Part III

The FCLOS Architecture

Chapter 6

Mobile OLAP in Wireless Infrastructure Based Networks

This chapter provides a detailed presentation of *FCLOS*, a complete mobile information system, explicitly designed for OLAP [112]. Specific parts of the system, requiring a more thorough analysis, are presented in Chapters 7, 8.

The chapter is structured as follows: Section 6.1 gives the motivation behind this work. In Section 6.2, we define the requirements of mOLAP architectures. Section 6.3 explains the system architecture, including the data model, the server and client architecture. In Section 6.4, the scheduling algorithm is presented. Section 6.5 explains the cost model, based on which an extensive experimental evaluation, presented in Section 6.6, takes places.

6.1 Motivation

Chapter 2 presented the background of *MDDBs*. Multidimensional data has been traditionally used in DWs to manage huge amounts of data, to aggregate and navigate into hierarchies. Nevertheless, as the amount of data, which applications have to handle, grows over the time, it is rational to expect a wider adoption of this model. Data warehousing is increasingly used not only for strategic, but for operative decision making as well. Ubiquitous access to corporate data is already success critical for every business.

In parallel, the wireless domain has been experiencing substantial growth in the past years. Great advances, both in wireless networks and respective mobile devices functioning within their proximity, enable a wide-scale adoption of such systems. In the respective application domain, vendors of mobile devices continuously produce smaller, cheaper and more powerful devices, which are able to run more sophisticated applications and network services. Consequently, organizations are deploying mobile applications because substantial business benefits can be safely assumed. As seen in Chapter 3, mobile data management is already an established research and application area.

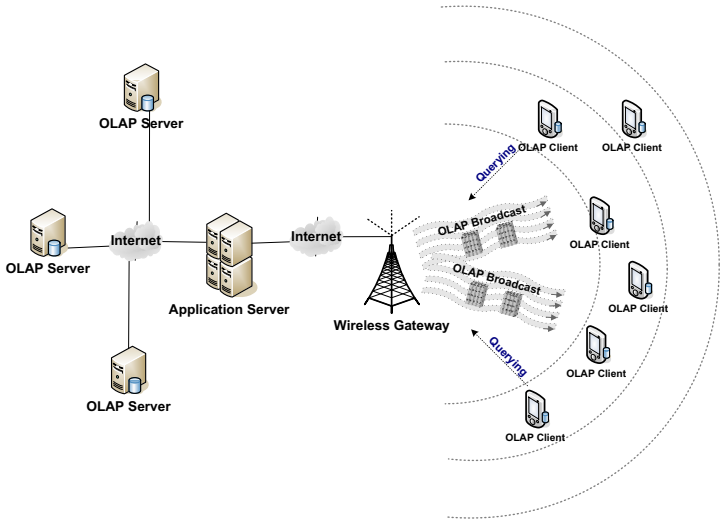

Figure 6.1: mOLAP in wireless infrastructure based networks

The intersection of the aforementioned areas motivates us to examine the case of mOLAP applications. As a motivating application scenario, we refer to an example described in [139], which considers the case of brokers accessing a stock market gallery data mart. At opening and closing times, different stocks in different financial dimensions are analyzed by many traders using some mobile device, typically laptops. Some of these stocks are more popular than other; similarly, some analytical dimensions are more important than other. In such scenarios, a data mart equipped with a broadcast gateway is responsible for serving the incoming requests. Figure 6.1 depicts a general mOLAP architecture in infrastructure based networks. It is important to underline though, that our architecture is by no means restricted to dissemination of OLAP data, but can be used for dissemination of any kind of multidimensional, aggregated data as well.

Section 4.3 thoroughly analyzed, why general broadcast systems cannot provide efficient, scalable and robust mOLAP. In Section 5.3.5, we argued that existing mOLAP architectures, although considering exclusively OLAP data, do not fully exploit its properties, being mere extensions of existing general broadcast systems. Before presenting *FCLOS* though, we define the requirements for mOLAP architectures.

6.2 Requirements

Section 5.3 defined the criteria based on which mOLAP architectures can be evaluated and thus indirectly presented the main mOLAP requirements. However, requirements and evaluation criteria are not always identical, therefore we explicitly define the requirements for mOLAP architectures:

1. *Offline functionality*: In the case of network disconnections, end users should retain offline functionality, i.e., dataset navigation capability.
2. *Online functionality*: mOLAP end users should be able to pose any kind of query, just as they would in the case of desktop OLAP.
3. *Efficiency*: mOLAP architectures should optimize query access time, energy consumption and generated traffic.
4. *Scalability*: The system's performance should be independent of the number of end clients.
5. *Self adaptiveness*: The system should not be designed under specific workload assumptions. It should operate both under relatively low workloads without unnecessarily consuming bandwidth, and under relatively high workloads retaining acceptable performance.
6. *Load balancing*: The system should maintain load balancing mechanisms, when server or clients get overloaded.
7. *Query distribution independence*: The system should be independent of the query distribution.
8. *Physical structure independence*: The system should be independent of the data cube physical implementation. The integration of any future data cube physical structure should be seamlessly feasible, without this influencing the scheduling decisions.

