A One-Pass Aggregation Algorithm with the Optimal Buffer Size in Multidimensional OLAP

Young-Koo Lee[†], Kyu-Young Whang[†], Yang-Sae Moon[†], and Il-Yeol Song[††]

[†] Department of Computer Science and
Advanced Information Technology Research Center (AITrc)
Korea Advanced Institute of Science and Technology (KAIST), Taejon, Korea
{yklee,kywhang,ysmoon}@mozart.kaist.ac.kr
[††] College of Information Science and Technology,
Drexel University, Philadelphia, Pennsylvania, USA
song@drexel.edu

Abstract

Aggregation is an operation that plays a key role in multidimensional OLAP (*MOLAP*). Existing aggregation methods in MOLAP have been proposed for file structures such as multidimensional arrays. These file structures are suitable for data with uniform distributions, but do not work well with skewed distributions. In this paper, we consider an aggregation method that uses dynamic multidimensional files adapting to skewed distributions. In these multidimensional files, the sizes of page regions vary according to the data density in these regions, and the pages that belong to a larger region are accessed multiple times while computing aggregations. To solve this problem, we first present an aggregation computation model, called the *Disjoint-Inclusive Partition (DIP) computation model*, that is the formal basis of our approach. Based on this model, we then present the one-pass aggregation algorithm. This algorithm computes aggregations using the *one-pass buffer size*, which is the minimum buffer size required for guaranteeing one disk access per page. We prove that our aggregation algorithm is optimal with respect to the one-pass buffer size under our aggregation computation model. Using the DIP computation model allows us to correctly predict the order of accessing data pages in advance. Thus, our algorithm achieves the optimal one-pass buffer size by using a buffer replacement policy, such as Belady's B_0 or Toss-Immediate policies, that exploits the page access order computed in advance. Since the page access order is not known *a priori* in general, these policies have been known to lack practicality despite its theoretic significance. Nevertheless, in this paper, we show that these policies can be effectively used for aggregation computation.

We have conducted extensive experiments. We first demonstrate that the one-pass buffer size theoretically derived is indeed correct in real environments. We then compare the performance of the one-pass algorithm with those of other ones. Experimental results for a real data set show that the one-pass algorithm reduces the number of disk accesses by up to 7.31 times compared with a naive algorithm. We also show that the memory requirement of our algorithm for processing the aggregation in one-pass is very small being 0.05%∼0.6% of the size of the database. These results indicate that our algorithm is practically usable even for a fairly large database. We believe our work provides an excellent formal basis for investigating further issues in computing aggregations in MOLAP.

Proceedings of the 28th VLDB Conference,
Hong Kong, China, 2002

1 Introduction

On-line analytical processing (OLAP) is a database application that allows users to easily analyze large volumes of data in order to extract the information necessary for decision-making [4]. OLAP queries make heavy use of aggregation for summarizing data since summarized trends derived from the records are more useful for decision-making rather than individual records themselves. Since computing aggregation is very expensive, good aggregation algorithms are crucial for achieving performance in OLAP systems [1, 10, 14, 24].

OLAP is based on a multidimensional data model that employs multidimensional arrays for modeling data [4]. The multidimensional data model consists of *measures* and *dimensions*: measures are the attributes that are analyzed; dimensions are the attributes that determine the values of the measures. A dimension is mapped to an axis of the multidimensional array. A measure is mapped to a value stored in a cell. This model allows OLAP users to analyze changes in the values of the measures according to changes in the values of the dimensions.

OLAP systems are categorized into two classes according to their storage structures: relational OLAP (ROLAP) and multidimensional OLAP (MOLAP) [4]. ROLAP, built on top of the relational database system, stores OLAP data in tables. In contrast, MOLAP uses multidimensional files that can efficiently store and manage multidimensional data. Recently, as the effectiveness of the multidimensional files on OLAP is recognized, there have been attempts to use them even in ROLAP [10, 24].

While aggregation methods for ROLAP have been extensively studied, the corresponding work for MOLAP has been rare. MOLAP primarily uses static methods that materialize precomputed aggregates [10, 20]. However, these methods suffer from storage and periodic update overheads caused by storing the precomputed aggregate results; thus, precomputation is limited only to the frequently asked queries. Therefore, we need dynamic methods that can compute aggregates on the fly even for the queries whose results have not been materialized.

Dynamic methods for computing aggregates in MOLAP have been limited to a few kinds of multidimensional file structures. Earlier methods use either multidimensional arrays [24] or compressed multidimensional arrays [14]. However, these structures have shortcomings. Multidimensional arrays are inadequate for data with a skewed distribution. Compressed multidimensional arrays degrade performance for non-aggregate OLAP operators such as range queries by destroying multidimensional clustering.

In this paper, we present a dynamic aggregation method using a multidimensional file that can maintain multidimensional clustering and that adapts to skewed data distributions. We first present an aggregation computation model that employs the new notion of the *disjoint-inclusive partition* for multidimensional files. We will formally define this notion in Section 4.1. We then present a one-pass dynamic aggregation algorithm based on this model. Our algorithm computes aggregations using a *one-pass buffer size*, which is the minimum buffer size required for guaranteeing one disk access per page. We formally derive the one-pass buffer size and prove that our aggregation algorithm is optimal with respect to the one-pass buffer size under our aggregation computation model. Consequently, any algorithm under our computation model, regardless of buffer replacement policies and page access orders, cannot have a one-pass buffer size smaller than our algorithm can have.

We implement the algorithm using the Multilevel Grid File (MLGF) [21, 23], which is a dynamic multidimensional file having a balanced tree structure. Through experiments, we demonstrate that the one-pass buffer size theoretically derived is indeed correct. We also show that performance of our algorithm is superior to those of other algorithms. We further show that the memory requirement of our algorithm is only a small fraction (0.05%∼0.6%) of the size of the database, making our algorithm practically usable.

The rest of the paper is organized as follows. Section 2 briefly reviews multidimensional files and buffer replacement policies. Section 3 presents the motivation for this research and describes a general method for computing aggregations using multidimensional files. Section 4 proposes our aggregation computation model. Section 5 presents the one-pass aggregation algorithm based on our model. Section 6 presents the results of the experiments. Finally, Section 7 concludes the paper.

2 Preliminaries

2.1 Multidimensional Files

We first define some terminology used for multidimensional files [21]. A *file* is a collection of *records*, where a record consists of a list of *attributes*. A subset of these attributes that determines the placement of the records in the file is called the *organizing attributes*. A file has a *multidimensional organization* if it contains more than one organizing attribute. A *domain* of an attribute is a set of values from which an attribute value can be drawn. We define the *domain space* as the Cartesian product of the domains of all the organizing attributes. We call any subset of the domain space a *region*. We call the region allocated to a page P a *page region* and denote it by $\tilde{P}$.

Multidimensional files have the multidimensional clustering property. The property enables efficient multi-attribute accesses, which retrieve qualified records using multiple attributes. The multidimensional clustering means that similar records, whose organizing attributes have similar values, are stored in the same page. To support the multidimensional clustering, multidimensional files partition the domain space into regions and store the records in each region on the same page. Thus, the directory represents the state of the domain space partition.

Multidimensional files can be classified into two categories according to the way the boundary value for splitting the region is determined [12]. One uses record-oriented splitting; the other uses region-oriented splitting. *Record-oriented splitting* divides the region into two subregions so that each subregion has the same number of records. Thus, the boundary value for splitting the region depends on the record distribution. *Region-oriented splitting* bisects the region regardless of the record distribution. Thus, the boundary values for splitting the region are predetermined independent of the record distribution.

In this paper, we perform experiments using the MLGF [21][23][12], a dynamic multidimensional file structure that uses region-oriented splitting and that adapts well to skewed distributions. The MLGF is a balanced tree consisting of a multilevel directory and data pages. A distinct characteristic of the MLGF is that it uses the *local splitting strategy*, which splits only the region where splitting is required rather than across the entire hyperplane. As a result, the directory growth is linearly dependent on the growth of the inserted records regardless of data distributions, data skew, or correlation among different organizing attributes [23]. This characteristic is shared by other multidimensional files such as the buddy tree, LSD tree, and k-d-B-tree, that use the local splitting strategy [7]. Thus, the MLGF gracefully adapts to highly skewed and correlated distributions that frequently occur in OLAP data.

2.2 Buffer Replacement Policies

A buffer replacement policy is the strategy used for choosing a page, called the *victim*, that will be removed from the buffer in order to make space available for a new page [6]. Typical replacement policies include LRU [5], CLOCK [6], and LRU-k [17]. A common strategy of replacement policies for minimizing the buffer fault rate is to select the page that has the longest expected time until the next access. LRU estimates the time of the next access to a page using the time of the last access to that page. LRU-k, a generalization of LRU, uses the time of the last k-th access. CLOCK is a simple and widely used approximation of LRU. These policies are effective when the page access order is not known in advance.

When the page access order is known in advance, we can increase the effectiveness of buffers by taking advantage of the order. Belady's B_0 [5] and the toss-immediate policies [9] are such examples. In order to use Belady's B_0 policy, we must know the complete page access order in advance. It selects as a victim the page that has the longest time till the next access. This policy has been proven to be the optimal replacement policy [5]. On the other hand, in order to use the toss-immediate policy, it is sufficient to know pages that will no longer be accessed at a given time. Upon each page access, it immediately invalidates the page that will not be used further. When all pages in the buffer are to be accessed further, the victim should be selected among themselves. In order to handle this situation, the toss-immediate policy is usually used together with a general replacement policy such as LRU or CLOCK. Since the page access order is not known *a priori* in general, these policies have been known to lack practicality despite its theoretical significance. Nevertheless, in this paper, we show that these policies can be effectively used for aggregation computation.

3 Computing Aggregates Using Multidimensional Files

In this section, we present our motivation for using multidimensional files in aggregation computation. Section 3.1 defines necessary terminology. Section 3.2 presents a general method for computing aggregates using multidimensional files. Section 3.3 discusses the necessity of buffers in aggregation computation.

