# Chapter 8

# Compressed Aggregations for mobile OLAP

This chapter introduces the mobile Dwarf (m-Dwarf), a compressed data cube physical structure, with no loss of semantic information and explicitly designed for mobile environments [114]. *m-Dwarfs* are integrated in the *FCLOS* architecture. Due to the achieved compression, a substantial reduction of access time, energy consumption and bandwidth utilization against architectures which transmit STs is observed.

The chapter is structured as follows: Section 8.1 gives the motivation behind this work. In Section 8.2, we present related work in the area of data cube physical structures and explain the involved tradeoff in the mOLAP domain. Section 8.3 presents the *m-Dwarf* data cube physical structure and provides an extensive performance evaluation, while Section 8.4 describes how *m-Dwarfs* can be seamlessly integrated in the *FCLOS* architecture. In Section 8.5, we present the results of the experimental evaluation transmitting *m-Dwarfs* in mOLAP applications. Finally, Section 8.6 provides an overview of the examined physical structures.

## 8.1 Motivation

Wireless bandwidth is not unlimited. Therefore, mobile applications, regardless of the application domain, try to minimize the generated traffic. In mOLAP, the transmitted structures are data cubes. There exist several physical implementations of the data cube, each of them exhibiting advantages and disadvantages. Reasonably, the choice of the physical implementation of the transmitted data cubes, directly influences, among others, the mOLAP generated traffic.

All discussed mOLAP systems so far transmit STs, the simplest data cube physical structure. Although quite simple, STs have some characteristics which suit to the mOLAP domain. Nevertheless, we argue that there is room for improvement, especially for *FCLOS*. Our motivation is aggravated by the results of Fig. 6.9, which reveal that for *FCLOS*, $T_C$, namely the time clients spent receiving

data from the downlink channel, comprises 53% of the total query access time.

In this chapter, we extend the *FCLOS* architecture, which enables dissemination of aggregated data in infrastructure based wireless networks, by integrating the *m-Dwarf*, a compressed data cube physical structure. In short, the major contributions of this chapter are:

- The introduction of the highly compressed data cube physical structure *m-Dwarf*, explicitly designed to meet the requirements of mOLAP.
- The evaluation of the extended *FCLOS* architecture in conjunction with *m-Dwarfs*.

## 8.2 Background and Related Work

### 8.2.1 Data Cube Physical Structures

In this section, we discuss existing data cube physical structures. A formal definition of the data cube is given in Section 2.3.

mOLAP systems, such as *SBS* and *FCLOS*, transmit plain materialized views, also known as STs, the simplest data cube physical structure. STs are unindexed relations which consist of all tuples of a corresponding fact table. Dimensions are represented in their full cardinality. There are no precomputed and stored aggregated data according to hierarchy levels within the structure.

Naturally, the DW research community has contributed with several advanced data cube physical structures. In *Cubetrees* [131, 84], group-bys are mapped into orthogonal hyperplanes of a multidimensional index. Common sort orders are then used to cluster the points of each group-by into continuous disk space. A packing algorithm guarantees full page utilization. Updates are handled through a Merge-Packing algorithm that scans the old aggregates and merges them with the update increment, which is sorted in compatible order.

*Condensed-Cube* [162] is based on the ideas of base single tuple and projected single tuple compression. Cube tuples from the same set of fact table tuples are condensed into one cube tuple, so that one cube tuple can answer many queries without further aggregation. *PrefixCube* [47] identifies some still existing redundancies in Condensed-Cubes to achieve a further compression.

*Quotient Cube* [88] creates a summary structure by partitioning the set of cells of a data cube into classes, such that cells in a class have the same aggregate measure value. In Quotient-Cube, only the upper bound and the lower bound of a class are physically stored. *QC-Tree* [89] continues the work of Quotient-Cube and proposes a compact data structure to organize Quotient-Cube, only storing the upper bounds of Quotient-Cube classes.

*Dwarf* [149] is a highly compressed structure for computing, storing and querying data cubes. Dwarf identifies prefix and suffix structural redundancies and factors them out by coalescing their storage. Comparisons show that Dwarfs outperform previous techniques on all relevant areas: storage space, creation time,

Table 8.1: Base fact table

| A | B | C | D | Sales |
|---|---|---|---|---|
| 1 | 3 | 1 | 3 | 10 |
| 1 | 3 | 2 | 2 | 20 |
| 1 | 3 | 3 | 1 | 10 |
| 2 | 1 | 4 | 2 | 40 |
| 2 | 1 | 4 | 3 | 30 |
| 2 | 1 | 5 | 2 | 20 |

query response time and updates of cubes. *IceDwarf* [104] combines the strength of Iceberg-Cube [28] and Dwarf. It uses the Dwarf structure for cube tuple organization and only stores those cube tuples that satisfy the iceberg condition. Obviously, our proposed structure, the *m-Dwarf* is based on Dwarfs, and thus more details about Dwarfs are explained in Section 8.2.2.

However, all of these approaches were designed for traditional DWs. The cubes are stored in and queried by powerful stationary machines, using a high speed wired network. In this context, their major objective is to primarily enable query response time, fast updates and secondarily reducing the storage size. Nowadays, storage size for enterprise servers is not a real issue. On the contrary, for the mOLAP domain we are primarily interested in reducing the generated traffic, which translates in reduced storage size. Less generated traffic will positively influence query access time and energy consumption.