6.3 System Architecture

A general mOLAP architecture is depicted in Fig. 6.1. It consists of the following basic components:

- **Application (FCLOS) server**
- **OLAP servers (backend)**
- **Wireless gateway**
- **Mobile clients**

Assume n mobile clients $\{M_1, M_2, ..., M_n\}$ that issue queries using a wireless uplink channel. A wireless gateway acts as an intermediate element to finally propagate the incoming requests through the internet to the application server. There is no direct communication between mobile clients. The application server is responsible for fetching the appropriate data from the backend OLAP servers, if not already available in its local resources. Through the wireless gateway it uses a downlink channel to broadcast the data. Without loss of generality, we assume one

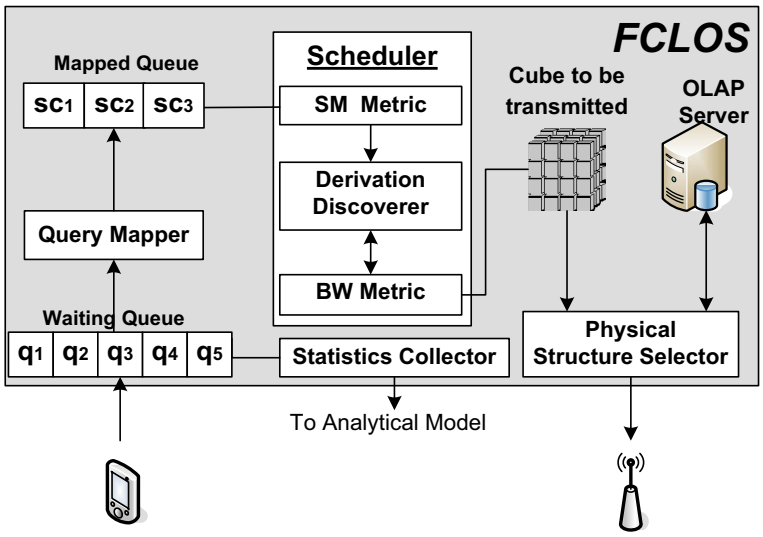

Figure 6.2: Overview of the FCLOS server architecture

downlink channel for one queried data cube. If more than one cube is queried, the server can accordingly use more downlink channels and operate multiple instances of the *FCLOS* scheduler.

6.3.1 Data Model

The assumed data model has been thoroughly described in Chapter 2. *FCLOS* does not make any explicit assumptions about the queried *MDDB*. Contrary to existing mOLAP systems, the client is allowed to pose any query targeting the database.

6.3.2 Server Architecture

The big picture of the *FCLOS* server architecture is depicted in Fig. 6.2. The following paragraphs explain the details of specific components.

Query Mapping

Upon reception of a query, *FCLOS* maps it to the corresponding node of the aggregation lattice. Section 2.6 explained how query mapping works, both for *DCL* and *hDCL*. *FCLOS* can operate with both query mappings, without any architectural modification. Optimal query mapping is the topic of Chapter 7, where we analytically and experimentally substantiate the choice of *DCL* query mapping.

Naturally, a question that may arise is why to map queries anyway. *FCLOS* employs query mapping because it is proven that the point-to-point communica-

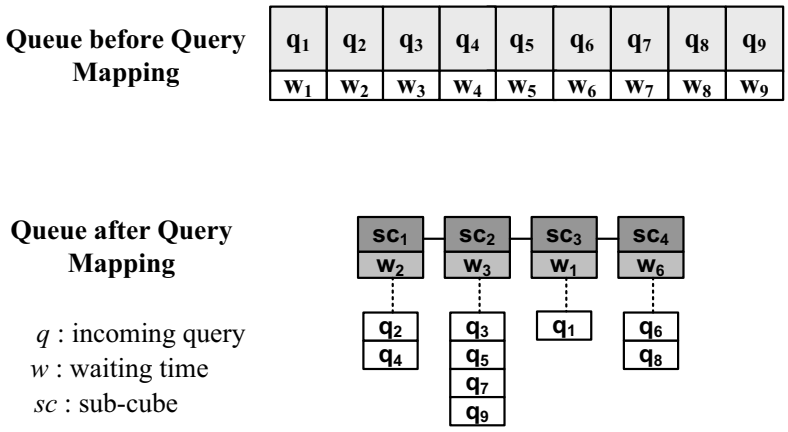

Figure 6.3: Waiting queue before and after query mapping

tion model is inefficient for mOLAP [139]. In other words, serving each query individually, assuming that clients are not able to perform local processing, not only exhibits poor performance, but does not scale with the number of requests as well. The number of possible different queries targeting a database is infinite. The objective behind query mapping is to produce a finite number of handled data items, which not only simplifies, but also assists the operation of broadcast schedulers.

Figure 6.3 depicts 9 incoming requests $\{q_1, q_2, ..., q_9\}$, mapped to their corresponding sub-cubes $\{sc_1, ..., sc_4\}$. Even in this trivial example, it can be seen that the number of handled items is reduced to 4.

Scheduling

After query mapping, the queries have to be scheduled. The scheduling algorithm, which is a fundamental component of *FCLOS*, is presented in detail in Section 6.4.

