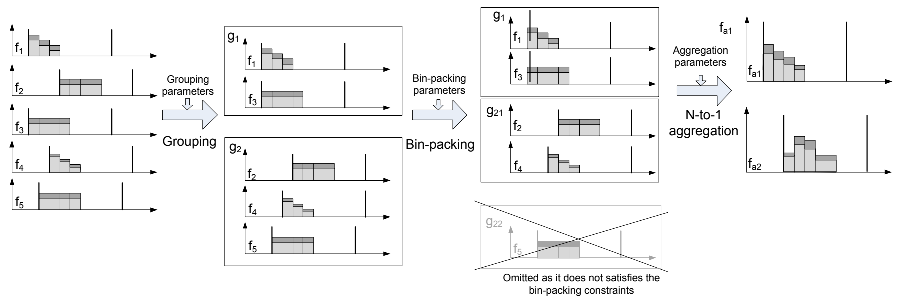


Fig. 3. Aggregation of flex-objects in the N-to-M aggregation approach

The complete N-to-M aggregation process is visualized in Figure 3. Here, given the initial flex-object set $\{f_1, f_2, ..., f_5\}$ and grouping, bin-packing, and aggregation parameters, two aggregated flex-objects, f_{a1} and f_{a2}, are produced. The grouping parameters are set so that the difference between the earliest start time (t_{es}) is at most 2. The bin-packing parameters require that the number of flex-objects in resulting groups are 2, i.e., $w_{min} = w_{max} = 2$, $w(g) = |g|$. In the aggregation phase, the *start-aligned* option is used. In practice, the user will choose from a number of meaningful pre-defined parameter settings, e.g., *short/long profiles* or *amount as early as possible.*

6 Incremental N-to-M Aggregation

In this section, we present an incremental version of the N-to-M aggregation approach. The set of flex-objects F is updated with a sequence of incoming updates: u_1, u_2,...,u_k. Each update u_i is of the form (f, c_i), where f is a flex-object and $c_i \in \{+, -\}$ indicates insertion ('+') or deletion ('$-$') of f to/from F. The incremental approach outputs a sequence of aggregated flex-object updates which correspond to u_1, u_2,...,u_k. The approach has four phases: *grouping*, *optimization*, *bin-packing*, and *aggregation*.

Grouping phase. We map each flex-object into a d-dimensional point. This point belongs to a cell in a d-dimensional uniform grid. Users specify the extent of each cell in each dimension using thresholds T_1, T_2, ..., T_d from the grouping parameters. Every cell is identified by its coordinates in the grid. We only keep track of *populated* cells, using an in-memory hash table, denoted as the *group hash.* This table stores key-value pairs, where the key is the cell coordinates and the value is the set of flex-objects from F mapped to this cell. We combine adjacent populated cells into a *group.* A group can be either *created*, *deleted*, or *modified.* Group changes are stored in a list, denoted as the *group changes list.* Figure 4 visualizes the effect of adding a flex-object f_1. f_1 is mapped to a 2-dimensional point which lies in the grid cell c_2. The coordinates of c_2 are used to locate a group in the group hash. The found group is updated by inserting f_1 into its list of flex-objects. Finally, a change record indicating that the group was *modified* is inserted into the group changes list. In the cases when a group is not found in the group hash, a new group with an unique *id* and a single populated cell c_2 is created. Also, if the group changes list already contains a change record for a particular group, the record is updated to reflect the combination of the changes.

Optimization phase. This phase is only executed when aggregation is triggered, either (1) periodically, (2) after a certain number of updates, or (3) when the latest aggregates are requested. During this phase, we consolidate the group changes list. For each update of a group g in the list, we identify its adjacent groups g_1^a, g_2^a, ... by probing the group-hash. Then, for each adjacent group g_i^a, a minimum bounding rectangle (MBR) is computed over all points that contains flex-objects from the groups g and g_i^a. If the extent of the MBR in all dimensions are within the user-specified thresholds, we combine the groups g and g_i^a