3.1 Terminology

Aggregation is an operation that classifies records into groups according to the values of the specified attributes and determines one value per group by applying the given aggregate function [8]. An *aggregate* is the summarized value per group obtained through aggregation. We call the attributes used for grouping records the *grouping attributes*, and the attribute to which the aggregate function is applied the *aggregation attribute*. We define the *grouping domain space* as the Cartesian product of the domains of all the grouping attributes. We call any subset of the grouping domain space a *grouping region*. When we partition the grouping domain space into grouping regions to compute aggregation, we call them *aggregation windows*. We define a *partial aggregation* as aggregation for an aggregation window.

We define the *page grouping region* of a page P, denoted by $R = \prod_G \tilde{P}$, as the projection of the page region $\tilde{P}$ onto the grouping domain space consisting of a set G of grouping attributes. When a page region $\tilde{P}$ and a region Q overlap, we simply say that *the page P and the region Q overlap*. Likewise, when a page grouping region $\prod_G \tilde{P}$ and an aggregation window W overlap, we say that *the page P and the aggregation window W overlap*. We define *aggregation window pages* of an aggregation window W as the pages that overlap with W.

Example 1: Figure 1 shows an example of computing aggregation in a multidimensional file with three organizing attributes X, Y, and Z. The domain space has been divided into six regions with the records in each region being stored in pages A, B, C, D, E, and F. In the figure, the grouping attributes are X and Y, and the grouping domain space is X:[0,99]$\times Y$:[0,99]. An example of the grouping region is X:[0,49]$\times Y$:[0,49]. The four grouping regions forming a partition of the grouping domain space X:[0,49]$\times Y$:[0,49], X:[0,49]$\times Y$:[50,99], X:[50,99]$\times Y$:[0,49], and X:[50,99]$\times Y$:[50,99], represented by dashed lines in the figure, are the aggregation windows. Finally, A and E are the aggregation window pages of the aggregation window X:[0,49]$\times Y$:[0,49]. □

3.2 A General Method for Aggregation Computation

A simple method to compute aggregation is to use one aggregation window that is equal to the grouping domain space. We first create a result table consisting of entries <values of grouping attributes, aggregate value> in the main memory.

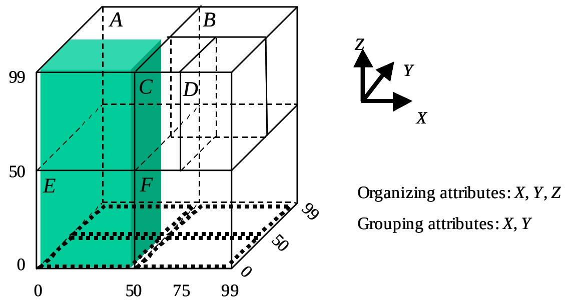


Figure 1. Computing aggregation in a multidimensional file.

We then scan the file and retrieve the records. For each record retrieved, using the values of the grouping attributes, we locate the corresponding entry from the result table, aggregate the value of the aggregation attribute, and store the result in the entry. If there is no corresponding entry, insert a new entry. This method can compute aggregates by scanning a file only once. However, this method would not be effective for a large volume of data because of limited availability of main memory.

An alternative method to compute aggregation is to use multiple aggregation windows that form a partition of the grouping domain space. The rationale behind this method is that computing aggregates for an aggregation window is independent of those for other aggregation windows. This is because records in different aggregation windows have different values for the grouping attributes forming different groups. If the size of an aggregation window is small, so is the size of the partial aggregation result. Thus, we can choose aggregation windows in such a way that the size of a partial aggregation result can fit in the result table. We call the result table to contain the result of an aggregation window a *window result table*.

The structure of a multidimensional file allows us to efficiently retrieve the records in an aggregation window. This is because multidimensional clustering renders range queries efficient. In contrast, in a relational database, it is not efficient to retrieve the records that belong to an aggregation window since the table structure does not support multidimensional clustering. Our algorithm exploits the clustering characteristic of the multidimensional file.

Figure 2 shows the algorithm, General_Aggregation, derived from the above principles. Step 1 partitions the grouping domain space into aggregation windows. Step 2 computes the partial aggregation for each aggregation window. Step 2.1 forms the range query for the partial aggregation. In the query, for the grouping attributes, the intervals correspond to the aggregation window; for other organizing attributes, the intervals correspond to the whole domain of each attribute. In Steps 2.2 and 2.2.1, the partial aggregation is computed using the records via the range query. The intermediate result for the partial aggregation is stored in the window result table residing in the main memory.

Example 2: We explain General_Aggregation uisng the example shown in Figure 1. First, the algorithm partitions the grouping domain space X:[0,99]$\times Y$:[0,99] into aggregation windows X:[0,49]$\times Y$:[0,49], X:[0,49]$\times Y$:[50,99], X:[50,99]$\times Y$:[0,49], and X:[50,99]$\times Y$:[50,99]. Then, the algorithm computes the partial aggregation for each aggregation window using the range query. For example, the range query for the aggregation window X:[0,49]$\times Y$:[0,49] is X:[0,49], Y:[0,49], and Z:[0,99], which is represented by the shaded bar in Figure 1. □

```
Algorithm General_Aggregation
Input: (1) Multidimensional file md-file that contains
           OLAP data
       (2) Set $G$ of grouping attributes
       (3) Aggregation attribute $A$
Output: Result of aggregation
Algorithm:
1 Partition the grouping domain space into aggregation
  windows.
2 For each aggregation window, DO
  2.1 Construct a range query. Here, the query region
      consists of the intervals corresponding to the ag-
      gregation window for the grouping attributes and
      the entire domain of each attribute for the other
      organizing attributes.
  2.2 Process the range query against md-file.
      2.2.1 For each record retrieved, using the values of
            the attributes in $G$, find the corresponding
            entry from the window result table, aggre-
            gate the value of the attribute $A$, and store
            the result into the entry.
```

Figure 2. The general aggregation algorithm General_Aggregation that uses a multidimensional file.

When we partition the grouping domain space in Step 1 of General_Aggregation, it is desirable to select aggregation windows so as to make the resulting sizes of partial aggregations similar to one another. This partition reduces the (maximum) size of the window result table that must reside in the main memory. We propose a method in Appendix A that selects aggregation windows using a histogram. In the remaining part of this paper, we assume that the aggregation windows are given, and focus on Step 2.

3.3 The Role of the Buffer in Aggregation Computation

Computing an aggregation with General_Aggregation may cause multiple disk accesses for the same pages. This is because, in Step 2.2 of the algorithm, a page in the multidimensional file is accessed once for each aggregation window that overlaps with the page. In multidimensional files, the sizes of page regions vary because they are determined by the data density in these regions. Hence, the pages that belong to a larger region are more frequently accessed since they tend to overlap with more aggregation windows.

In general, we use the buffer to reduce disk access. So can we for the General_Aggregation algorithm. In order to maximize the effectiveness of the buffer, a page should be accessed in such an order that it resides in the buffer until the next access. Thus, it is desirable to traverse the aggregation windows overlapping with a particular page contiguously. At the same time, a buffer replacement policy plays a critical role in effective use of the buffer. Buffer replacement policies such as LRU or CLOCK are typically used when we do not know the order of accessing pages *a priori.* A very interesting observation is that, when we compute aggregations using a multidimensional file, we have a way of computing the order of accessing pages *a priori.* We compute this order by using the relationships among aggregation windows and page grouping regions. Once we know the access order, we can obtain the theoretically optimal performance by using Belady's B_0 policy or the toss-immediate policy. We discuss these issues in Sections 4 and 5.

4 Aggregation Computation Model Based on Disjoint-Inclusive Partition of Multidimensional Files[†]

The page regions in multidimensional files have various shapes, and thus, the topological relationships among page regions and aggregation windows are complex. When these relationships have certain properties, we can improve the performance of computing aggregation by taking advantage of them. In this section, we first define the notions of the disjoint-inclusive relationship and the disjoint-inclusive partition. We then define our aggregation computation model that employs these notions for a multidimensional file. We then derive a lower bound of the one-pass buffer size under our aggregation computation model. We next discuss controlling the page access order such that pages to be accessed multiple times are accessed in contiguous partial aggregations.

4.1 Disjoint-Inclusive Partition (DIP) Multidimensional Files

Definition 1: Two regions S_1 and S_2 satisfy the *disjoint-inclusive* relationship if they satisfy Eq. (1).

$$S_1 \cap S_2 \neq \emptyset \Rightarrow (S_1 \supseteq S_2 \vee S_1 \subseteq S_2). \quad (1)$$

Definition 1 states that if two regions overlap, one includes the other. □

Definition 2: Let $\mathbb{D}$ be the domain space with the organizing attributes $A_1, A_2, \ldots, A_n$. A *disjoint-inclusive partition (DIP)* of $\mathbb{D}$ is a set of regions $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_k\}$ satisfying the following conditions:

(1) $(\bigcup_{i=1}^{k} Q_i = \mathbb{D}) \wedge (Q_i \cap Q_j = \emptyset, i \neq j)$.
(2) $(\forall G \subseteq \{A_1, A_2, \ldots, A_n\} \forall i, j (1 \leq i, j \leq k))$ ($\prod_G Q_i$ and $\prod_G Q_j$ satisfy the disjoint-inclusive relationship). □

We call a multidimensional file whose page regions form a DIP a *DIP multidimensional file.* Condition (1) of Definition 2 is a necessary and sufficient condition for $\mathcal{Q}$ to be a partition of $\mathbb{D}$. Condition (2) indicates that, when two regions in $\mathcal{Q}$ are projected onto any $\prod_G \mathbb{D}$ space, the projected regions must satisfy the disjoint-inclusive relationship.