Backend

The *FCLOS* scheduler decides which sub-cube is going to be broadcast. It is assumed that the OLAP server either has already stored all possible sub-cubes or retrieves them from the backend OLAP servers. In other words, the materialization of views is not a *FCLOS* task. In addition to that, because in mOLAP transmissions last much longer than in general broadcast systems, the time required to fetch the appropriate data, does not influence scheduling.

Transmitted Structure

FCLOS design makes its scheduling decisions completely independent of the sub-cubes' physical implementation. However, the choice of a data cube physical

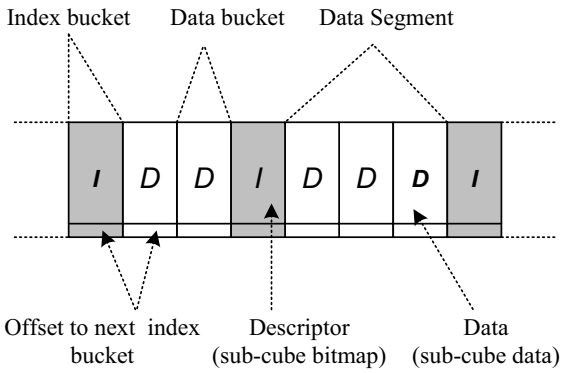

Figure 6.4: FCLOS bucket

structure influences the overall performance. Throughout this chapter, we assume that the physical structure used is a ST. STs are unindexed relations, which consist of all tuples of a corresponding fact table. In Chapter 8, we integrate the *m-Dwarf* in our architecture, which is a physical structure explicitly designed for mOLAP, and show its impact on architectural and performance issues.

Bucket Structure

FCLOS employs a very simple bucket indexing scheme. Figure 6.4 depicts the transmitted buckets. There is a conventional distinction between *index* and *data* buckets. Both bucket types contain a field that indicates the offset to the next index bucket, so that clients get informed when to tune again. The main field of the index bucket contains a *sub-cube descriptor*. This descriptor is based on the corresponding bitmap of the sub-cube and includes additional information about the identification of the multidimensional schema. For example, in a transmission of sub-cube *PT* of Fig. 2.4, the descriptor contains its bitmap 101. Obviously, the main field of the data bucket contains the sub-cube itself. The number of data buckets of a *data segment* is not fixed, depending on the size of the transmitted sub-cube.

In the mOLAP domain, complex indexing structures, such as the ones discussed in Section 3.2.1, are not necessary. This is justified by the fact that clients' queries are satisfied by only one data segment. In other words, clients do not have to tune in different data segments in order to retrieve the data buckets comprising their answer. Instead, they just need to wait until the appropriate index bucket, which precedes the requested data segment, appears. Moreover, due to the fact that sub-cubes are order of magnitude bigger than data items usually transmitted by general broadcast systems, data segments dominate the generated traffic, making the contribution of the index buckets to the generated traffic practically negligible.

Updates

mOLAP systems are centralized architectures, and thus can easily cope with updates. Assuming that the OLAP server handles updates caused by external resources, the issue ends up being a synchronization problem. The server can simply broadcast either every update or periodically in a digest way. The fact that *FCLOS* transmits fact table data enables the server to transmit only the affected tuples, and let the clients deal with the necessary bottom-up propagation of aggregations [51]. In *FCLOS*, updated tuples are transmitted in special buckets, between normal data segments. Appropriate index buckets ensure that all clients are tuned in.

6.3.3 Client Architecture

As already emphasized throughout this document, *FCLOS* assumes fat clients, which are able to store and locally process data. This facilitates, among others, offline functionality.

Querying

Clients use an uplink channel in order to send their queries to the server. *FCLOS* assumes that clients are aware of the *MDDB*'s metadata. If this is not true, i.e., this is the first query targeting the *MDDB*, an explicit query has to be issued. Alternatively, given that the size of metadata is in average order of magnitude smaller than the size of the transmitted sub-cubes, the server can periodically broadcast the metadata.

It is important to underline though, that not all queries can be answered only by the fact table. Selections or clauses might not target the values of the fact table, but the values or attributes of the dimension tables. Therefore, when clients enter the network, they must be aware not only of the schema's metadata, but of the dimension tables' values as well, if full functionality is required. Since the size of dimension tables is also typically much smaller than the size of fact tables, there are many options. The server might broadcast dimension tables through a separate downlink channel of limited capacity, or periodically through the main downlink channel, or answer on-demand. In any case, the overhead is practically negligible.

Local Processing

Clients are able to locally store and process the received data. *FCLOS* does not assume any specific database client architecture. The way data is stored and processed is naturally dependent on the data cube physical implementation. Chapter 8 explains the tradeoff between the size of the transmitted structure and the required local processing. Assuming the sub-cubes n_a and n_b for which $n_a \succeq n_b$, it suffices to simply scan every tuple of n_a to produce n_b.