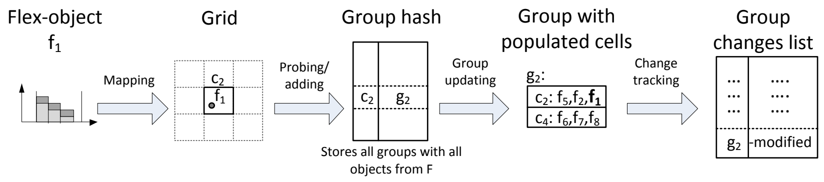


Fig. 4. Processing the addition of a flex-object in the grouping phase

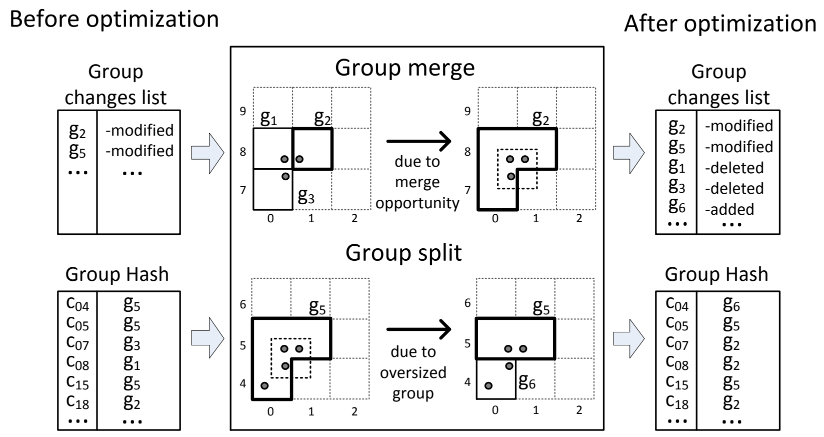


Fig. 5. Flow of data in the optimization phase

(see merge in Figure 5). Otherwise, if the MBR of g in any dimension is larger than the size of a grid cell, we perform a group split (depicted in Figure 5). Any over-sized group is partitioned into groups of a single grid cell, and, for every individual group, an MBR is computed. Then, the two groups with the closest MBRs are merged until the grouping constraint is violated. Then, g is substituted the with newly built groups. Groups changes incurred during merging and splitting are added to the group change list.

Bin-packing phase. We maintain a hash table, denoted the *bin hash*, which maps from each group, produced in the grouping phase, to its bins (as described in Section 5). In this phase, we propagate updates from the group change list to bins. We first compare existing bins with an updated group to compute the deltas to obtain added and deleted flex-objects, Δ_{added} and Δ_{delete}, respectively. Then, we discard from the bins the flex-objects that are in Δ_{delete}. Groups with total weight less than w_{min} are deleted and flex-objects from these groups as well as from Δ_{added} are included into other existing bins using the first fit decreasing strategy [22]. New bins are created, if needed.

Figure 6 shows how the bins of the group g_7 are updated when lower and upper bounds w_{min} and w_{max} are set. Finally, all bins changes are pipelined to the aggregation phase. Flex-objects that did not fit to any bin (due to their weight being lower than w_{min} or higher than w_{max}) are stored in a separate list.