### 8.2.2 The Dwarf Data Cube Physical Structure

In order to facilitate the presentation of the *m-Dwarf*, this section describes the fundamentals of the Dwarf cube. Dwarf [149] is a highly compressed structure for computing, storing and querying data cubes. The construction of a Dwarf is preceded by a single sort on the fact table using one of the cube's dimensions as the primary key, and collating the other dimensions in a specific order. The choice of the dimensions' ordering has an effect on the total size of the Dwarf. Dimensions with higher cardinalities are more beneficial if they are placed on the higher levels of the Dwarf. Figure 8.1 depicts the fact of Table 8.1 (6 tuples, 4 dimensions, 1 measure resulting in a ST which occupies 120 bytes) physically implemented as Dwarf.

The height of the Dwarf is equal to the number of dimensions, each of which is mapped onto one of the levels shown in the figure. The root node contains cells of the form [*key*; *pointer*], one for each distinct value of the first dimension. The pointer of each cell points to the node below containing all the distinct values of the next dimension that are associated with the cell's key. The node pointed by a cell and all the cells inside it are dominated by the cell. For example the cell

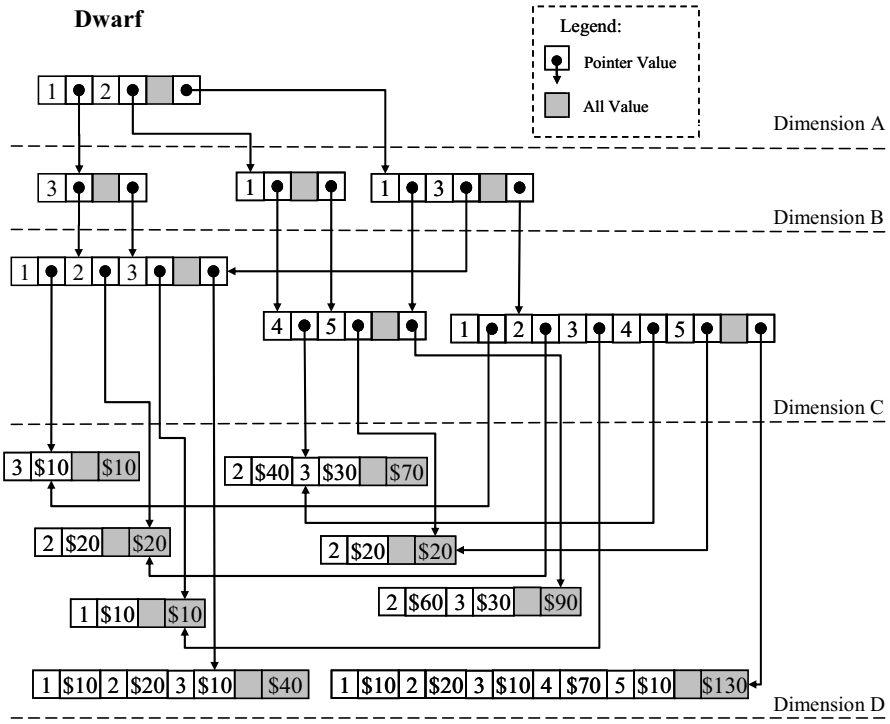


Figure 8.1: Dwarf of Table 8.1

$A1$ of the root dominates the node containing the key $B3$ whereas the cell $A2$ the node containing the key $B1$. Each non-leaf node has a special $ALL$ cell, shown as a small gray area to the right of the node, holding a pointer and corresponding to all the values of the node.

The Dwarf data structure has the following properties [149]:

1. It is a directed acyclic graph with just one root node and has exactly $D$ levels, where $D$ is the number of cube's dimensions.

2. Leaf nodes contain cells of the form: $[key; aggrValues]$.

3. Nodes in levels other than the $D$-th level (non-leaf nodes) contain cells of the form: $[key; pointer]$. A cell $C$ in a non-leaf node of level $i$ points to a node at level $i + 1$, which it dominates. The dominated node then has the node of $C$ as its parent node.

4. Each node also contains a special cell, which corresponds to the cell with the pseudo-value $ALL$ as its key. This cell contains either a pointer to a non-leaf node or to the $aggrValues$ of a leaf node.

5. Cells belonging to nodes at level $i$ of the structure contain keys that are

values of the cube's $i$-th dimension. No two cells within the same node contain the same key value.

6. Each cell $C_i$ at the $i$-th level of the structure, corresponds to the sequence $S_i$ of $i$ keys found in a path from the root to the cell's key. This sequence corresponds to a group-by with $(D - i)$ dimensions unspecified. All group-bys having sequence $S_i$ as their prefix, will correspond to cells that are descendants of $C_i$ in the Dwarf structure. For all these group-bys, their common prefix will be stored exactly once in the structure.

7. When two or more nodes (either leaf or non-leaf) generate identical nodes and cells to the structure, their storage is coalesced, and only one copy of them is stored. In such a case, the coalesced node will be reachable through more than one path from the root, all of which will share a common suffix.