Table 6.1: FCLOS scheduler notation

Notation	Definition
D	Number of dimensions
Q	FCLOS' waiting queue (after query mapping)
e	An element of the queue ($e \in Q$)
$\|Q\|$	Length of the waiting queue Q
$sc_a \succeq sc_b$	sc_a is an ancestor of sc_b
SM	The sub-cube metric of $FCLOS$
BW	The broadcast weight metric of $FCLOS$
BCL	Broadcast cluster in FCLOS

6.4 The Scheduling Algorithm

FCLOS introduces a novel family of scheduling algorithms, explicitly designed towards efficient dissemination of multidimensional data into wireless networks. Table 6.1 provides a notation overview for this section.

The scheduler handles the requests that are already mapped to the respective aggregation lattice nodes, as shown in Fig. 6.3. An element e of the $FCLOS$ waiting queue Q is an incoming query mapped to its corresponding sub-cube. We now formally present the algorithm.

6.4.1 Steps

FCLOS scheduling is a three step procedure:

Step 1

In the first step, *FCLOS* uses the novel metric *SM (Sub-cube Metric)*, defined as:

$$SM = R \times W \times D \tag{6.1}$$

where R is the number of requests for a specific sub-cube, W the waiting time of a request (1st arrival) and D the dimensionality of the requested sub-cube.

For each element e of the queue, its SM is computed:

$$\forall e \ \in Q \quad SM_e = R_e \times W_e \times D_e \tag{6.2}$$

Step 2

In the second step, *FCLOS* detects every possible *broadcast cluster* BCL. A BCL consists of one ancestor j^* and its children. The *ancestor node* in a broadcast

cluster is the sub-cube, from which all other sub-cubes comprising the *BCL* can be subsumpted:

$$\exists\ j^* \in BCL_{j^*} \subseteq Q : \forall e \in BCL_{j^*}\ j^* \succeq e \tag{6.3}$$

Note that the complexity of finding the *BCLs* is not a practical issue. While theoretically the complexity is $O(n^2)$ (where $n = |Q|$), practically the examined ancestor candidates are much fewer than $|Q|$. The algorithm considers as candidates only the requested sub-cubes, whose level l is the highest in the aggregation lattice, as well as all sub-cubes of level $l-1$ for which no lattice ancestor has been requested. Every other candidate does not have to be examined by the algorithm because it is guaranteed to be a successor of the already examined candidates, and therefore its *BCLs* are subsets of already examined *BCLs*. In practice, this step does not influence the performance of the system at all, since during transmissions, the system can schedule the next data item.

Step 3

In the final step, after having identified all possible *BCLs*, we employ a novel metric under the name *BW (Broadcast Weight)*. *BW* practically represents the weight not of one specific element in the queue, but the one of a potential broadcast cluster. If k represents a sub-cube belonging to an identified *BCL* then the *BW* of that specific *BCL* is defined as:

$$BW = \sum_{k \in BCL} SM_k \tag{6.4}$$

The algorithm computes the respective BW_c for each identified BCL_c:

$$\forall BCL_c\ \ BW_c = \sum_{k \in BCL_c} SM_k \tag{6.5}$$

Eventually, the ancestor node j^* of the cluster BCL_{j^*}, namely the cluster with the maximum *BW*, is broadcast. In this way, all clients that have requested sub-cubes which belong to BCL_{j^*} are served.

$$transmit\ j^*\ of\ BCL_{j^*}\ :\ \ \forall BW_c\ BW_{j^*} \geq BW_c \tag{6.6}$$

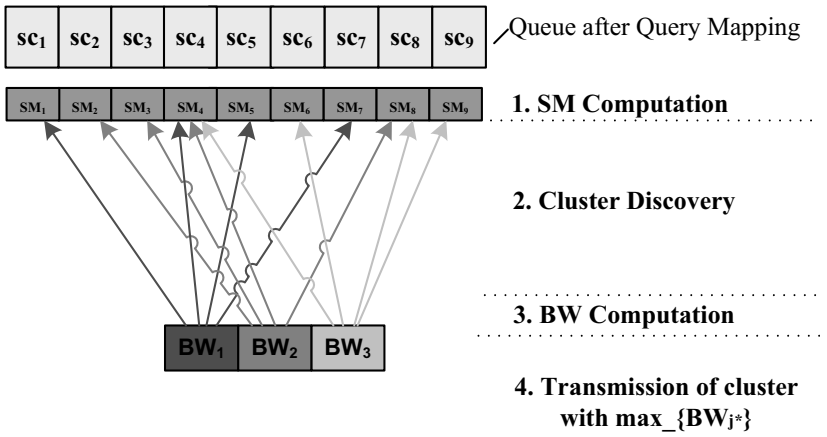

Figure 6.5: Steps of the FCLOS scheduling algorithm

Algorithm 6.4.1: SCHEDULER($^{**}Q$)

Input : *Queue* Q
Output : $Sub-cube$ *to be transmitted* j^*

global $^*sm, ^{**}bcl, ^*bw, j^*$
comment: sm is the array holding the SM values
comment: bcl is the array holding all recognized BCLs
comment: bw is the array holding the BW values for all BCLs
comment: j^* is the sub-cube index
main
1 : *Compute SM for every queue element* $(sub-cube)$
2 : *Find all possible clusters* $(BCLs)$
3 : *Compute* BW_k *for every identified cluster* BCL_C
4 : *Select for transmission the ancestor node of the cluster* BCL_{j^*}, *whose* BW_{j^*} *is maximum*
return (j^*)

Figure 6.5 illustrates the operation of the scheduling algorithm.