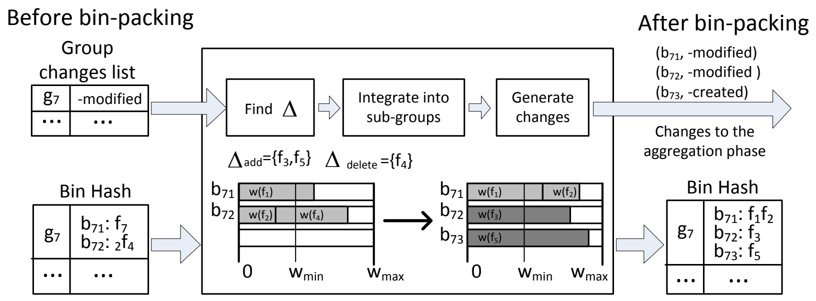


Fig. 6. Flow of data in the bin packing phase

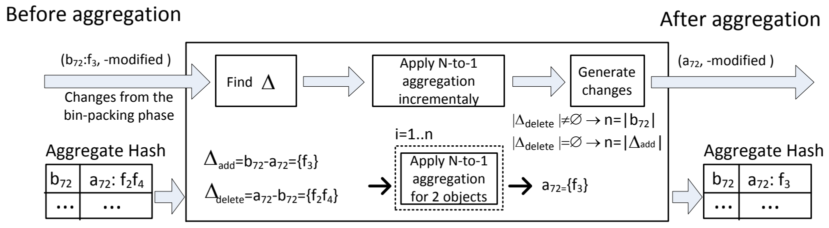
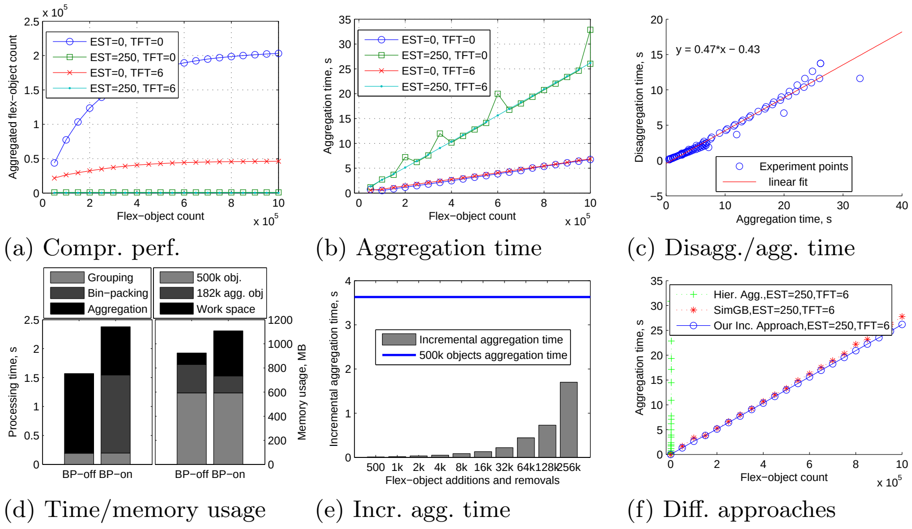


Fig. 7. Aggregation phase

Aggregation phase. We maintain a hash table, denoted as the *aggregate hash*, which maps from each individual bin to an aggregated flex-object. Each aggregated flex-object has references to the original flex-objects. Thus, for every bin change, added and deleted non-aggregated flex-objects (see Δ_{add} and Δ_{delete} in Figure 7) are found and used to incrementally update an aggregate flex-object. If there are no deletes, N-to-1 aggregation is applied for every added object. Otherwise, an aggregate is recomputed from scratch by applying N-to-1 aggregation on all object in a bin. Finally, all changed aggregated flex-objects are provided as output.

7 Experimental Evaluation

In this section, we present the experimental evaluation of the full incremental N-to-M aggregation approach. As there are no other solutions for flex-object aggregation and disaggregation, we propose two rival implementations: *Hierarchical Aggregation* and *SimGB*. In *Hierarchical Aggregation*, we use agglomerative hierarchical clustering for the grouping phase. First, the approach assigns each flex-object to individual clusters. Then, while no grouping constraints are violated, it incrementally merges the two closest clusters. The distance between two clusters is calculated based on the values of the grouping parameter flex-object attributes. For *SimGB*, we apply the *similarity group-by operator* [19] for one grouping parameter at a time, thus partitioning the input into valid groups of similar flex-objects. For the evaluation, We use a synthetic flex-object



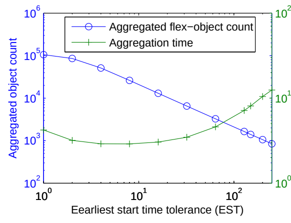
(a) Compr. perf. (b) Aggregation time (c) Disagg./agg. time

(d) Time/memory usage (e) Incr. agg. time (f) Diff. approaches

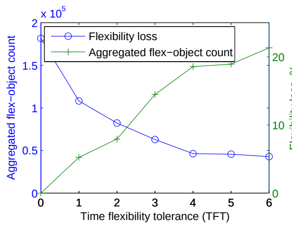
Fig. 8. Results of the scalability and incremental behavior evaluation

dataset from the the MIRABEL project. The dataset contains one million energy consumption request flex-objects. The *earliest start time* (t_{es}) is distributed uniformly in the range $[0, 23228]$. The number of slices and the time flexibility values ($t_{ls} - t_{es}$) follow the normal distributions $\mathcal{N}(8, 4)$ and $\mathcal{N}(20, 10)$ in the ranges $[10, 30]$ and $[4, 12]$, respectively; the slice duration is fixed to 1 time unit for all flex-objects, thus profiles are from 2.5 to 7.5 hours long. Experiments were run on a PC with Quad Core Intel®Xeon®E5320 CPU, 16GB RAM, OpenSUSE 11.4 (x86_64), and Java 1.6. Unless otherwise mentioned, the default values of the experiment parameters are: (a) The number of flex-objects is $500k$. (b) $EST = 0$ (*Earliest Start Time Tolerance*) and $TFT = 0$ (*Time Flexibility Tolerance*) are used as the grouping parameters. They apply on the *Earliest Start Time* (t_{es}) and *Time Flexibility* ($t_{ls} - t_{es}$) flex-object attributes, respectively. (c) The aggregate constraint is unset (bin-packing is disabled). We also perform experiments with bin-packing enabled (explicitly stated).