Example 3: Figure 3 illustrates two examples of possible partitions of the domain space in a multidimensional file having three organizing attributes X, Y, and Z. Figure 3(a) satisfies Condition (1) of Definition 2 since the domain space has been partitioned into six regions $A \sim F$. Furthermore, the projected regions of any two regions in Figure 3(a) onto the space $\prod_G \mathbb{D}$, where $G = \{X, Y\}$, satisfy the disjoint-inclusive relationship. Since the disjoint-inclusive relationship also holds for any other combination of attributes for G, Figure 3(a) is a DIP. Figure 3(b) also satisfies Condition (1) of Definition 2. As shown in this figure, however, in the space $\prod_G \mathbb{D}$ where $G = \{X, Y\}$, $\prod_G A$ and $\prod_G D$(also $\prod_G A$ and $\prod_G E$) overlap without satisfying the disjoint-inclusive relationship. Therefore, Figure 3(b) is not a DIP. □

Lemma 1 presents a sufficient condition for a multidimensional file to have a DIP.

Lemma 1: If a multidimensional file satisfies the following splitting rules (1) and (2), the set of regions in the domain space resulting from the splits forms a DIP.

[†]Due to the VLDB policy, we prominently cite our earlier work in the following:

The notion of disjoint-inclusive partition in this section and the page access order (in Section 4.4) have appeared in an earlier paper [13]. The current paper significantly extends the preliminary work with the one-pass nature on the optimal buffer size (which is the main focus of the paper) newly introduced in Sections 4.3, 5, 6, and 7.

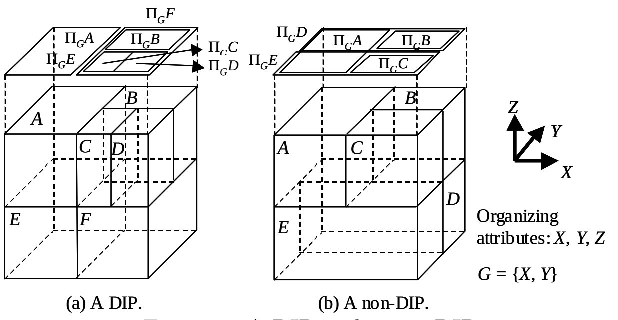


Figure 3. A DIP and a non-DIP.

(1) The multidimensional file uses region-oriented splitting.
(2) Let Q_i and Q_j be regions in the domain space and *SplitAxes(Q_i)* and *SplitAxes(Q_j)* be multisets of split axes used in obtaining Q_i and Q_j, respectively. Then, either $\mathrm{SplitAxes}(Q_i) \subseteq \mathrm{SplitAxes}(Q_j)$ or $\mathrm{SplitAxes}(Q_i) \supseteq \mathrm{SplitAxes}(Q_j)$.

Proof: See Appendix B. □

We identify a special case satisfying the splitting rule (2) of Lemma 1. Let *SplitAxesSeq(Q_i)* be a sequence of splitting axes used in obtaining Q_i. Then, a condition that $\mathrm{SplitAxesSeq}(Q_i)$ is a prefix of $\mathrm{SplitAxesSeq}(Q_j)$ or *vice versa* is a sufficient condition for the splitting rule (2). The cyclic splitting strategy [12], which selects the splitting axis cyclically, is a typical example that satisfies the condition. Example 4 shows a non-cyclic splitting strategy that satisfies the splitting rule (2).

Example 4: Figure 4 shows a multidimensional file having two organizing attributes X and Y. To achieve better performance in query processing, it is known to be desirable to make the shape of the page regions to have an interval ratio similar to that of query regions [12]. Figure 4 illustrates an example partition when the interval ratio of the page region $X : Y$ is $2 : 1$. To obtain such a partition, we use a splitting strategy that selects the Y axis as the splitting axis twice as frequently as the X axis. The sequences of the splitting axes for the page regions $A \sim E$ are as follows: $\mathrm{SplitAxesSeq}(A) = X$, $\mathrm{SplitAxesSeq}(B) = XY$, $\mathrm{SplitAxesSeq}(C) = XYY$, and $\mathrm{SplitAxesSeq}(D) = \mathrm{SplitAxesSeq}(E) = XYYX$. For any two regions, the sequence for a region is a prefix of that for the other. Thus, the partition shown in Figure 4 is a DIP. □

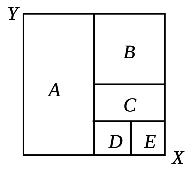


Figure 4. A non-cyclic splitting strategy that produces a DIP.

4.2 The Aggregation Computation Model

Definition 3: The *DIP computation model* for computing aggregations using a multidimensional file is the one that satisfies the following four conditions:

(1) A DIP multidimensional file is used.
(2) The aggregation with respect to the grouping domain space is computed as the union of partial aggregations, each of which is computed with respect to an aggregation window. Here, aggregation windows form a partition of the grouping domain space.
(3) Disjoint-inclusive relationship is satisfied among aggregation windows and page grouping regions.
(4) Each partial aggregation is computed by retrieving records through a range query against the multidimensional file. □

The DIP computation model computes aggregations using a DIP multidimensional file. Conditions (2) and (4) allow us to compute aggregates by taking advantage of multidimensional clustering. We have discussed such an aggregation method in Section 3.2. Conditions (1) and (3) allow us to use the DIP property as examplified . They provide the basis for developing an efficient aggregation algorithm and for analyzing the algorithm formally.

We note that it is always possible to partition the grouping domain space into aggregation windows that have the disjoint-inclusive relationship with page grouping regions as required by Condition (3). Page grouping regions of a DIP multidimensional file mutually satisfy the disjoint-inclusive relationship. Thus, if we select each aggregation window as a union of multiple page grouping regions, aggregation windows have the disjoint-inclusive relationship with the page grouping regions. This is examplified in the algorithm in Appendix A.

4.3 A Lower Bound on the One-Pass Buffer Size

In this section, we obtain a lower bound of the one-pass buffer size when computing aggregations under the DIP computation model. A lower bound is a buffer size that is at least required by any algorithm to guarantee one disk access per page. It is used in Section 5.1 to prove the optimality of our algorithm with respect to the one-pass buffer size.

Definition 4: Let $R = \prod_G \tilde{P}$ be the page grouping region of a page P. We say that P is an *L-page (a large page)* if R properly includes the least one aggregation window. We also say that, for an aggregation window W, P is an *L-page of* W if R properly includes W ($W \subset R$). □

Theorem 1: Consider computing aggregations under the DIP computation model. Let $\mathcal{W} = \{W_1, W_2, \ldots, W_k\}$ be a set of aggregation windows. Then, $\max_{W_i, 1 \leq i \leq k}\{(\text{the number of L-pages of } W_i) + \alpha_i\}$ is a lower bound of the one-pass buffer size. Here, α_i is 1 if at least one aggregation window page of W_i is a non-L-page and it is accessed after all the L-pages of W_i have been read into the buffer; α_i is 0 otherwise.

Proof: Intuitively, the buffer should contain as many pages as the number of L-pages. In addition, one page is needed to fetch a non-L-page into the buffer if it exists. See Appendix C for a detailed proof. □

The DIP computation model does not impose any constraint on buffer replacement policies and page access orders. Therefore, we can develop a variety of aggregation methods based on the DIP computation model by adopting different buffer replacement policies and page access orders. Theorem 1 states that the one-pass buffer size for any possible aggregation method cannot be less than the lower bound obtained.

4.4 Page Access Order

Our algorithm controls the page access order so that repeatedly accessed pages (L-pages) are accessed in contiguous partial aggregations. The objective is to let an L-page to be accessed all at once while it remains in the buffer so that it does not have to be accessed and loaded into the buffer any further. In Section 5.1 we prove that our algorithm, controlling the page access order in this way, has the optimal one-pass buffer size. To achieve this objective, we need to contiguously traverse the aggregation windows that overlap with those L-pages. To obtain such a traversal order, we use the notion of a space filling curve [7] induced from a given set of regions. Space filling curves have been used as the page access orders to increase the buffering effect in other applications (such as multidimensional spatial join [11]) as well. But, the selection of the space filling curve has been based on heuristic methods. In contrast, we use the optimal space filling curve selected based on the formal properties of the DIP multidimensional file.

Definition 5: For a given set $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ of regions in the multidimensional space $\mathbb{D}$, where elements of $\mathcal{S}$ satisfy the disjoint-inclusive relationship, we define the *induced space filling curve (ISFC)* as a space filling curve satisfying the following condition:

Condition [*contiguous interval*]: Let *the ISFC value*, *ISFC(p)*, is the value allocated to a point p in $\mathbb{D}$ by the ISFC. For any region S_i, points in S_i map to a contiguous interval of ISFC values. Formally, for any region S_i and any ISFC value v such that $\min_{p \in S_i}\{\text{ISFC}(p)\} \leq v \leq \max_{p \in S_i}\{\text{ISFC}(p)\}$, a point p satisfying $\text{ISFC}(p) = v$ must be in S_i and *vice versa.*

Here, we call $\mathcal{S}$ the *ISFC basis.* We define the *ISFC value of a region* S_i, *ISFC*(S_i), as $\max_{p \in S_i}\{\text{ISFC}(p)\}$. □

Lemma 2: For a given set $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ of regions in the multidimensional space $\mathbb{D}$, where elements of $\mathcal{S}$ satisfies the disjoint-inclusive relationship, there exists at least one ISFC with $\mathcal{S}$ as the ISFC basis.

Proof: See Appendix D. □

Definition 5 indicates that, in an ISFC order, all smaller regions included in a larger region S_i are traversed first, and then, those that are not included in S_i are traversed. We take advantage of of the notion of the ISFC in our aggregation algorithm: when there are smaller aggregation windows overlapping with a larger page grouping region for a page P, we use an ISFC order to make those aggregation windows traversed contiguously, so that the page P may reside in the buffer without swapping.