A traversal in the Dwarf always follows a path of length $D$, starting from the root to a leaf node. It has the form $\langle [N_1.val|ALL], [N_2.val|ALL], ..., [N_D.val|ALL] \rangle$, meaning that the $i$-th key found in the path is either a value $N_i.val$ of the $i$-th dimension or the pseudo-value $ALL$.

### 8.2.3 The mOLAP Tradeoff: Compression vs. Client Processing

Since the primary application objective in mOLAP is to minimize query access time, this chapter examines physical structures which can, indirectly, assist this objective. It is important to underline that in mOLAP, query access time, as defined in Section 6.5, refers to the total period of time that a client spends since posing a query until the requested data is actually fetched in its local storage, while in conventional desktop OLAP applications, query access time practically refers to the response time of the OLAP server.

The choice of a data cube physical structure for mOLAP depends on the following tradeoff: the more processed (more aggregated values contained) the dataset is, the more space it occupies, and consequently less client processing for query answering is necessary. Unprocessed datasets occupy less space at the expense of increased client processing.

Moreover, even physical structures, which contain the same amount of aggregated values, might occupy more or less space and might need more or less local processing to produce query results, depending on the physical implementation.

Again consulting the results of Fig. 6.9, it appears rational to minimize as much as possible generated traffic at the expense of increased local processing. The experimental evaluation of Section 8.5 confirms this argument.

### 8.2.4 Coarse-grained Dwarfs

In the context of the aforementioned tradeoff, using STs in mOLAP has a given advantage: the lack of precomputed aggregated values substantially reduces the

total size of the transmitted structure. Since the cost of wireless communication is much higher than the computational cost of a client locally performing aggregations, this design choice seems quite rational.

As seen in Section 5.3.4, the idea of transmitting Dwarfs instead of STs was introduced by *DV-ES* [142]. It is there shown that in this way, a performance gain is feasible. The key observation is that some sub-cubes occupy less space when stored as STs, whereas others when stored as Dwarfs, even though Dwarfs contain aggregated values too. Accordingly, the smaller in size structure is transmitted. However this approach has two fundamental shortcomings:

1. The storage saving is highly dependent on the dataset.
2. Additional complexity is introduced, since clients have to handle different structures.

In order to tackle with these problems, we introduce *m-Dwarf*, a physical structure which is based on coarse-grained Dwarfs. Dwarfs achieve impressive storage savings by exploiting prefix and suffix redundancies [149]. Naturally, Dwarfs were designed for desktop applications, where storage size is less important than query performance. In order to improve query performance, the Dwarf structure contains all aggregated values. Although this approach is rational for tera byte DWs, it is ill-suited to mobile applications, since aggregated values represent a substantial percentage of the total storage size.

## 8.3 The m-Dwarf Data Cube Physical Structure

### 8.3.1 Design

It is possible to create Dwarfs, which do not contain aggregate values. The amount of aggregations can be controlled with the granularity parameter $G_{min}$ [149]. If at some level of the Dwarf structure, the number of tuples that contributes to the sub-dwarf beneath the currently constructed node of level $L$ is less than $G_{min}$, then for that sub-dwarf, no aggregation cells are computed. $G_{min} = \infty$ leads to fully coarse-grained Dwarfs (fcg-Dwarfs), Dwarfs with no aggregated values at all).

Assume a data cube consisting of $D$ dimensions. The *m-Dwarf*, exactly like its corresponding Dwarf, consists of $D$ levels. An *m-Dwarf* does not contain any aggregation values. Conceptually, it is an *fcg-Dwarf*, but physically it is stored in a different way. It looks more like a serializable structure than like a graph, which makes a transmission more straightforward. Instead of pointers, it uses two *pseudo-separators*:

- *Dimension separator*: It separates the dimensional levels of the data cube, representing what Dwarf pointers to a lower level represent. An *m-Dwarf* contains $D - 1$ dimension separators, one for each level (level $D - 1$ of the structure does not need any dimension separator).

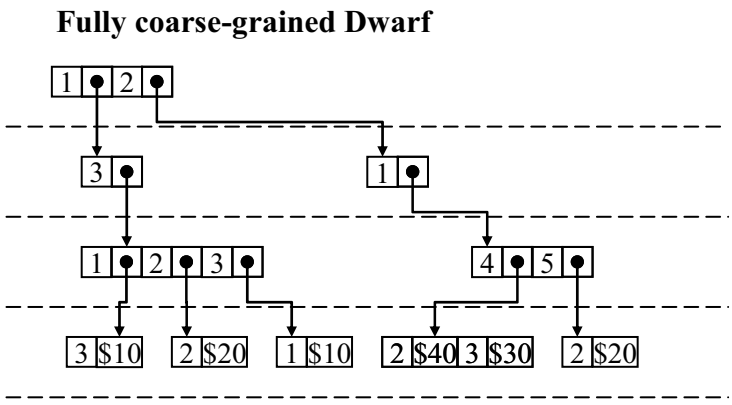


Figure 8.2: Fully coarse-grained Dwarf of Table 8.1

- *Node separator*: It separates cells of the same level. Cell values of the same Dwarf node have an ascending order. Thus, a node separator separates different sub-dwarfs. Node separators can exist in the levels {1, ..., D-1} of the *m-Dwarf*. Due to the ascending order of values, level 0 (root) does not need node separators.