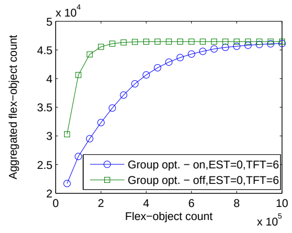
Scalability For evaluating flex-object compression performance and scalability, the number of flex-offers is gradually increased from $50k$ to $1000k$. Aggregation is performed using two different EST and TFT parameter values: EST equal to 0 or 250, and TFT equal to 0 or 6. Disaggregation is executed with randomly generated instances of aggregated flex-objects. The results are shown in Figure 8(a-d). Figure 8(a-b) shows that different aggregation parameter values lead to different compression factors and aggregation times. Disaggregation is approx. 2 times faster than aggregation (see Figure 8(c)) regardless of the flex-object count and grouping parameter values. Most of the time is spent in the bin-packing (if enabled) and N-to-1 aggregation phases (the 2 left bars in Figure 8(d)). Considering the overhead associated with incremental behavior, the amount of memory used by the approach is relatively small compared to the



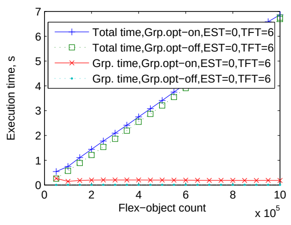
(a) Effect of EST



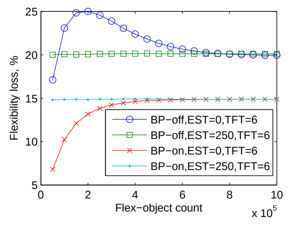
(b) Effect of TFT



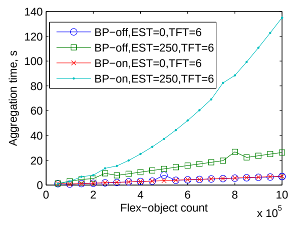
(c) Grp. opt. effect



(d) Grp. opt. cost



(e) Bin-packing effect



(f) Bin-packing cost

Fig. 9. Results of the grouping, optimization, and bin-packing evaluation

footprint of the original and aggregated flex-objects. Memory usage increases when bin-packing is enabled.

Incremental Behavior. When evaluating incremental aggregation performance, we first aggregate 500k flex-objects. Then, for different k values ranging from 500 to 256k, we insert k new flex-objects and remove k randomly selected flex-objects. The total number of flex-objects stays at 500k. For every value of k, we execute incremental aggregation. As seen from Figure 8(e), the updates can be processed efficiently so our approach offers substantial time savings compared to the case when all 500k flex-objects are aggregated from scratch (the line in the figure). We then compare the total time to process flex-objects with our incremental approach to the other two (inherently non-incremental) approaches - *Hierarchical Aggregation* and *SimGB*. As seen in Figure 8(f), our incremental approach is competitive to SimGB in terms of scalability. The overhead associated with the change tracking in our approach is not significant in the overall aggregation time. Additionally, the hierarchical clustering-based approach (Hier. Agg.) incurs very high processing time even for small datasets (due to a large amount of distance computations), and is thus not scalable enough for the flex-object aggregation problem.

Grouping Parameters Effect. As seen in Figure 9(a), the *EST* significantly affects the flex-object compression factor. For this dataset, increasing *EST* by a factor of two leads to a flex-object reduction by approximately the same factor. However, the use of high *EST* values results in aggregated flex-object profiles with more slices. Aggregating these requires more time (see “aggregation time” in Figure 9(a)). The *TFT* parameter has a significant impact on the flexibility loss (see “flexibility loss” in Figure 9(b)). Higher values of *TFT* incur higher flexibility losses. When it is set to 0, aggregation incurs no flexibility loss, but results in a larger amount of aggregated flex-objects. When the number of distinct time flexibility values in a flex-object dataset is low (as in our case), the best compression with no flexibility losses can be achieved when $TFT = 0$ and the other grouping parameters are unset (or set to high values). However, due to the long durations of profiles and high total amount values, the produced aggregated flex-object might violate the aggregate constraint.