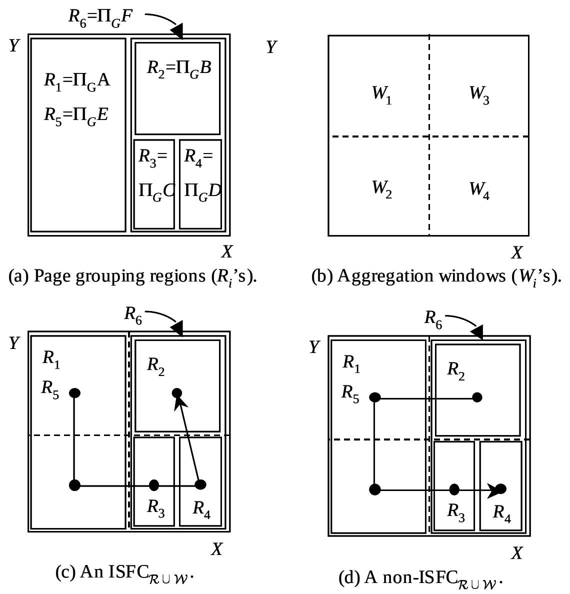
We now present a method that uses an ISFC as the traversal order of aggregation windows and discuss its characteristics. Let $\mathcal{R}$ be a set of page grouping regions in a DIP multidimensional file and $\mathcal{W}$ a set of aggregation windows, where elements of $\mathcal{R}$ and $\mathcal{W}$ satisfy the disjoint-inclusive relationship. We define the *ISFC*$_{\mathcal{R}\cup\mathcal{W}}$ as the ISFC in the grouping domain space using $\mathcal{R} \cup \mathcal{W}$ as the ISFC basis.

Example 5: Figure 5 illustrates examples of an ISFC$_{\mathcal{R}\cup\mathcal{W}}$ and a non-ISFC$_{\mathcal{R}\cup\mathcal{W}}$. Figures 5(a) and 5(b) show page grouping regions (R_i's) and aggregation windows (W_i's), respectively, used in Example 1. Figures 5(c) and 5(d) show the overlay of page grouping regions shown in Figure 5(a) on top of aggregation windows shown in Figure 5(b). Figure 5(c) satisfies the definition of the ISFC$_{\mathcal{R}\cup\mathcal{W}}$: the order in which the regions are traversed is $R_1(W_1 \rightarrow W_2) \rightarrow R_6(W_4(R_3 \rightarrow R_4) \rightarrow W_3(R_2))$. Figure 5(d) does not, however, because points in region R_1 are traversed while those in region R_6 still are. □

If we use the ISFC$_{\mathcal{R}\cup\mathcal{W}}$ as the order of traversing aggregation windows, the algorithm has a characteristic described in the following Lemma 3. Lemma 3 is used to prove Lemma 4.

Lemma 3: Consider computing aggregations under the DIP computation model. Let $\mathcal{W} = < W_1, W_2, \ldots, W_k >$ be a list of aggregation windows, where W_i's are ordered in an ISFC$_{\mathcal{R}\cup\mathcal{W}}$ order, i.e., if $i < j$, then $\text{ISFC}_{\mathcal{R}\cup\mathcal{W}}(W_i) < \text{ISFC}_{\mathcal{R}\cup\mathcal{W}}(W_j)$. Then, the aggregation windows that overlap with an L-page are contiguous, i.e., they are $W_l, W_{l+1}, \ldots, W_h (1 \leq l \leq h \leq k)$.

Proof: See Appendix E. □



R_6=$\Pi_G F$
Y
X
R_1=Π_GA
R_5=$\Pi_G E$
R_2=$\Pi_G B$
R_3= $\Pi_G C$
R_4= $\Pi_G D$
(a) Page grouping regions (R_i's).
W_1
W_2
W_3
W_4
(b) Aggregation windows (W_i's).
R_1
R_2
R_3
R_4
R_5
R_6
(c) An ISFC$_{\mathcal{R}\cup\mathcal{W}}$.
(d) A non-ISFC$_{\mathcal{R}\cup\mathcal{W}}$.

Figure 5. An ISFC$_{\mathcal{R}\cup\mathcal{W}}$ and a non-ISFC$_{\mathcal{R}\cup\mathcal{W}}$ ($G = \{\text{X}, \text{Y}\}$).

By Lemma 3, if we compute aggregations traversing aggregation windows in the ISFC$_{\mathcal{R}\cup\mathcal{W}}$ order, the aggregation windows that overlap with an L-page are processed contiguously. In other words, repeatedly accessed pages are accessed in contiguous aggregation windows.

In an actual implementation, a specific ISFC$_{\mathcal{R}\cup\mathcal{W}}$ is determined by the intrinsic characteristics of a multidimensional file. We present a way of determining an ISFC$_{\mathcal{R}\cup\mathcal{W}}$ order based on the splitting rules of Lemma 1. All page grouping regions in $\mathcal{R}$ have been created according to the splitting rules of Lemma 1. In addition, since the set of aggregation windows $\mathcal{W}$ must satisfy the disjoint-inclusive relationship with the page grouping regions, we select the aggregation windows from the regions that can be created according to the splitting rules. These regions are the ones resulting from splitting the domain space recursively. Under these circumstances, we can use as the ISFC$_{\mathcal{R}\cup\mathcal{W}}$ a space filling curve having the recursive property that accords with the order of splitting axes selected. For example, when a multidimensional file uses the cyclic splitting strategy, we can use Z-order [7] as the ISFC$_{\mathcal{R}\cup\mathcal{W}}$. We use Z-order in our experiments.

5 One-Pass Aggregation Algorithm

In this section we present the one-pass aggregation algorithm based on the DIP computation model and prove that the algorithm is optimal with respect to the one-pass buffer size under the model. Section 5.1 derives the one-pass buffer size when using the toss-immediate policy as the buffer replacement policy.[3] Section 5.2 presents the one-pass aggregation algorithm.

[3]With the DIP computation model, we can compute the complete page access order in advance so that we can use Belady's B_0 policy, which is more efficient than the toss-immediate policy. But, since computing the complete access order is complicated and is not the focus of this paper, we present this aspect in a future paper. The toss-immediate policy can be easily used because it only requires identifying pages that will no longer be accessed, i.e., it requires only partial information on the page access order. Both policies have the same one-pass buffer size. For the sizes smaller than the one-pass buffer size, Belady's B_0 policy has better performance than the toss-immediate policy.

5.1 One-Pass Buffer Size when Using the Toss-Immediate Policy

To use the toss-immediate policy, we need to know which pages will no longer be accessed at a given time. We can identify such pages using the following Lemma 4.

Lemma 4: Consider computing aggregations under the DIP computation model. Let $\mathcal{W} = \{W_1, W_2, \ldots, W_k\}$ be a set of aggregation windows. Suppose we compute partial aggregations in an $\mathrm{ISFC}_{\mathcal{R}\cup\mathcal{W}}$ order. Then, a page P_{curr} accessed during the partial aggregation for W_{curr} will no longer be accessed if $\mathrm{ISFC}_{\mathcal{R}\cup\mathcal{W}}(\prod_G \tilde{P}_{curr}) \leq \mathrm{ISFC}_{\mathcal{R}\cup\mathcal{W}}(W_{curr})$.
Proof: See Appendix F. □

In the following Theorem 2, we derive the one-pass buffer size when using the toss-immediate policy as the buffer replacement policy.

Theorem 2: Consider computing aggregations under the DIP computation model. Let $\mathcal{W} = \{W_1, W_2, \ldots, W_k\}$ be a set of aggregation windows. Suppose we compute partial aggregations in an $\mathrm{ISFC}_{\mathcal{R}\cup\mathcal{W}}$ order and use toss-immediate as the buffer replacement policy. Then, the one-pass buffer size is equal to the lower bound of the one-pass buffer size ($\max_{W_i, 1\leq i\leq k}\{$(the number of L-pages of W_i)$+\alpha_i\}$) obtained in Theorem 1. Thus, this size is the minimum.
Proof: See Appendix G. □

5.2 Aggregation Algorithm Using One-Pass Buffer Size

Figure 6 shows the algorithm One_Pass_Aggregation that extends the algorithm General_Aggregation by using the $\mathrm{ISFC}_{\mathcal{R}\cup\mathcal{W}}$ order and the one-pass buffer. Step 1 partitions the grouping domain space into aggregation windows satisfying the disjoint-inclusive relationship with page grouping regions. Step 2.1 computes the one-pass buffer size, BUFSIZE, derived in Theorem 2. A multidimensional file contains information about the regions in its directory. Accordingly, we can compute BUFSIZE by just reading its directory.[4] Step 2.2 allocates the memory of BUFSIZE. Step 3.1 constructs the range query for the partial aggregation for an aggregation window W_{curr}. Step 3.2 evaluates the range query, and checks whether the current page P_{curr} will no longer be used when its processing has been completed. If the $\mathrm{ISFC}_{\mathcal{R}\cup\mathcal{W}}$ value of the page grouping region of P_{curr} is smaller than or equal to that of W_{curr}, it will no longer be accessed by Lemma 4, and the page is removed from the buffer according to the toss-immediate policy. Step 3.3 computes aggregates using the records retrieved from the range query and stores them in the window result table. By Theorem 2, the algorithm One_Pass_Aggregation has the one-pass buffer size that is minimum among all the algorithms based on the DIP computation model.