6.4.2 Analysis

Having formally presented the scheduling algorithm of *FCLOS*, we proceed by explaining the intuition behind the algorithm's design. Obviously, the SM prioritizing metric is influenced by the $R \times W$ metric, described in Section 4.1.1. Previous mOLAP systems adapt this metric for the specific domain by using the

Table 6.2: A scheduling example: STOBS vs. FCLOS

	R	**W**	**S**	**D**	$\frac{R \times W}{S}$	$R \times W \times D$
S	3	200	15	1	40	600
PT	2	15	30	2	1	30
PS	1	100	20	2	5	200
P	4	150	10	1	60	600

$\frac{R \times W}{S}$ metric.

Contrary to the well established practice of promoting smaller in sizes items, not only in mOLAP, but in general broadcast systems as well, *FCLOS* uses an unconventional approach and promotes bigger in size items. This is the intuition behind the SM metric, which puts the dimensionality D in the nominator, thus promoting bigger sub-cubes.

The dimensionality of a sub-cube is generally proportional to its size, but naturally in this way SM does not distinguish between sub-cubes with the same dimensionality, although big size differences between them may exist. This is an intended compromise in order to keep the scheduling decisions completely independent of the data cube physical implementation. Chapter 8 provides more insight about this issue.

But why does *FCLOS* promote sub-cubes with higher dimensionality? The answer is straightforward. Since the distinguishing characteristic of mOLAP scheduling is the exploitation of subsumptions, the degree of exploitation should be maximized. Sub-cubes with higher dimensionality, or the corresponding nodes of the aggregation lattice have more successors than nodes being in a lower level of the lattice. For instance, a *DCL* node with dimensionality D has $2^D - 1$ successors. Therefore, the dimensionality D should be a positive prioritizing factor, exactly as in the SM metric.

In addition to that, the scheduling algorithm does not produce the schedule based exclusively on the prioritizing metric SM. *FCLOS* is designed to exploit every subsumption probability. In this context, when identifying *BCLs*, every queue element e is considered as an ancestor candidate an. The search for *BCLs* is completely independent of traditional scheduling metrics such as R or W.

By introducing our new metric BW and by separately detecting all possible clusters, *FCLOS* does not exploit subsumptions whenever this is possible but rather enforces it. This results in a better exploitation of the broadcasting feature, since, quite expectedly, the number of members of the served BCL_{j^*} is now higher in average. Results for new queries have now higher probability of already existing in client's storage, partly due to previous scheduling decisions taken by *FCLOS*.

The following example illustrates the scheduling procedure of *FCLOS* and *STOBS* (we omit *SBS* due to its similarity with *STOBS*). Assume a queue with 4 sub-cubes (*PS, PT, P, S*), from the *DCL* of Fig. 2.4 and the values of Table 6.2, for *R, W, S* and *D*. While the operation of *STOBS* after the computation of the

Table 6.3: Clusters and broadcast weights in FCLOS

BCL	**BW**
(***PS***, *S*, *P*)	1400
(***PT***, *P*)	630
(***P***)	600
(***S***)	600

Table 6.4: Schedules: STOBS vs. FCLOS

	1st	**2nd**	**3rd**	**4th**
STOBS	***P***	***S***	***PS***	***PT***
FCLOS	(***PS***,*S*,*P*)	***PT***	-	-

$\frac{R \times W}{S}$ metric is straightforward, *FCLOS* searches candidate clusters and finds the four shown in Table 6.3. Finally, as seen in Table 6.4 the two schedulers produce entirely different schedules. Even in this trivial example, it is evident that for the same incoming load, *FCLOS* needs 2 transmissions, whereas *STOBS* 4. Naturally, the number of transmissions alone does not suffice for a thorough evaluation. Section 6.6 provides detailed insight.

6.5 Cost Model

The performance of mOLAP systems can be evaluated using the following metrics:

- *Query Access Time:* The total period of time that a client spends since posing a query until the requested subset is actually fetched in its local storage (T_{all}). It is the time the request spends in the server's waiting queue T_Q, plus the time the client spends receiving data from the downlink channel T_C, plus the time the client locally processes the data T_L.

$$T_{all} = T_Q + T_C + T_L \tag{6.7}$$

- *Energy Consumption:* The energy a client consumes since posing a query until the requested subset is actually fetched in its local storage. It is the energy consumed in doze mode waiting for appropriate data in the downlink channel E_Q, plus the consumed energy being in active mode and receiving data from the downlink channel E_C, plus the energy consumed for local data processing E_L.

$$E_{all} = E_Q + E_C + E_L \tag{6.8}$$

- *Generated Traffic:* Amount of data transmitted by the server into the wireless network. Experimentally, it can be quantified with several metrics: generated traffic per issued query Tr_q, generated traffic per broadcast Tr_b or total amount of generated traffic Tr_{sum}.