Optimization and Bin-packing. We now study the optimization and bin-packing phases. As seen in Figure 9(c-d), the optimization phase is relatively cheap (Figure 9(d)), and it substantially contributes to the aggregated flex-object count reduction (Figure 9(c)). For bin-packing evaluation, the aggregate constraint was set so that the time flexibility of an aggregate is always at least 8 ($w_{min} = 8$, equiv. to 2 hours). By enabling this constraint, we investigate the overhead associated to bin-packing and its effect on the flexibility loss. As seen in Figure 9(e), by bounding the time flexibility for every aggregate, the overall flexibility loss can be limited. However, bin-packing introduces a substantial overhead that depends on the number of objects in flex-object groups after the optimization phase (see Figure 9(f)). When this number is small ($EST = 0$, $TFT = 6$), the overhead of bin-packing is insignificant. However, when groups are large ($EST = 250$, $TFT = 6$), bin-packing overhead becomes very significant.

In summary, we show that our incremental aggregation approach scales linearly in the number of flex-object inserts. The overhead associated with incremental behavior is insignificant. Our approach performs aggregation *incrementally* just as fast as efficient non-incremental grouping approaches (*SimGB*). The trade off-between flex-object compression factor and flexibility loss can be controlled using the grouping parameters. The compression factor can be further increased efficiently by group optimization. Disaggregation is approx. 2 times faster than aggregation.

8 Related Work

Related research fall in several categories.

Clustering. Many clustering algorithms have been proposed, including density-based (e.g., BIRCH [24]), centroid-based (e.g., K-Means [13]), hierarchical clustering (e.g., SLINK [18]), and incremental algorithms such as incremental K-means [25] and incremental BIRCH [10]. In comparison to our approach, clustering solves only the grouping part of the problem, which is a lot simpler than the whole problem. For grouping alone, the closest work is incremental grid-based clustering [16,9,12], where we, in comparison, improve the clusters across the grid boundaries and limit the number of items per each cluster.

Similarity Group By. SimDB [20] groups objects based on the similarity between tuple values, and is implemented as a DBMS operator in [19]. However, SimDB again only solves the grouping part of the problem, and is (unlike our approach) not incremental, which is essential for us.

Complex objects. Complex objects with multidimensional data exists in many real-world applications [14] and can be represented with *multidimensional data models* [17]. Several research efforts (e.g., [5] and [23]) have been proposed to aggregate complex objects. However, these efforts do not consider the specific challenges related to aggregating flex-objects.

Temporal Aggregation. Several papers have addressed aggregation for temporal and spatio-temporal data including: instantaneous temporal aggregation [3], cumulative temporal aggregation [21,1,11], histogram-based aggregation [6] and multi-dimensional temporal aggregation [4]. These techniques differ in the way how a time line is partitioned into time intervals and how an aggregation group is associated with each time instant. The efficient computation of these time intervals poses a great challenge and therefore various techniques that allow computing them efficiently are proposed [8,7,15]. Unfortunately, these techniques only deal with simple data items without flexibilities, making them unsuitable for aggregation of flex-objects.

9 Conclusion and Future Work

Objects with inherent flexibilities, so-called flexibility objects (flex-objects), occur in both scientific and commercial domains. Managing flex-objects with existing DBMSs is infeasible due to their complexity and data volume. Thus, a new tailor-made database for flex-objects is needed. This paper was the first to discuss flex-object databases, focusing on the most important operations of a flex-object database: aggregation and disaggregation. The paper formally defined the concept of flexibility objects and provided a novel and efficient grid-based solution considering the grouping of flex-objects, alternatives for computing aggregates, the disaggregation process, and the requirements associated to these. The approach allowed efficient incremental computation. Extensive experiments on data from a real-world energy domain project showed that the apporach provided very good performance while satisfying all entailed requirements.

As future work, other challenges related to the flex-object database have to be addressed. These include flex-object storage and visualization, as well as support for other types of queries (flexibility availability, adjustment potential,..). Another interesting topic is aggregation and disaggregation techniques for flex-objects with flexibility in the profile slices durations.