6 Performance Evaluation

In this section we present the result of the performance evaluation for the One_Pass_Aggregation algorithm. The objectives of the experiments are three fold. First, we validate

[4]In a real implementation, we do not need to read the directory or compute BUFSIZE. In Step 2 we do not allocate the buffer. Instead, in Step 3.2, we dynamically allocate a buffer page for a newly accessed page when there is none available. This implementation has the same one-pass buffer size. In addition, this implementation improves the performance in a multiuser environment because of dynamic (incremental) allocation of the buffer. However, to help the reader understand the paper more easily, we have described the algorithm including the step that computes the one-pass buffer size.

```
Algorithm One_Pass_Aggregation
Input: (1) DIP multidimensional file md-file that
           contains OLAP data
       (2) Set $G$ of grouping attributes
       (3) Aggregation attribute $A$
Output: Result of aggregation
Algorithm:
1 Partition the grouping domain space into aggregation
  windows so that aggregation windows and page grouping
  regions satisfy the disjoint-inclusive relationship.
2 Initialize the buffer:
  2.1 Compute the one-pass buffer size, BUFSIZE=
      $\max_{W_i,1\leq i\leq k}\{$(the number of L-pages of $W_i$)$+\alpha_i\}$.
  2.2 Allocate the buffer of size BUFSIZE.
3 Traversing aggregation windows in an $\mathrm{ISFC}_{\mathcal{R}\cup\mathcal{W}}$ order,
  for each aggregation window $W_{curr}$, DO
  3.1 Construct a range query. Here the query region
      consists of the intervals corresponding to the aggre-
      gation window for the grouping attributes and the
      entire domain of each attribute for the other orga-
      nizing attributes.
  3.2 Process the range query against md-file.
      3.2.1 While evaluating the range query, when pro-
            cessing the current page $P_{curr}$ is completed,
            if $\mathrm{ISFC}_{\mathcal{R}\cup\mathcal{W}}(\prod_G \tilde{P}_{curr}) \leq \mathrm{ISFC}_{\mathcal{R}\cup\mathcal{W}}(W_{curr})$,
            then remove $P_{curr}$ from the buffer.
      3.2.2 For each record retrieved, using the values of
            the attributes in $G$, find the corresponding
            entry from the window result table, aggregate
            the value of the attribute $A$, and store the
            result into the entry.
```

Figure 6. The One_Pass_Aggregation Algorithm.

Theorem 2 regarding the size of the one-pass buffer. Second, we show the performance of the One_Pass_Aggregation algorithm is superior compared with those of other algorithms described in Section 6.1. Third, we show that our algorithm requires a relatively small memory (0.05%∼0.6% of the size of the database) for processing the aggregation in one pass. Section 6.1 explains the experimental environment and data sets. Section 6.2 presents the experimental results.

6.1 The Experimental Environment and Data Sets

Data Sets: We use both synthetic and real data sets for the experiments. The synthetic data sets have records consisting of six attributes: five are organizing attributes representing the dimensions, and one is the measure. The data types of all the attributes are 4-byte integers, whose domain is $[-2^{31}, 2^{31}-1]$. To simulate the OLAP data where records are distributed in many clusters, we use a distribution that superposes 100 overlapping multivariate normal distributions for data distributions of the organizing attributes. The i-th attribute of the multivariate normal distribution has a normal distribution of $N(\mu_i, \sigma^2)$, where the mean μ_i is randomly selected within the domain $[-2^{31}, 2^{31}-1]$, and the standard deviation σ is varied from 2^{20} to 2^{29} ($\frac{1}{2^{12}} \sim \frac{1}{2^3}$ of the domain). For sensitivity analysis, we generate 5,000, 50,000, and 500,000 records for each distribution. Thus, a total of 500,000 (29.9MB = 7,667 pages), 5,000,000 (279.3MB = 71,492 pages), and 50,000,000 (2,648.3MB = 677,964 pages) records are in each data set. We call them *SMALL-DATA*, *MEDIUM-DATA*, and *LARGE-DATA*, respectively. The page size used is 4KB.

The real data set that we use is the Forest Cover Type database from the UCI KDD archive [2]. It has about 581,012 records consisting of 54 attributes, ten of which are numerical. We use five-dimensional projection of the data set (the projected data set has 580,616 records) and use these five attributes as the dimensions and organizing attributes. These attributes are Elevation, Aspect, Slope, Horizontal_distance_to_hydrology, and Vertical_distance_to_hydrology. We use a dummy attribute as the measure. We call this data set *REAL-DATA*. The size of REAL-DATA is 34.3MB (8,784 pages).

Methods of Experiments: We have performed extensive experiments using the data sets. We have used three grouping attributes among the five organizing attributes. As the DIP multidimensional file storing the OLAP data, we have used the MLGF.[5]

We have performed five experiments varying the following parameters: the type of data sets, data set size, window result table size, and buffer size. Experiments A and B are for the first and second objectives. We use the number of disk accesses that occur during aggregation computation as the performance measure. The reason is two fold: 1) disk I/O has a major effect on the performance of aggregation; 2) we intend to demonstrate by experiment the correctness of the one-pass buffer size derived in Theorem 2. Experiments C, D, and E are for the third objective. In these experiments, we measure the memory requirement of the One_Pass_Aggregation algorithm. Here, the memory requirement is the sum of the window result table size and the one-pass buffer size.

1) **Experiment A:** MEDIUM-DATA with σ of 1.25×2^{28} is used. The effect of the buffer size is analyzed as it is varied from five to well beyond the one-pass buffer size, while the window result table size is set to 0.05% of the database size. The reason for using σ of 1.25×2^{28} and the window result table size of 0.05% will be explained in Section 6.2.
2) **Experiment B:** Experiment A is repeated using REAL-DATA instead of MEDIUM-DATA.
3) **Experiment C:** MEDIUM-DATA is used by varying σ from 2^{20} to 2^{29}. The memory requirement is measured as σ is varied, while the window result table size is set to 0.05% of the database size.
4) **Experiment D:** The memory requirement is measured as the window result table size is varied. For sensitivity analysis experiments are repeated for SMALL, MEDIUM, and LARGE-DATA with σ of 1.25×2^{28}.
5) **Experiment E:** Experiment D is repeated for REAL-DATA.

Algorithms Compared: We compare the performances of the following four aggregation algorithms.

1) **Naive_Aggregation** This algorithm is a straightforward one derived from the algorithm General_Aggregation in Section 3.2. It obtains aggregation windows using the partitioning algorithm of the equi-depth histogram [15] and traverses the aggregation windows using the row-major order. Here, the aggregation windows do not satisfy the disjoint-inclusive relationship with page grouping regions. Thus, Naive_Aggregation does not conform to the DIP computation model. In addition, as the buffer replacement policy, Naive_Aggregation uses the CLOCK policy.
2) **DIP_Aggregation** This algorithm uses the same aggregation windows as those of the One_Pass_Aggregation algorithm. But, it traverses the aggregation windows using the Hilbert order [7]. DIP_Aggregation uses the CLOCK policy. We use DIP_Aggregation to analyze the performance gain obtained by conforming to the DIP computation model, where we use the aggregation windows that satisfy the disjoint-inclusive relationship with page grouping regions.
3) **ISFC_Aggregation** This algorithm uses the same aggregation windows and the same traversal order as those of the One_Pass_Aggregation algorithm. In Step 3.2.1 of the One_Pass_Aggregation algorithm, however, it uses CLOCK instead of the toss-immediate as the buffer replacement policy. We use ISFC_Aggregation to analyze the performance gain obtained by conforming to the DIP computation model and by using the ISFC$_{\mathcal{R} \cup \mathcal{W}}$ order. But, here, the effect of using the toss-immediate policy is excluded.
4) **One_Pass_Aggregation** This algorithm is the one proposed in Section 5.2 using the toss-immediate policy. However, when the buffer is smaller than the one-pass buffer, it is not feasible to use the toss-immediate policy only. Thus, for such buffers, we use the toss-immediate+CLOCK policy (which we simply call *toss-immediate*) instead. That is, if there is no page remaining to remove in Step 3.2.1 of the One_Pass_Aggregation algorithm, we use the CLOCK policy to select the victim.

6.2 Experimental Results

Experiment A: Figure 7 shows the results of Experiment A. The horizontal axis represents the size of the buffer. The vertical axis represents the *normalized I/O access*, which is defined as the number of page accesses normalized by the total number of pages in the file. The normalized I/O access of 1.0 represents the theoretically optimal performance, and the buffer size achieving it is the one-pass buffer size. The experiment shows the one-pass buffer size for One_Pass_Aggregation is 94, which turns out to be equal to $\max_{W_i, 1 \le i \le k}\{(\text{the number of L-pages of } W_i) + \alpha_i\}$[6] as predicted in Theorem 2.

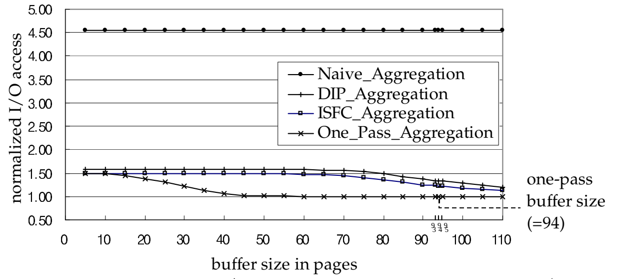


Figure 7. Normalized I/O access for MEDIUM-DATA (71,492 pages) with $\sigma = 1.25 \times 2^{28}$, where the window result table size is 36 pages.

Figure 7 shows that DIP_Aggregation has a far better performance than Naive_Aggregation. The result verifies the effectiveness of our approach using the DIP computation model. Figure 7 also shows that ISFC_Aggregation is better than DIP_Aggregation over the entire range of the buffer size. This is a natural result because the ISFC$_{\mathcal{R} \cup \mathcal{W}}$ order, which is derived from the characteristic of the DIP multidi-

[5]We use the MLGF for the experiments here, but our methodology is applicable to other kinds of DIP multidimensional files. Multidimensionl files that obey the two rules in Lemma 1 can be DIP multidimensional files. The buddy tree [19], the quad tree [18], and the grid file [16] are examples. On the other hand, some cannot be made DIP multidimensional files. The R*-tree [3] is an example.

[6]The number of L-pages has been counted by reading the directory for each aggregation window to calculate the predicted value.

mensional file, is the optimal order for traversing the aggregation windows. We note that the performance gap between ISFC_Aggregation and DIP_Aggregation is not large in this experiment because the Hilbert order is similar in buffering effect to the ISFC$_{\mathcal{R}\cup\mathcal{W}}$ order of this experiment, which is the Z-order.[7] Figure 7 also shows that One_Pass_Aggregation is better than ISFC_Aggregation over the entire range of the buffer size. This result verifies that the replacement policy that selects a victim using the page access order known *a priori* is very effective. For ISFC_Aggregation the number of disk accesses changes slowly compared with that of One_Pass_Aggregation as the buffer size increases. The reason is that the buffer does not help since the numbers of aggregation window pages are greater than the buffer size. In this case, the aggregation window pages of one aggregation window are all replaced with those of the next aggregation window to be accessed neutralizing the effect of the buffer. In contrast, for One_Pass_Aggregation, the number of disk accesses decreases steadily until the buffer size reaches the one-pass buffer size since the buffer is much more effectively used due to the toss-immediate policy.