Figures 8.2, 8.3 depict the fact of Table 8.1 physically implemented as *fcg-Dwarf* and *m-Dwarf*, respectively. All cells (numerical values, indices, pseudo-separators) are mapped to integer data (4 bytes).

The corresponding Dwarf occupies 376 bytes. In the *fcg-Dwarf*, all aggregated values of the Dwarf have been eliminated, decreasing its size to 120 bytes. The corresponding *m-Dwarf* achieves a higher compression occupying 104 bytes. The *m-Dwarf* contains 3 dimension separators and 1 node separator. In the second level of the structure, there is no node separator between the cell with the value 3 and the cell with the value 1, since this can be inferred by the descending order. However, a node separator is necessary between the cell with the value 3 and the cell with the value 4, since despite the ascending order, these cells belong to different sub-dwarfs.

The ascending order of values in Dwarf nodes enables a massive saving of node separators which results in a substantial storage reduction.

This trivial example shows the potential of the approach. The savings in bigger (and more realistic) datasets are much higher, as seen in following sections.

### 8.3.2 Construction

*m-Dwarfs* are constructed in a two step procedure. The first step is realized by the *CreatermDwarf* algorithm (Alg. 8.3.1). Essentially, it is a modified version of the *CreateDwarfCube* algorithm [149]. The temporary structure *rm-Dwarf* has almost the same nodes and cells as the corresponding Dwarf. The difference is

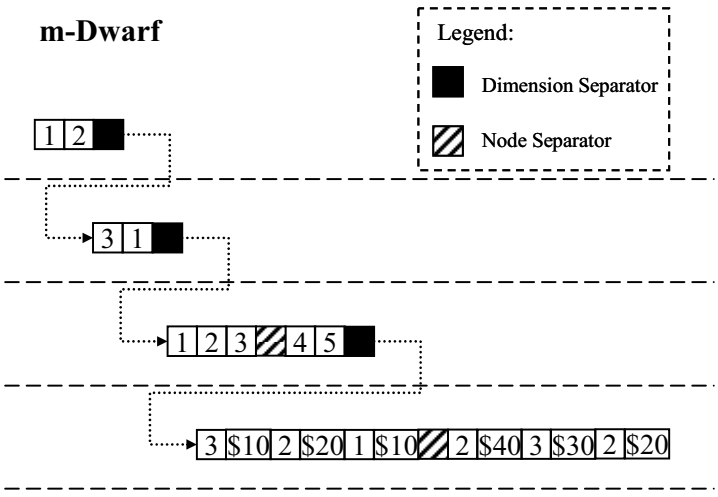


Figure 8.3: m-Dwarf of Table 8.1

that the special cells have a different meaning: they point to the *next node*, which in this context translates to the next node to be sequentially stored in the eventual array. We therefore call them *nextNodeCells*. The produced *rm-Dwarf* and the sequence in which nodes are constructed are depicted in Fig. 8.4.

**Algorithm 8.3.1:** CREATERMDWARF($F$)

*Input* : $FactTable\ F$
*Output* : $rmDwarf$

*create all nodes and cells for the first tuple*
*last tuple* $\leftarrow$ *first tuple of* $F$
**while** *more tuples exist unprocessed*
**do** {
  *current tuple* $\leftarrow$ *extract next tuple from* $F$
  $P \leftarrow$ *common prefix of current tuple, last tuple*
  $\{D - |P| - 1$ *new nodes created*$\}$
  *create nextNodeCells from previous nodes to new*
  *created nodeslast_tuple* $\leftarrow$ *current_tuple*
}
*create nextNodeCells to all rightmost nodes in all dimensions*

The second step of the construction is realized by the *CreatemDwarf* algorithm (Alg. 8.3.2), which takes as input the already constructed *rm-Dwarf*. The algorithm simply traverses the *rm-Dwarf* starting from root node and going from left to right. The *nextNodeCells* of the *rm-Dwarf* were created exactly in order to simplify this traversal. During the traversal, the *m-Dwarf* is constructed as an array which can be directly stored in a file.

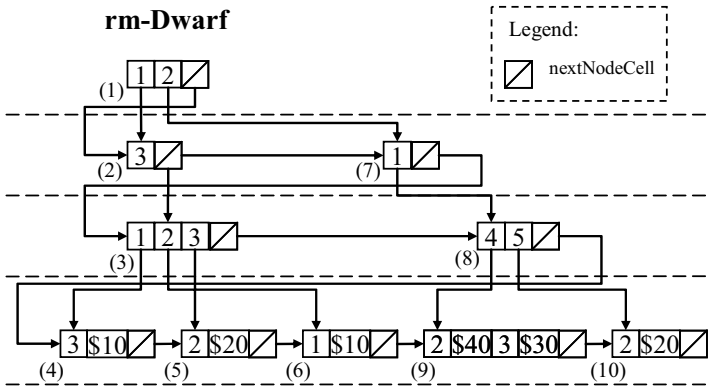


Figure 8.4: An example of the temporary structure rm-Dwarf

**Algorithm 8.3.2:** CREATEMDWARF($rmDwarf$)

*Input* : $rmDwarf$
*Output* : $mDwarf$

$mDwarf \leftarrow \emptyset$
$nextNode \leftarrow \emptyset$
**for each** *dimension* $D$
**do** {
  **for each** *node* $n$ *in* $N_d$
  **do** {
    *write all cells* $n$ *into* $mDwarf$
    $nextNode \leftarrow readNextNode()$
    **if** ($nextNode$ *exists* **and** $nextNode$ *is in dimension* $d$ **and** *last cell in* $n < first\ cell\ in\ nextNode$) **then** *write* $NS$ *into* $mDwarf$
  }
  *write* $DS$ *into* $mDwarf$
}