Table 6.5: Real data mart metadata

Dimension	Hierarchical levels	Cardinality
A	4	55000
B	4	1826
C	3	503
D	3	72
E	2	5

6.6 Experimental Evaluation

This section presents the results of extensive experimental evaluation of *FCLOS*. Our system is compared against the state of the art mOLAP systems *STOBS* and *SBS* using a simulator written in *C*. We do not include a point-to-point system in our evaluation, since it has been shown to exhibit much poorer performance than both *SBS* and *STOBS* [141, 139].

6.6.1 Simulation Environment

Mobile clients, randomly distributed in a square plane, query a data mart. Queries are propagated periodically using an 802.11 wireless network. When a suitable answer is received, the client issues a new query after a uniformly distributed time span. A complete evaluation of mOLAP systems requires the consideration of many parameters. The following paragraphs explain the details about the dataset, the workload and the client model used. Note that the same simulation environment is used in the following chapters as well.

Dataset

We used both synthetic and real datasets. However, we used a real but anonymized dataset as default. It is a data mart consisting of 5 hierarchical dimensions and 918,843 tuples. For each dimension, the number of hierarchical levels and its cardinality are shown in Table 6.5. The *DCL* for this dataset consists of 32 nodes, whereas the respective *hDCL* of 1200 nodes. As already discussed in Section 6.3.3, the size of the dimension tables is practically negligible compared to the size of the fact table. This is the case for this dataset too.

We created a semi-synthetic dataset in the following way. Based on the metadata of the real data mart used in [149], we populated the schema using an 80/20 self similar distribution. Self-similar distributions [55] have the property that within any region of the distribution, the skew is the same as in any other region. So, for example, all subranges of the 80/20 self similar distribution follow the 80/20 rule (h=0.20). For integers in $[1, N]$, the first $h \times N$ integers get $1 - h$ of the distribution. For example: if N=25 and h=0.20, then 80% of the weight goes

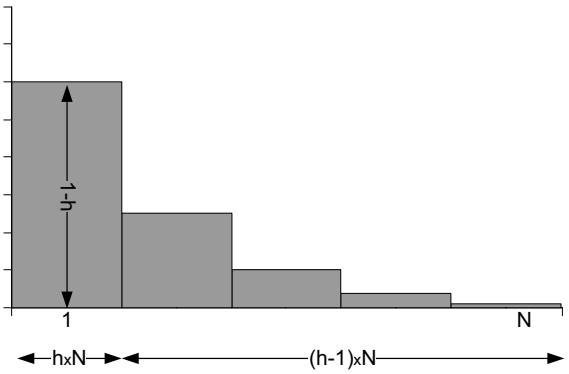

Figure 6.6: Histogram of self-similar distribution

Table 6.6: Semi-synthetic data mart metadata

Dimension	Hierarchical levels	Cardinality
A	6	7458
B	4	2765
C	4	3857
D	3	213
E	1	3247
F	1	660
G	1	4
H	1	4

to the first 5 integers and 64% of the weight goes to the first integer. Self-similar distributions have been extensively used to produce skewed database datasets. Figure 6.6 shows the histogram.

This data mart consists of 8 hierarchical dimensions. For each dimension, the number of hierarchical levels and its cardinality are shown in Table 6.6. The *DCL* for this dataset consists of 256 nodes, whereas the respective *hDCL* of 11200 nodes.

Query Distribution

While the server employs query mapping, clients are unaware of that. Consequently, in our experiments clients issue queries, exactly as they would in a traditional desktop environment. Similarly to [149] we use the following probabilities to model the query distribution:

- P_{dim}: The probability that each dimension is selected to participate in the

Table 6.7: Query distribution's characteristics

Workload	P_{dim}	P_{new}	P_{drill}	P_{roll}	P_{point}
WL_A	50%	50%	25%	25%	50%
WL_B	50%	30%	35%	35%	50%
WL_C	50%	100%	0%	0%	50%

new query. For example, for a 6-dimensional cube, if P_{dim}=0.5, then new queries include 3 dimensions on average.

- P_{new}: The probability that the new query is not related to the previous query. In OLAP applications, users typically perform a query, and then often execute a series of roll-up or drill-down queries.
- P_{drill}: The probability that the new query is a drill-down of the previous query.
- P_{roll}: The probability that the new query is a roll-up of the previous query.
- P_{point}: The probability that we specify just a single value for each dimension participating in a query. Otherwise, with probability 1-P_{point} we will specify a range query for that dimension.

Table 6.7 shows the values for the aforementioned probabilities, for the 3 query distributions used in the experiments.

Client Model

This paragraph explains the simulated client model. More specifically, we explain how T_L and E_L are measured.

The time a client M_c locally processes the data T_L is:

$$T_L = T_{DR} + T_{Aggr} \tag{6.9}$$

T_{Aggr} represents the time M_c spends aggregating a dataset in order to create the results of its initial query. If the received dataset is exactly what M_c had requested, then obviously T_{Aggr}=0. If M_c is able to answer a query locally, i.e., relying on stored data, T_{DR} represents the necessary time to retrieve the data from the hard disk to the RAM.