Experiment B: Figure 8 shows the results of Experiment B using REAL-DATA. Similar to Figure 7, Figure 8 indicates that One_Pass_Aggregation is the best, ISFC_Aggregation the next, DIP_Aggregation the next, and Naive_Aggregation the worst over the entire range of the buffer size. The phenomenon is more marked here since there are more random variation of data distribution in real data. The experiment shows that the one-pass buffer size for One_Pass_Aggregation is 49, which turns out to be equal to the prediction in Theorem 2. In Figure 8 we observe that One_Pass_Aggregation reduces the number of disk accesses by up to 7.31 times compared with Naive_Aggregation and up to 1.93 times compared with ISFC_Aggregation.

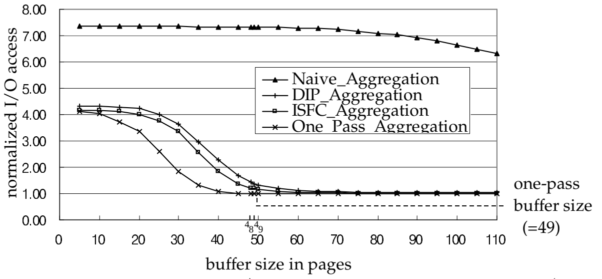
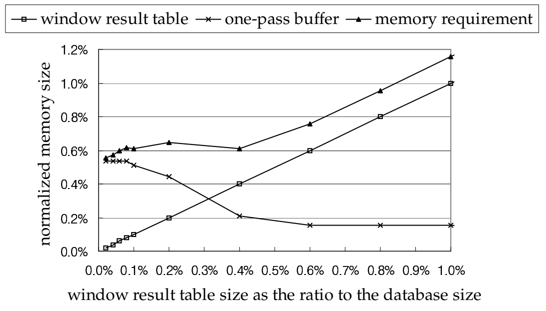


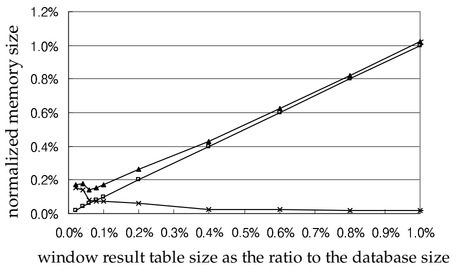
Figure 8. Normalized I/O access for REAL-DATA (8,784 pages), where the window result table size is 4.4 pages.

Experiment C: In this experiment, we have measured the memory requirement of the One_Pass_Aggregation algorithm varying σ from 2^{20} to 2^{29}. The results show that the memory requirement is the largest when σ is 1.25×2^{28}; thus, we use this value as the worst case one for σ in Experiments A and E.

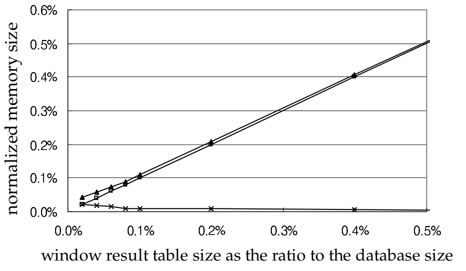
Experiment D: Figure 9 shows the results of Experiment D. Figure 9(a) shows the memory requirement of the algorithm for SMALL-DATA; Figure 9(b) for MEDIUM-DATA; and 9(c) for LARGE-DATA. The horizontal axis represents the window result table size; the vertical one the memory requirement normalized by the database size. For all three data sets, the one-pass buffer size decreases as the window result table size increases. The reason is that, as the window result table size increases, we can use larger aggregation windows, and the number of L-pages decreases. We observe that, for all three data sets, the memory requirement of the One_Pass_Aggregation algorithm for processing the aggregation in one pass is approximately 0.05%~0.60% of the size of the database and becomes smaller for larger databases. For the case of MEDIUM-DATA in Figure 9(b), the memory requirement is minimized when the window result table size is less than 0.1% of the database size. We can use this range for practical purposes. Thus, we have used the middle value 0.05% of this range for the window result table size in Experiments A~C.



(a) SMALL-DATA.



(b) MEDIUM-DATA.



(c) LARGE-DATA.

Figure 9. Memory requirement for SMALL, MEDIUM, and LARGE-DATA with $\sigma = 1.25 \times 2^{28}$.

Experiment E: Figure 10 shows the results of Experiment E. We can see that the trend is similar to that of SMALL-DATA (of approximately the same size) in Figure 9(a).

7 Conclusions

Efficient aggregation algorithms are crucial for achieving good performance in OLAP systems. In this paper, we have presented a dynamic aggregation algorithm that uses multidimensional files in MOLAP. We have presented the new notion of the disjoint-inclusive partition and proposed the aggregation computation model, called the DIP computation model, using this notion. Based on the model, we have proposed the aggregation algorithm, One_Pass_Aggregation, that computes aggregation using the one-pass buffer size. Our algorithm achieves the optimal one-pass buffer size by using a buffer

[7]Changing the split strategy, which changes the ISFC$_{\mathcal{R}\cup\mathcal{W}}$ order, would produce a larger gap. Experiments are currently being done on this effect.

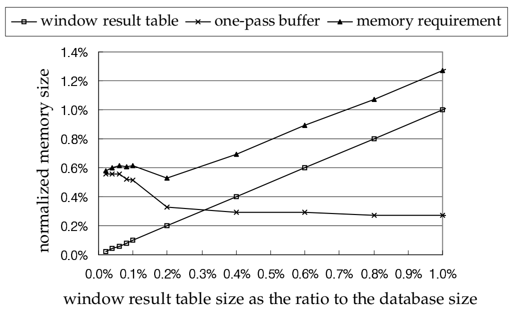


Figure 10. Memory requirement for REAL-DATA.

replacement policy, such as Belady's B_0 or Toss-Immediate policies, that exploits the page access order computed in advance. Since the page access order is not known *a priori* in general, these policies have been known to lack practicality despite its theoretic significance. Nevertheless, in this paper, we have shown that these policies can be effectively used for aggregation computation.

We also have proposed a formal framework for computing aggregation under the DIP computation model. First, in Theorem 1, we have formally derived a lower bound of the one-pass buffer size. The one-pass buffer size is the minimum buffer size required for guaranteeing one disk access per page. Then, in Lemma 3, we have proved that we can maximize the buffering effect by controlling the order of accessing pages with the ISFC to process repeatedly accessed pages (L-pages) in consecutive aggregation windows. Next, in Lemma 4, we have proved that we can identify the pages that are no longer to be accessed, enabling the use of the toss-immediate policy. Then, in Theorem 2, we have proved that the one-pass buffer size becomes minimum if we use the toss-immediate policy. Finally, we have presented the One_Pass_Aggregation algorithm that uses the one-pass buffer.

To verify the performance of One_Pass_Aggregation, we have performed extensive experiments with data sets having various distributions. The experimental results show that the one-pass buffer size of the algorithm predicted in Theorem 2 is indeed correct and that One_Pass_Aggregation using the toss-immediate policy always has better performance than ISFC_Aggregation using the conventional CLOCK policy, which is a widely used approximation of LRU. Moreover, experimental results for a real data set show that our algorithm reduces the number of disk accesses by up to 7.31 times compared with Naive_Aggregation. They also show that the memory requirement of One_Pass_Aggregation for processing the aggregation in one pass is a very small fraction (0.05%$\sim$0.6%) of the size of the database. These results indicate that our algorithm is practically usable even for a fairly large database.

Our algorithm is effective especially in a multiuser environment, where many aggregation queries are requested concurrently because it uses only those buffer pages that are essential to maintain the normalized I/O access of 1.0. We believe that our work is commercially implementable and, at the same time, provides an excellent formal basis for investigating further issues in computing aggregations in MOLAP.

Acknowledgements

This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

References

[1] Agarwal, S., Agrawal, R., Deshpande, P.M. et al., "On the Computation of Multidimensional Aggregations," In *Proc. Int'l Conf. on Very Large Data Bases*, pp. 506–521, Mumbai(Bombay), India, 1996.

[2] Bay, S. D., The UCI KDD Archive, University of California, Department of Information and Computer Science, Irvine, CA, 1999 (available at URL: http://kdd.ics.uci.edu/).

[3] Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B., "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," In *Proc. Int'l Conf. on Management of Data*, pp. 322–331, ACM SIGMOD, Atlantic City, NJ, 1990.

[4] Chaudhuri, S. and Dayal, U., "An Overview of Data Warehousing and OLAP Technology," *ACM SIGMOD Record*, Vol. 26, No. 1, pp. 65–74, Mar. 1997.

[5] Coffman, E.G. Jr. and Denning, P.J., *Operating Systems Theorey*, Prentice-Hall, 1973.

[6] Effelsberg, W. and Haerder, T., "Principles of Database Buffer Management," *ACM Trans. on Database Systems*, Vol. 9, No. 4, pp. 560–595, Dec. 1984.

[7] Gaede, V. and Günther, O., "Multidimensional Access Methods," *ACM Computing Surveys*, Vol. 30, No. 2, pp. 170–231, June 1998.

[8] Graefe, G., "Query Evaluation Techniques for Large Databases," *ACM Computing Surveys*, Vol. 25, No. 2, pp. 73–170, June 1993.

[9] Korth, H.F. and Silberschatz, A., *Database System Concepts*, McGraw-Hill, New York, Second Ed., 1991.