### 8.3.3 Evaluation

In this section, we use both semi-synthetic and real datasets to evaluate the storage savings achieved by *m-Dwarfs*. We compare our proposal against STs and fully coarse-grained Dwarfs (*fcgD*).

Tables 8.2, 8.3, 8.4 demonstrate the superiority of the *m-Dwarf* structure, in regard to storage size. All of them refer to the semi-synthetic datasets described in Section 6.6.1. $S_{ST}$, $S_{fcgD}$ and $S_{mD}$ stand for the mean size of the ST, the *fcg-Dwarf* and the *m-Dwarf*, respectively. The columns $S_{mD} < S_{ST}$ ($S_{mD} < S_{fcgD}$ ) refer to the percent of all sub-cubes, which occupy less space when stored as *m-Dwarf* rather than ST (*fcg-Dwarf*).

Table 8.2: Storage size vs. # of tuples

| **# of tuples** | $S_{ST}$ [KB] | $S_{fcgD}$ [KB] | $S_{mD}$ [KB] | $\frac{S_{mD}-S_{ST}}{S_{ST}}$ [%] | $\frac{S_{mD}-S_{fcgD}}{S_{fcgD}}$ [%] | $S_{mD} < S_{ST}$ [%] | $S_{mD} < S_{fcgD}$ [%] |
|---|---|---|---|---|---|---|---|
| 10K | 127 | 129 | 78 | 38.72 | 33.13 | 100 | 93.18 |
| 50K | 502 | 466 | 281 | 43.83 | 32.44 | 100 | 91.79 |
| 100K | 976 | 875 | 528 | 45.58 | 32.08 | 100 | 91.93 |
| 200K | 1719 | 1504 | 891 | 47.56 | 32.89 | 100 | 92.04 |
| 300K | 2428 | 2097 | 1239 | 48.19 | 32.63 | 100 | 91.79 |
| 500K | 3574 | 2932 | 1809 | 48.57 | 30.34 | 100 | 90.07 |
| 750K | 5234 | 4242 | 2603 | 49.28 | 29.80 | 100 | 90.64 |
| 1000K | 6554 | 5347 | 3193 | 50.04 | 31.13 | 100 | 90.71 |

Table 8.3: Storage size vs. dimensionality

| **# of dimensions** | $S_{ST}$ [KB] | $S_{fcgD}$ [KB] | $S_{mD}$ [KB] | $\frac{S_{mD}-S_{ST}}{S_{ST}}$ [%] | $\frac{S_{mD}-S_{fcgD}}{S_{fcgD}}$ [%] | $S_{mD} < S_{ST}$ [%] | $S_{mD} < S_{fcgD}$ [%] |
|---|---|---|---|---|---|---|---|
| 4D | 533 | 460 | 314.56 | 37.07 | 21.11 | 100 | 87 |
| 5D | 640 | 573 | 330.97 | 46.77 | 33.15 | 100 | 92.57 |
| 6D | 976 | 875 | 528.65 | 45.58 | 32.08 | 100 | 91.93 |
| 7D | 1309 | 1240 | 753.35 | 43.15 | 33.31 | 100 | 94.05 |
| 8D | 1405 | 1426 | 749.32 | 47.31 | 42.43 | 100 | 97.18 |

In Table 8.2, while keeping the dimensionality fixed to 6, we vary the number of tuples of the fact table from 10K to 1000K. In Table 8.3, while keeping the number of tuples to 100K, we analyze the effect of dimensionality. Table 8.4 investigates the effect of data skewness, controlled by the parameters of a self-similar distribution [55] on the produced data cube size. Self-similar distributions are explained in Section 6.6.1. The results reveal the superiority of our approach. *m-Dwarfs* are always smaller in size than the respective STs and smaller than the respective *fcg-Dwarfs* in more than 90% of the cube's sub-cubes.

In order to further evaluate the storage savings of the *m-Dwarf* structure, we also test bigger dataset. Apart from the real dataset, which is used as default throughout every experiment, we also test a semi-synthetic dataset of 500K tuples and 6 dimensions, as described in Section 6.6.1. Figures 8.5, 8.6 depict the storage savings of the *m-Dwarf* against STs and *fcg-Dwarfs*, for the semi-synthetic and the real dataset, respectively. For both datasets, the observations are quite similar.