The time for x bytes of data to be aggregated in the cache is:

$$T_{Aggr}(x) = \frac{x}{BusBandwidth} \tag{6.10}$$

The time for x bytes of data to be transferred from the hard disk to the RAM is:

$$T_{DR}(x) = \frac{x}{TR_{HD}}$$

Electrical energy consumption for a time span t is generally given by the following equation:

$$E = P \times t \Longleftrightarrow E = V \times I \times t \tag{6.11}$$

where P is the electrical power, V the voltage supply and I the current.

The energy consumption for x bytes of data to be received by the wireless card:

$$E_C(x) = \frac{P_{Rx} \times x}{WBandwidth} \tag{6.12}$$

where P_{Rx} is the electrical power during reception.

Similarly for the energy consumption E_L it is:

$$E_L = E_{DR} + E_{Aggr} \tag{6.13}$$

E_{Aggr} represents the energy M_c consumes aggregating a dataset in order to create the results of its initial query. If the received dataset is exactly what M_c had requested, then obviously E_{Aggr}=0. If M_c is able to answer a query locally, i.e., relying on stored data, E_{DR} represents the energy consumed to retrieve the data from the hard disk to the RAM.

The energy consumed for x bytes of data to be aggregated in the cache is:

$$E_{Aggr}(x) = T_{Aggr} \times P_{CPU} \tag{6.14}$$

The energy consumption for x bytes of data to be fetched by the hard disk:

$$E_{DR}(x) = T_{DR}(x) \times P_{DiskRead} \tag{6.15}$$

Table 6.8 summarizes the mobile client's simulated characteristics. Note that these values constitute a worst case scenario, since they correspond to PDA clients. If values for typical laptops are used instead, our system would only benefit, since our assumption of the wireless channel being the bottleneck would become even more valid.

6.6.2 Basic Evaluation

Table 6.9 provides an overview of the simulation parameters. Throughout the next paragraphs, unless explicitly defined, the default simulation values are used. Moreover, mean stands for the arithmetic mean defined as $\frac{1}{k}\sum_{i=1}^{k} x_i$, where k the number of repetitions and x_i the measurement for repetition i.

Query Access Time

We begin the experimental evaluation with the results of the mean query access time T_{all}. According to the cost model of Section 6.5 it is $T_{all} = T_Q + T_C + T_L$. Figure 6.7 reveals the superiority of $FCLOS$ against its competitors. Unsurprisingly, $STOBS$ and SBS essentially exhibit the same performance, since as explained in

Table 6.8: Mobile client technical characteristics

Metric	**Value**
Processor-RAM (Source: [2])	
RAM Clock Frequency	54 MHz
CPU Frequency	312 MHz
Bus Width	32 bits
Bus Bandwidth	1728 Mbit/s
Power (P_{CPU})	2 mW
Wireless Card (Source: [4])	
Voltage Supply	3.3V
Rx	170 mA
Tx	265 mA
Idle	75 mA
Bandwidth	11-54 Mbit/s
Hard Drive (Microdrive) (Source: [1])	
Voltage Supply	3.3V
Write Consumption	240 mA
Read Consumption	230 mA
Transfer Rate (TR_{HD})	10 Mbyte/s

Section 5.3, they constitute very similar approaches. The novel approach of *FCLOS* not only achieves a reduction of over 50%, but proves scalable too. As the client population grows, so does the number of incoming queries. *FCLOS* manages to build bigger *BCL*s and thus serve more clients per broadcast. On the contrary, its competitors, resembling traditional on-demand systems and sub-optimally exploiting subsumptions, experience slightly increased access time.

Naturally, it is necessary to analyze the effect of each factor T_Q, T_C and T_L to the sum T_{all}. The results of Fig. 6.9 reveal that while for *FCLOS* T_Q and T_C represent 41% and 53% of T_{all}, respectively, T_L represents only 6% of T_{all}. This is a very important ascertainment, since it confirms that server and wireless channel are the system's bottlenecks indeed. The same applies for *STOBS* and *SBS* regarding the T_{all}. However, in their case T_{all} is dominated by the T_Q, confirming the scheduling algorithm's inefficiency. On the contrary, in *FCLOS* T_{all} is more influenced by T_C. This is justified by the fact that *FCLOS* generates more traffic per broadcast, prolonging the T_C, as shown in the following paragraphs. [t]

In the next experiment, we investigate the percentage of online and offline answers. We call offline the queries that were able to be answered locally, without any server interaction, and online the ones that need a server transmission. Figure 6.10 demonstrates a critical system behavior. *FCLOS* does not have to resort to a server connection in 81% of its queries, while its competitors in approximately 60%. The conclusion is twofold. On the one hand, online performance is enhanced

Table 6.9: Overview of simulation parameters

Simulation Parameter	Range	Default
Bandwidth	1-54 Mbit/s	11 Mbit/s
Client Population	50-250	100
Dataset	real, synthetic	real
Cube Dimensionality	4-8	5
Workload	WL_A, WL_B, WL_C	WL_A
Query Interval	1-20s	3s
Queries per client	5-15	10