[10] Kotidis, Y. and Roussopoulos, N., "An Alternative Storage Organization for ROLAP Aggregate Views Based on Cubetrees," In *Proc. Int'l Conf. on Management of Data*, pp. 249–258, ACM SIGMOD, Seattle, Washington, 1998.

[11] Koudas, N. and Sevcik, K.C., "High Dimensional Similarity Joins: Algorithms and Performance Evaluation," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 12, No. 1, pp. 3–18, Jan./Feb. 2000.

[12] Lee, J., Lee, Y., and Whang, K., "Region Splitting Strategy for Physical Database Design of Multidimensional File Organizations," In *Proc. Int'l Conf. on Very Large Data Bases*, pp. 416–425, Athens, Greece, Aug. 1997.

[13] Lee, Y., Whang, K., Moon, Y., and Song, I., "An Aggregation Algorithm Using a Multidimensional File in Multidimensional OLAP," *Information Sciences* (accepted to appear), 2001.

[14] Li, J., Rotem, D., and Srivastava, J., "Aggregation Algorithms for Very Large Compressed Data Warehouses," In *Proc. Int'l Conf. on Very Large Data Bases*, pp. 651–662, Edinburgh, Scotland, UK, 1999.

[15] Muralikrishna, M. and DeWitt, D., "Equi-Depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries," In *Proc. Int'l Conf. on Management of Data*, pp. 28–36, ACM SIGMOD, Chicago, Illinois, June 1988.

[16] Nievergelt, J., Hinterberger, H., and Sevcik, K.C., "The Grid File: An Adaptable, Symmetric Multikey File Structure," *ACM Trans. on Database Systems*, Vol. 9, No. 1, pp. 38–71, Mar. 1984.

[17] O'Neil, E.J., O'Neil, P.E., and Weikum, G., "The LRU-K Page Replacement Algorithm for Database Disk Buffering," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, Washington, DC, May 1993.

[18] Samet, H., "The Quadtree and Related Hierarchical Data Structures," *ACM Computing Surveys*, Vol. 16, No. 2, pp. 187–260, June 1984.

[19] Seeger, B. and Kriegel, H.-P., "The Buddy-Tree: An Efficient and Robust Access Method for Spatial Data Base Systems," In *Proc. 6th Int'l Conf. on Very Large Data Bases*, pp. 590–601, 1990.

[20] Vassiliadis, P. and Sellis, T., "A Survey of Logical Models for OLAP Databases," *ACM SIGMOD Record*, Vol. 28, No. 4, pp. 64–69, 1999.

[21] Whang, K. and Krishnamurthy, R., Multilevel Grid Files, IBM Research Report RC 11516(51719), 1985.

[22] Whang, K., Vander-Zanden, B.T., and Taylor, H.M., "A Linear-time Probabilistic Counting Algorithm for Database Applications," *ACM Trans. on Database Systems*, Vol. 15, No. 2, pp. 208–229, June 1990.

[23] Whang, K. et al., "Dynamic Maintenance of Data Distribution for Selectivity Estimation," *The VLDB Journal*, Vol. 3, No. 1, pp. 29–51, Jan. 1994.

[24] Zhao, Y., Deshpande, P.M., and Naughton, J.F., "An Array-Based Algorithm for Simultaneous Multidimensional Aggregates," In *Proc. Int'l Conf. on Management of Data*, pp. 159–170, ACM SIGMOD, Tucson, Arizona, 1997.

Appendix A: A Method for Selecting Aggregation Windows

To select aggregation windows, we need to partition the grouping domain space into aggregation windows so as to make the sizes of the results of partial aggregations similar to that of the result table. To do that, we need to know the number of records with different values for the grouping attributes in an aggregation window. For this purpose, we use multidimensional histograms for the grouping attributes. Since histograms are used for processing various queries as well as aggregation, we assume that histograms [23] are maintained together with the OLAP database. However, maintaining histograms for all the combinations of the grouping attributes would incur much overhead. To solve this problem, we use only one histogram for all the organizing attributes and use the total number of records in a grouping region as an upper bound of the number of records having unique values for the grouping attributes.[8]

Figure 11 shows the algorithm SelectAggregationWindows that selects aggregation windows using the histogram. The inputs are a histogram (H) for the organizing attributes, projection of the sequence of splitting axes onto the grouping attributes (*SplitSequence*=$A_1A_2A_3\ldots$), the result table size in the number of records (R_{max}), and a grouping region to be partitioned into aggregation windows (S). The output of the algorithm SelectAggregationWindows is the set of aggregation windows that partition the domain space. Step 1 computes no_records(S) by projecting the histogram H onto the grouping attributes. Here, no_records(S) is an upper bound for the number of records having different values for the grouping attributes in S. Step 2 selects S as the aggregation window if no_records(S) is equal to or less than R_{max}. If no_records(S) is greater than R_{max}, Step 3 splits S into S_1 and S_2 using the axis A_1 as the splitting axis so that the result table may be able to accommodate the result of aggregation over S. Step 4 calls SelectAggregationWindows recursively using S_1 for S and $(A_2A_3\ldots)$ for *SplitSequence*. Likewise, Step 5 calls SelectAggregationWindows recursively using W_2 and $(A_2A_3\ldots)$.

[8]This estimate is an upper bound of the number of records having different values for the grouping attributes. To obtain a more accurate estimate, we need to revise the estimate using the duplication factor [22] which we maintain as a statistic. However, we do not discuss the details since it is beyond the scope of the current paper.

```
Algorithm SelectAggregationWindows
Input: (1) a histogram $H$ for the organizing attributes
       (2) projection of the sequence of the splitting axes
           onto the grouping attributes SplitSequence=
           $A_1A_2A_3\ldots$
       (3) the result table size $R_{max}$
       (4) a grouping region $S$ to be partitioned
Output: a set of aggregation windows partitioning $S$
Algorithm:
1 Get no_records($S$) using the histogram $H$.
2 IF no_records($S$) $\le R_{max}$ THEN select $S$ as the aggre-
    gation window and return.
3 Split $S$ into $S_1$ and $S_2$ using axis $A_1$ as the splitting
    axis.
4 Call SelectAggregationWindows using $S_1$ for $S$ and
    $(A_2A_3\ldots)$ for SplitSequence.
5 Call SelectAggregationWindows using $S_2$ for $S$ and
    $(A_2A_3\ldots)$ for SplitSequence.
```

Figure 11. The SelectAggregationWindows Algorithm.

Appendix B: Proof of Lemma 1

Let $A_1, \ldots, A_n$ be the organizing attributes of the multidimensional file and $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ the set of regions in the domain space resulting from the splits. Then, for any two regions Q_i and Q_j, the following holds:
(1) Since region-oriented splitting bisects the region upon split, the projections of Q_i's onto the domain of an attribute A_l must be one of those intervals that can be obtained by bisecting the domain of A_l recursively. Thus, the projections $\prod_{A_l} Q_i$ and $\prod_{A_l} Q_j$ satisfy the disjoint-inclusive relationship. Now, let $\mathrm{NS}_{A_l}(Q_i)$ be the number of splits on the attribute A_l necessary for obtaining Q_i. If $\prod_{A_l} Q_i$ and $\prod_{A_l} Q_j$ overlap, one with the smaller value of $\mathrm{NS}_{A_l}(\cdot)$ includes the other.
(2) Since either $\mathrm{SplitAxes}(Q_i) \subseteq \mathrm{SplitAxes}(Q_j)$ or $\mathrm{SplitAxes}(Q_i) \supseteq \mathrm{SplitAxes}(Q_j)$, either $\forall_{A_l}(\mathrm{NS}_{A_l}(Q_i) \le \mathrm{NS}_{A_l}(Q_j))$ or $\forall_{A_l}(\mathrm{NS}_{A_l}(Q_i) \ge \mathrm{NS}_{A_l}(Q_j))$ holds.

Now, consider Q_i and Q_j such that $\prod_G Q_i$ and $\prod_G Q_j$ overlap, where G is any subset of $\{A_1, \ldots, A_n\}$. Then, $\prod_{A_l} Q_i$ and $\prod_{A_l} Q_j$ must also overlap for all A_l in G. This fact along with Conditions (1) and (2) leads to the conclusion either $\forall_{A_l} \in G(\prod_{A_l} Q_i \supseteq \prod_{A_l} Q_j)$ or $\forall_{A_l} \in G(\prod_{A_l} Q_i \subseteq \prod_{A_l} Q_j)$. In other words, either $\prod_G Q_i \supseteq \prod_G Q_j$ or $\prod_G Q_i \subseteq \prod_G Q_j$ holds meaning that one region includes the other. Therefore, $\mathcal{Q}$ forms a DIP. □

Appendix C: Proof of Theorem 1

First, we prove that every L-page of an aggregation window W_i is also an L-page of another aggregation window W_j. Let $P_1, P_2, \ldots, P_m$ be L-pages of W_i and $R_l (1 \le l \le m)$ the page grouping region of P_l. Then, by the definition of the L-page, $W_i \subset R_l$ for all R_l. In addition, since the page grouping regions satisfy the disjoint-inclusive relationship, $R_{l_1} \subseteq R_{l_2}$ or $R_{l_2} \subseteq R_{l_1}$ $(1 \le l_1, l_2 \le m)$. Therefore, there exists a permutation $\{a_l\}$ of $\{1, 2, \ldots, m\}$ such that $W_i \subset R_{a_1} \subseteq R_{a_2} \subseteq \ldots \subseteq R_{a_m}$. Then, since the aggregation windows form a partition of the grouping domain space and R_{a_1} is larger than W_i, R_{a_1} must overlap with another aggregation window W_j. Moreover, since the aggregation windows satisfy the disjoint-inclusive relationship with the page grouping regions, $W_j \subset R_{a_1}$ must hold. Since $W_j \subset R_{a_1} \subseteq R_{a_2} \subseteq \ldots \subseteq R_{a_m}$, all L-pages of W_i are also L-pages of W_j.