Table 8.4: Storage size vs. data distribution

| **Self-Similarity** | $S_{ST}$ [KB] | $S_{fcgD}$ [KB] | $S_{mD}$ [KB] | $\frac{S_{mD}-S_{ST}}{S_{ST}}$ [%] | $\frac{S_{mD}-S_{fcgD}}{S_{fcgD}}$ [%] | $S_{mD} < S_{ST}$ [%] | $S_{mD} < S_{fcgD}$ [%] |
|---|---|---|---|---|---|---|---|
| Uniform | 1899 | 2003 | 1344 | 31.27 | 27.59 | 100 | 89.96 |
| 60/40 | 1840 | 1902 | 1194 | 36.55 | 31.12 | 100 | 92.07 |
| 70/30 | 1566 | 1511 | 925 | 41.85 | 31.85 | 100 | 92.39 |
| 80/20 | 976 | 875 | 528 | 45.58 | 32.08 | 100 | 91.93 |
| 90/10 | 390 | 351 | 196 | 47.34 | 35.73 | 100 | 90.75 |

Evidently, the *m-Dwarf* occupies less space for every sub-cube compared to STs. However, for 10% of the sub-cubes the *fcg-Dwarf* occupies less space than the respective *m-Dwarf*. Naturally, the reduction is dependent on the sub-cube. In Fig. 8.6, we make the same comparison the semi-synthetic dataset. The result is similar.

However, it is important to underline that the compression achieved by the *m-Dwarf* comes at the cost not only of no aggregated values, but also of poor indexing. While in traditional desktop DWs, compression without indexing is practically useless, we argue that for the purpose of transmitting data through a network channel, indexing requirements can be relaxed. Additional indexing structures can be constructed upon reception of the data.

## 8.4 m-Dwarfs in FCLOS

In Chapter 6, we introduced *FCLOS*, a mOLAP architecture, explicitly designed for dissemination of aggregated data in infrastructure based wireless networks. $FCLOS_{mD}$ is an extension, which instead of STs, transmits *m-Dwarfs*.

$FCLOS_{mD}$ retains the scheduling function of *FCLOS* without any modification. This means that given an incoming queue, the two schedulers would serve the requests exactly in the same order. This happens because the size of the view is never involved in the decision process. The only element in the scheduling decision that is related to the sub-cube is its dimensionality (in the $SM = R \times W \times D$ metric), which is yet independent of the physical implementation.

Unlike *DV-ES*, which transmits STs and Dwarfs, $FCLOS_{mD}$ transmits only one physical structure, *m-Dwarfs*. This alleviates clients from having to handle with two physical structures. Moreover, it simplifies the scheduling decisions.

Naturally, the impact of the new structure on the client should also be considered. According to the cost model of Section 6.5, $T_L = T_{DR} + T_{Aggr}$, where $T_{Aggr}$ represents the time a client spends aggregating a dataset in order to create the

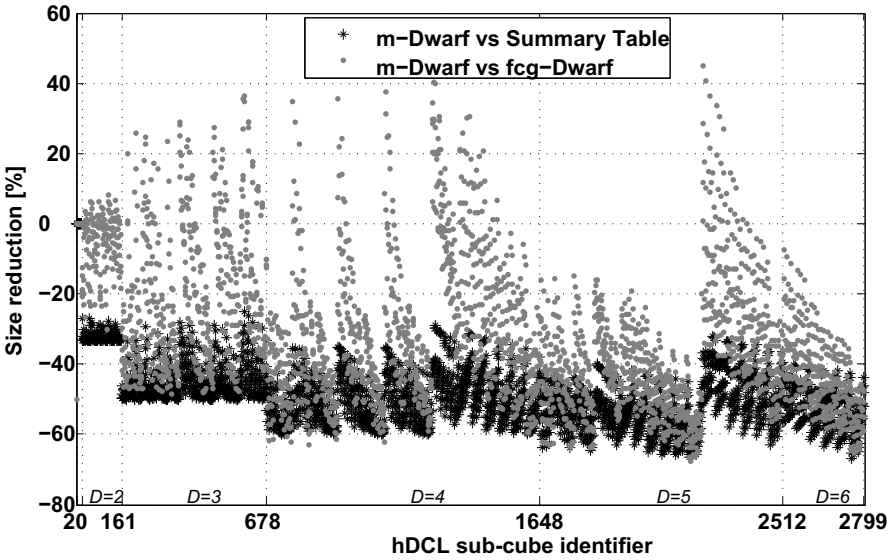


Figure 8.5: m-Dwarf storage savings for a semi-synthetic dataset (identifiers as in Fig. 2.5)