References

1. Arasu, A., Widom, J.: Resource sharing in continuous sliding-window aggregates. In: Proc. of VLDB. pp. 336–347 (2004)
2. Boehm, M., Dannecker, L., Doms, A., Dovgan, E., Filipic, B., Fischer, U., Lehner, W., Pedersen, T.B., Pitarch, Y., Siksnys, L., Tusar, T.: Data management in the mirabel smart grid system. In: Proc. of EnDM (2012)
3. Böhlen, M.H., Gamper, J., Jensen, C.S.: How would you like to aggregate your temporal data? In: Proc. of TIME. pp. 121–136 (2006)

4. Böhlen, M.H., Gamper, J., Jensen, C.S.: Multi-dimensional aggregation for temporal data. In: Proc. of EDBT. pp. 257–275 (2006)
5. Cabot, J., Mazón, J.N., Pardillo, J., Trujillo, J.: Specifying aggregation functions in multidimensional models with ocl. In: Proc. of ER. pp. 419–432 (2010)
6. Chow, C.Y., Mokbel, M.F., He, T.: Aggregate location monitoring for wireless sensor networks: A histogram-based approach. In: Proc. of MDM. pp. 82–91 (2009)
7. Gao, D., Gendrano, J.A.G., Moon, B., Snodgrass, R.T., Park, M., Huang, B.C., Rodrigue, J.M.: Main memory-based algorithms for efficient parallel aggregation for temporal databases. Distributed Parallel Databases 16(2), 123–163 (Sep 2004)
8. Gordevičius, J., Gamper, J., Böhlen, M.: Parsimonious temporal aggregation. In: Proc. of EDBT. pp. 1006–1017 (2009)
9. Hou, G., Yao, R., Ren, J., Hu, C.: A clustering algorithm based on matrix over high dimensional data stream. In: Proc. of WISM. pp. 86–94 (2010)
10. Jensen, C.S., Lin, D., Ooi, B.C.: Continuous clustering of moving objects. IEEE Trans. Knowl. Data Eng. 19(9), 1161–1174 (2007)
11. Jin, C., Carbonell, J.G.: Incremental aggregation on multiple continuous queries. In: Proc. of ISMIS. pp. 167–177 (2006)
12. Lei, G., Yu, X., Yang, X., Chen, S.: An incremental clustering algorithm based on grid. In: Proc. of FSKD. pp. 1099–1103. IEEE (2011)
13. Macqueen, J.B.: Some methods of classification and analysis of multivariate observations. In: Proc. of 5th Berkeley Symposium on Math. Stat. and Prob. pp. 281–297 (1967)
14. Malinowski, E., Zimnyi, E.: Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications. Springer, 1 edn. (2008)
15. Moon, B., Fernando Vega Lopez, I., Immanuel, V.: Efficient algorithms for large-scale temporal aggregation. TKDE 15(3), 744–759 (March 2003)
16. Park, N.H., Lee, W.S.: Statistical grid-based clustering over data streams. SIGMOD Rec. 33(1), 32–37 (2004)
17. Pedersen, T.B., Jensen, C.S., Dyreson, C.E.: A foundation for capturing and querying complex multidimensional data. Information Systems 26(5), 383–423 (Jul 2001)
18. Sibson, R.: SLINK: An optimally efficient algorithm for the single-link cluster method. The Computer Journal 16(1) (Jan 1973)
19. Silva, Y.N., Aly, A.M., Aref, W.G., Larson, P.A.: SimDB: A similarity-aware database system. In: Proc. of SIGMOD (2010)
20. Silva, Y.N., Aref, W.G., Ali, M.H.: Similarity group-by. In: Proc. of ICDE. pp. 904–915 (2009)
21. Yang, J., Widom, J.: Incremental computation and maintenance of temporal aggregates. VLDB 12(3), 262–283 (Oct 2003)
22. Yue, M.: A simple proof of the inequality $FFD(L) \leq \frac{11}{9}OPT(L) + 1, \forall L$ for the FFD bin-packing algorithm. Acta Mathematicae Applicatae Sinica 7(4), 321–331 (Oct 1991)
23. Zhang, D.: Aggregation computation over complex objects. Ph.D. thesis, University of California, Riverside, USA (2002)
24. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: an efficient data clustering method for very large databases. In: Proc. of SIGMOD. pp. 103–114 (1996)
25. Zhang, Z., Yang, Y., Tung, A.K.H., Papadias, D.: Continuous k-means monitoring over moving objects. IEEE Trans. Knowl. Data Eng. 20(9), 1205–1216 (2008)