Using the above property, we can obtain a lower bound as follows. To guarantee one disk access per page, a page loaded into the buffer must remain until the page no longer needs to be accessed. Therefore, the L-pages of W_i loaded into the buffer during the partial aggregation for W_i (W_j) must remain in the buffer until the partial aggregation for W_j (W_i) is completed. To support this buffering, we need at least as many buffers as the number of the L-pages of W_i during the partial aggregation for W_i. In addition, we need one extra buffer for non-L-pages, which are accessed only within the partial aggregation for one aggregation window. Thus, α_i becomes 1 if at least one non-L-page exists and it is accessed after all the L-pages in W_i have been read into the buffer. In other cases, a buffer for L-pages can be used for the non-L-page obviating the need for the extra buffer. Finally, since we consider all the partial aggregations, we need as many buffers as $\max_{W_i, 1 \le i \le k}\{$(the number of L-pages of W_i) $+ \alpha_i\}$. □

Appendix D: Proof of Lemma 2

Since the regions in $\mathcal{S}$ satisfy the disjoint-inclusive relationship, we can partition $\mathcal{S}$ into subsets $\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_m (1 \le m \le n)$ as follows. First, let $\mathcal{U}_1$ be $\{S_i \in \mathcal{S} | \forall_{S_j \in \mathcal{S}} (S_i \not\subset S_j)\}$. That is, a region S_{1_i} in $\mathcal{U}_1$ is an element of $\mathcal{S}$, but is not properly included by any region $S_j \in \mathcal{S}$. Next, let $\mathcal{U}_2$ be $\{S_i \in \mathcal{S} - \mathcal{U}_1 | \forall_{S_j \in \mathcal{S} - \mathcal{U}_1} (S_i \not\subset S_j)\}$. That is, a region S_{2_i} in $\mathcal{U}_2$ is an element of $\mathcal{S} - \mathcal{U}_1$, but is not properly included by $S_j \in (\mathcal{S} - \mathcal{U}_1)$. We note that each region S_{2_i} in $\mathcal{U}_2$ is properly included in a region in $\mathcal{U}_1$; otherwise, it would be contained in $\mathcal{U}_1$ instead of $\mathcal{U}_2$. In a similar manner, let $\mathcal{U}_l (1 < l \le m)$ be $\{S_i \in \mathcal{S} - (\mathcal{U}_1 \cup \ldots \cup \mathcal{U}_{l-1}) | \forall_{S_j \in \mathcal{S} - (\mathcal{U}_1 \cup \ldots \cup \mathcal{U}_{l-1})} (S_i \not\subset S_j)\}$. Then, each region S_{l_i} in $\mathcal{U}_l$ is properly included in a region in $\mathcal{U}_{l-1}$.

Now, we prove the existence of an ISFC with $\mathcal{S}$ as the basis by constructing one ISFC. First, for $S_{m_i} \in \mathcal{U}_m$, we select an arbitrary order that starts at the lower-left corner of S_{m_i}, ends at the upper-right corner of S_{m_i}, and traverses all the points in S_{m_i}. We use this order as a space filling curve for $S_{m_i} (\in \mathcal{U}_m)$ and call it SFC$_{m_i}$. Next, for $S_{(m-1)_i} \in \mathcal{U}_{m-1}$, we also select an arbitrary order, which starts at the lower-left corner, ends at the upper-right corner, and traverses all regions S_{m_k} that are properly included in $S_{(m-1)_i}$. We use this order as a space filling curve for $S_{(m-1)_i}$. Here, for each $S_{m_k} \in \mathcal{U}_m$ traversed, we insert the order of S_{m_k}, SFC$_{m_k}$, into that of $S_{(m-1)_i}$, SFC$_{(m-1)_i}$. In a similar manner, we select an arbitrary order SFC$_{l_i}$ for $S_{l_i} \in \mathcal{U}_l (1 \le l < m)$. Here, for each $S_{(l+1)_k} \in \mathcal{U}_{l+1}$ traversed, we insert the order of $S_{(l+1)_k}$, SFC$_{(l+1)_k}$, into that of S_{l_i}, SFC$_{l_i}$. We repeat this procedure until we reach $\mathcal{U}_1$. We now make an SFC by combining all the SFC$_{1_i}$'s of S_{1_i}'s in $\mathcal{U}_1$. It is trivial to show that the SFC satisfies the condition of ISFC. Therefore, there exists an ISFC that use $\mathcal{S}$ as an ISFC basis. □

Appendix E: Proof of Lemma 3

Let $R = \prod_G \tilde{P}$ be the page grouping region of an L-page P and $\mathcal{W}\prime =< W_{i_1}, W_{i_2}, \ldots, W_{i_m} > (1 \le m \le k)$ be a list of aggregation windows overlapping with R. Then, $W_{i_j} \subset R$ by the definition of the L-page. This leads to $\bigcup_{j=1}^{m} W_{i_j} \subseteq R$. In addition, since the aggregation windows in $\mathcal{W}$ form a partition of the entire grouping domain space, R must be included in the union of the aggregation windows that overlap with R. This leads to $R \subseteq \bigcup_{j=1}^{m} W_{i_j}$. Thus, $\bigcup_{j=1}^{m} W_{i_j} = R$. In addition, since ISFC$_{\mathcal{R} \cup \mathcal{W}}$ is an ISFC that uses the union of page grouping regions and aggregation windows as the ISFC basis, all the points in W_{i_j}'s have the ISFC$_{\mathcal{R} \cup \mathcal{W}}$ values between the maximum and minimum of the ISFC$_{\mathcal{R} \cup \mathcal{W}}$ values in R. Therefore, when the aggregation windows are ordered in the ISFC$_{\mathcal{R} \cup \mathcal{W}}$ order, W_{i_j}'s are contiguous. □

Appendix F: Proof of Lemma 4

Let S be a region and p a point in the multidimensional space $\mathbb{D}$. We denote $\min_{p \in S}\{\text{ISFC}_{\mathcal{R} \cup \mathcal{W}}(p)\}$ and $\max_{p \in S}\{\text{ISFC}_{\mathcal{R} \cup \mathcal{W}}(p)\}$ by ISFC$^{min}_{\mathcal{R} \cup \mathcal{W}}(S)$ and ISFC$^{max}_{\mathcal{R} \cup \mathcal{W}}(S)$, respectively. In addition, let $R_i = \prod_G \tilde{P}_i$ and $R_{curr} = \prod_G \tilde{P}_{curr}$. A page P_i must overlap with an aggregation window W_j if it is to be accessed in a partial aggregation for W_j. Equation (2) represents the condition that P_i and W_j overlap. This is because for P_i and W_j to overlap, by the definition of the ISFC, the two lines $[\text{ISFC}^{min}_{\mathcal{R} \cup \mathcal{W}}(R_i), \text{ISFC}^{max}_{\mathcal{R} \cup \mathcal{W}}(R_i)]$ and $[\text{ISFC}^{min}_{\mathcal{R} \cup \mathcal{W}}(W_j), \text{ISFC}^{max}_{\mathcal{R} \cup \mathcal{W}}(W_j)]$ must overlap.

$$\begin{aligned}(\text{ISFC}^{min}_{\mathcal{R} \cup \mathcal{W}}(R_i) \le \text{ISFC}^{max}_{\mathcal{R} \cup \mathcal{W}}(W_j)) \wedge \\ (\text{ISFC}^{max}_{\mathcal{R} \cup \mathcal{W}}(R_i) \ge \text{ISFC}^{min}_{\mathcal{R} \cup \mathcal{W}}(W_j)) \qquad (2)\end{aligned}$$

Let W_{next} be an aggregation window to be traversed after W_{curr}. Then, $\text{ISFC}^{max}_{\mathcal{R} \cup \mathcal{W}}(W_{curr}) < \text{ISFC}^{min}_{\mathcal{R} \cup \mathcal{W}}(W_{next})$ holds. The condition $\text{ISFC}^{max}_{\mathcal{R} \cup \mathcal{W}}(R_{curr}) < \text{ISFC}^{max}_{\mathcal{R} \cup \mathcal{W}}(W_{curr})$ also holds since it is equivalent to the assumption $(\text{ISFC}_{\mathcal{R} \cup \mathcal{W}}(R_{curr}) \le \text{ISFC}_{\mathcal{R} \cup \mathcal{W}}(W_{curr}))$ of Lemma 4. We then obtain the inequality $\text{ISFC}^{max}_{\mathcal{R} \cup \mathcal{W}}(R_{curr}) < \text{ISFC}^{min}_{\mathcal{R} \cup \mathcal{W}}(W_{next})$. Therefore, R_{curr} cannot overlap with W_{next}, and P_{curr} will not be further accessed. □

Appendix G: Proof of Theorem 2

Let NP(t) represent 'the number of pages that have been previously accessed and that are still to be accessed in the future $+ \beta$' at a page access time t. Here, β is 1 if the page to be accessed at t does not exist in the buffer; 0 otherwise. To guarantee one disk access per page, once the pages are read into the buffer, they must remain in the buffer until they are no longer to be accessed. Since the toss-immediate policy chooses as the victim a page that will no longer be accessed, the required one-pass buffer size—the buffer size required for keeping the pages previously read in the buffer until they are no longer to be accessed—is equal to the maximum value of NP(t), where t spans from the first page access to the last. Now, we note that, in NP(t), the pages that have been previously accessed and that are still to be accessed are the L-pages themselves of the current aggregation window. This is because the pages that are L-pages of the previous aggregation windows and that are still to be accessed are also L-pages of the current aggregation window by Lemma 3. We also note that 'the page that does not exist in the buffer at t' is either a non-L-Page, which is accessed only in the corresponding aggregation window, or an L-page that is accessed for the first time. Therefore, when NP(t) becomes a local maximum, all the L-pages in an aggregation window must have been read into the buffer, and a non-L-page, if any, is about to be accessed. At this time, β has the same value as α_i in Theorem 1. Thus, the maximum value of NP(t) becomes $\max_{W_i, 1 \le i \le k}\{$(the number of L-pages of W_i) $+ \alpha_i\}$, which is equal to the lower bound identified in Theorem 1. □