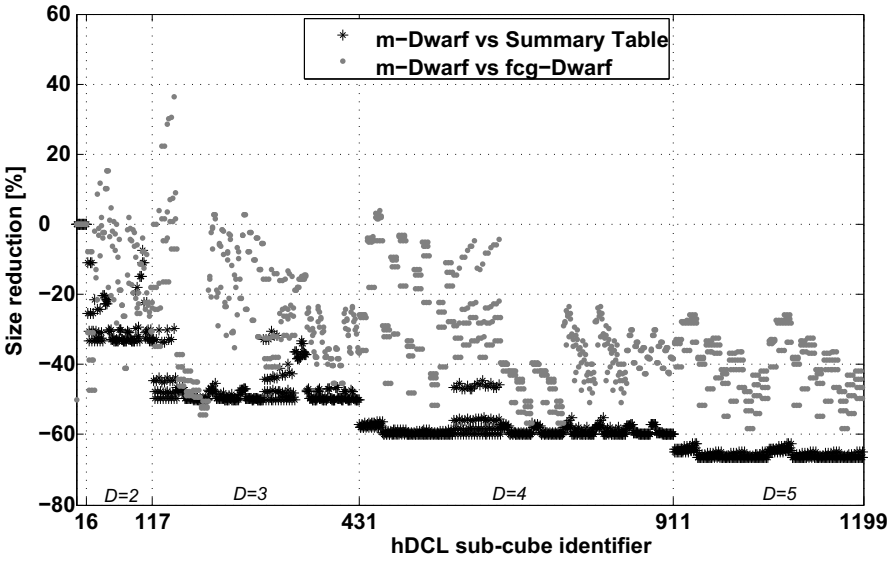


Figure 8.6: m-Dwarf storage savings for a real dataset (identifiers as in Fig. 2.5)

results of its initial query, and $T_{DR}$ represents the necessary time to retrieve the data from the hard disk to the RAM.

Regarding $T_{DR}$, it is clear that the smaller in size the structure is the less $T_{DR}$ will be. As far as $T_{Aggr}$ is concerned, plausibly *m-Dwarfs* require increased processing due to the lack of aggregated values and indices.

Depending on the application scenario there are two options:

1. Clients store and process the physical structure received (ST or *m-Dwarf*). This option avoids uncompression upon receptions, but might result in inefficient query processing.

2. Clients transform upon reception the compressed structure into a more advanced one, suitable for local answering (e.g., *fcg-Dwarf* or Dwarf). On the one hand, the advantages of advanced physical structures (query performance, updates) can be handled. On the other hand though, an uncompression overhead is introduced.

We argue that uncompression upon reception is a more rational choice, but obviously this depends on the application scenario. If the client does not want to pose any additional queries, uncompression poses an unnecessary overhead. For intensive analysis, uncompression is eventually beneficial.

## 8.5 Experimental Evaluation

The simulation environment is exactly the same used in Section 6.6.1. We evaluate the performance of $FCLOS_{mD}$, against the existing mOLAP dissemination architectures *FCLOS*, *h-FCLOS*, *SBS* [141] and *DV-ES* [142]. *h-FCLOS* is an *FCLOS* extension, which exactly as *DV-ES*, employs a hybrid approach of transmitting Dwarfs and STs [111].

### Generated Traffic

Generated traffic is a very important metric, since other applications may be running through the wireless gateway. Figures 8.7, 8.8, 8.9 depict the per query, per broadcast and total generated traffic, respectively. All of them reveal the superiority of $FCLOS_{mD}$ on the one hand, and on the other hand, an almost negligible reduction for the hybrid approach *h-FCLOS* compared to *FCLOS*. The reduced traffic of $FCLOS_{mD}$ comes without any surprise, since the transmitted structure (*m-Dwarf*) occupies less space than the respective ST. The reduction of the total generated traffic compared to *FCLOS* is 32%.

*DV-ES* despite transmitting more compressed structures, performs slightly worse than *SBS*. This can be justified by the fact that the scheduling component which they share, uses a metric directly influenced by the size of the subcubes. Therefore, the integration of any other physical structure provokes different scheduling decisions, and thus the effect of the transmitted structure cannot be

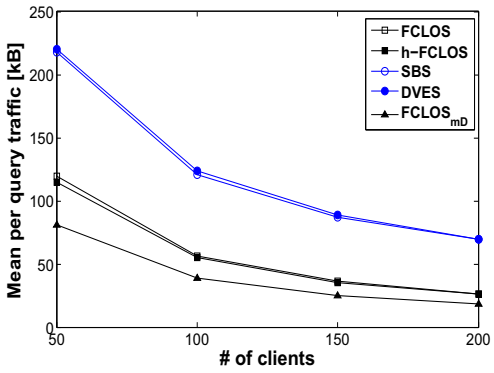


Figure 8.7: Mean per query generated traffic ($Tr_q$)

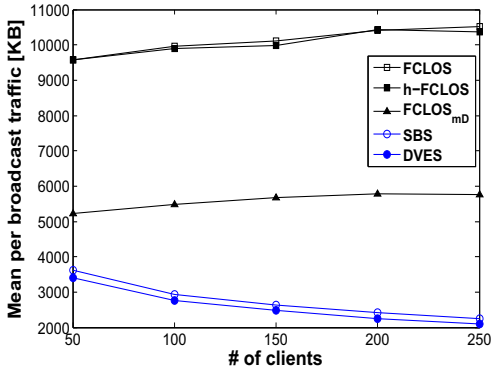


Figure 8.8: Mean per broadcast generated traffic ($Tr_b$)

isolated. On the contrary, the *FCLOS* scheduling component relies on the dimensionality of the sub-cube, which is independent of the physical implementation.

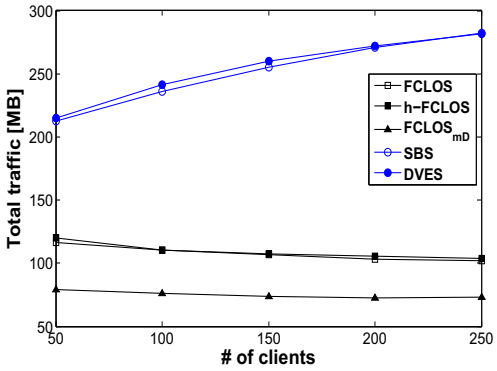


Figure 8.9: Total amount of generated traffic ($Tr_{sum}$)

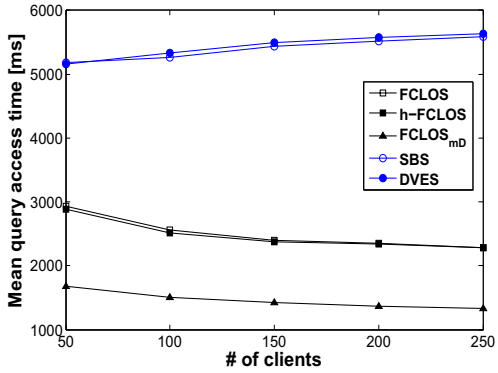


Figure 8.10: Mean query access time ($T_{all}$)

### Query Access Time

In addition to that, we compare the mean query access time. Figure 8.10 shows the results. Increasing the number of mobile clients increases the experienced access time. This was expected, since the schedulers have to serve much more clients, substantially prolonging the time a request has to spend in the queue. $FCLOS_{mD}$ outperforms its competitors by exhibiting a reduction of around 40% compared to *FCLOS*. The energy consumption results are similar.

Figure 8.11 shows the percentage of each factor $T_Q$, $T_C$ and $T_L$ to the sum $T_{all}$. As expected, the difference between $FCLOS_{mD}$ and *FCLOS* is not huge. However,

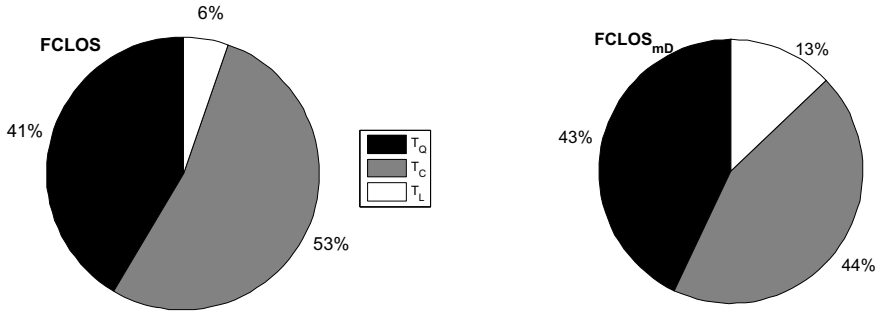


Figure 8.11: Percentage of $T_Q$, $T_C$ and $T_L$ to $T_{all}$

Table 8.5: Evaluation of examined physical structures

| | **SM** | **Dwarf** | **fcg-Dwarf** | **m-Dwarf** |
|---|---|---|---|---|
| Storage | - | - - | + | ++ |
| Query performance | - | ++ | - | - - |
| Updates | - | + | + | - |
| Construction time | 0 | + | ++ | + |
| **mOLAP** | **-** | **- -** | **+** | **++** |

there is a substantial reduction of $T_C$, due to the transmission of *m-Dwarfs*, as well as an increase of $T_C$, due to the extra processing overhead.

## 8.6 Evaluation of examined physical structures

Having presented *m-Dwarf* and exhibited its suitability for the mOLAP domain, we conclude the analysis of this chapter by providing an overview of the examined physical structures. We use the typical domain evaluation metrics storage, query performance and construction time, in addition to mOLAP suitability. Although these structures are not generally comparable, Table 8.5 attempts a comparison in the context of the mOLAP domain. The criteria are weighed again using the symbols of Table 4.1.

As already underlined, it is evident that *m-Dwarf* is a compressed physical structure, designed to occupy minimized space. Inevitably, achieving the compression, other metrics such as query performance deteriorate. However, in mOLAP the wireless channel is the bottleneck and minimized structures can be disseminated faster. Upon reception, clients can transform *m-Dwarfs* in whatever other data cube physical structure, suited for local processing.

## 8.7 Summary

This chapter deals with efficient dissemination of multidimensional, aggregated data in wireless networks, by means of compression. We introduced the *m-Dwarf*, a highly compressed data cube physical structure, explicitly designed for mOLAP. The *m-Dwarf* is by no means a general data cube physical structure; it is definitely unsuitable for traditional desktop based DWs. The achieved compression comes at the cost not only of no aggregated values, but also of poor indexing. While in traditional desktop DWs, compression without indexing is practically useless, we argue that for the purpose of transmitting data through a network channel, indexing requirements can be relaxed.

In mOLAP, the wireless channel is usually the bottleneck. Consequently, compression of transmitted data at the expense of increased client processing improves the overall performance. The *m-Dwarf* structure can be seamlessly integrated into the *FCLOS* architecture. Experimental evaluation reveals a substantial reduction of query access time, generated traffic and energy consumption. The reduction of the total generated traffic achieved by $FCLOS_{mD}$ compared to *FCLOS* is 32%, whereas compared to *SBS* 68%.

In the resulting architecture, scheduling decisions are completely separated from the size of the objects being handled. This not only allows the selection between different data cube physical structures, but also constitutes the architecture quite extendable. Any future data cube physical structure can be seamlessly incorporated.