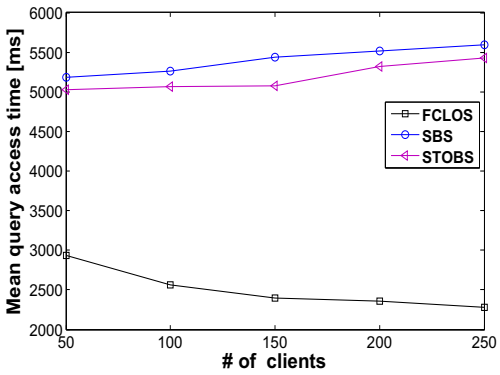

Figure 6.7: Mean query access time (T_{all})

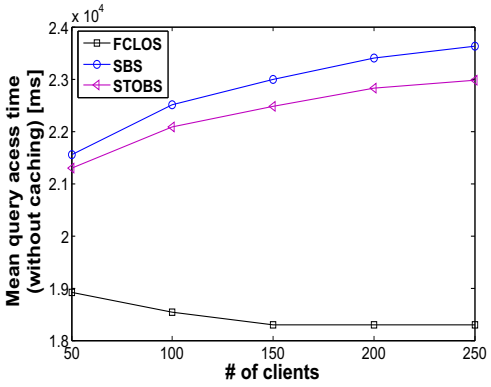

Figure 6.8: Mean query access time without offline answers

because queries that do not have to be sent to the server not only are served faster, but alleviate the server as well, thus accelerating the execution of the rest of the queries. On the other hand, offline functionality is substantially increased.

But does *FCLOS* perform so well, simply because it answers more frequently queries locally? The answer is no. Figure 6.8 provides a subset of the results of Fig. 6.7. It represents the access time for the online queries only. Again the superiority of *FCLOS* is evident, retaining the desired scalability. Naturally, the optimization for these queries is not so big. Remember that the fundamental objective of *FCLOS* is to facilitate offline functionality, so that a connection is not necessary. However, it is evident that even without exploiting client local resources, *FCLOS* outperforms its competitors.

Energy Consumption

The energy consumption overhead is a crucial metric, given the objective of maximizing the operating time of mobile devices. The results of Fig. 6.11 are reasonably

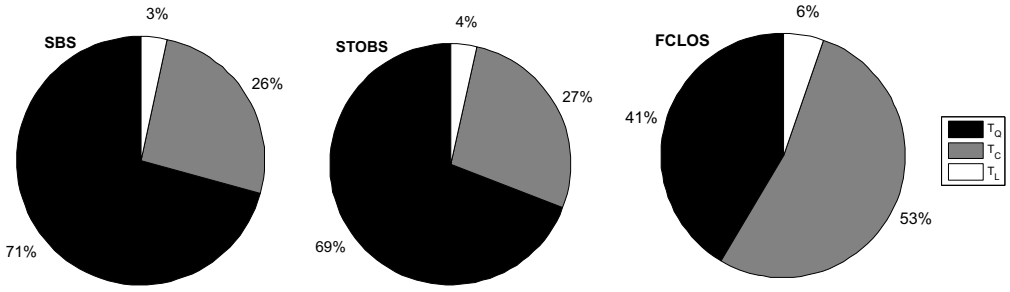

Figure 6.9: Percentage of T_Q, T_C and T_L to T_{all}

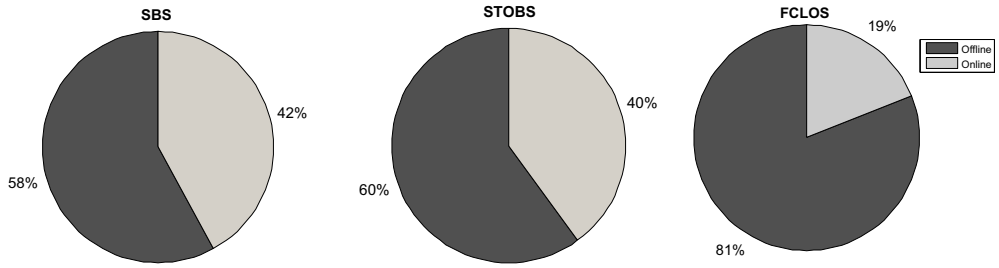

Figure 6.10: Percentage of offline and online query answers

similar to the ones of the query access time. The reason for that is that the two metrics are related. The more a client waits for the answer, the more energy it consumes.

Similarly to Fig. 6.9, Fig. 6.12 depicts the effect of each factor E_Q, E_C and E_L to the sum E_{all}. The main distribution difference compared to the access time, is that the energy consumption is dominated by E_C. This is justified by two facts. Not only are clients in active mode when downloading and in doze mode when waiting for the requested data to appear in the downlink channel, but moreover as seen in Fig. 6.9, *FCLOS* queries spend more time in the downlink channel than in the queue as well.

Generated Traffic

The amount of per query generated traffic is a very important metric, since other applications may be running through the wireless gateway. Moreover, this is always a significant factor in volume based networks. The importance becomes even bigger, considering that the wireless channel is the system's main bottleneck.

Figure 6.13 depicts the generated traffic per issued query. Remember that each broadcast serves more than one query. Again the superiority of *FCLOS* is evident, reducing the generated traffic by over 50%. Remarkably, all approaches scale well. This justifies the subsumption based scheduling followed by all approaches, regardless of the specific implementation.

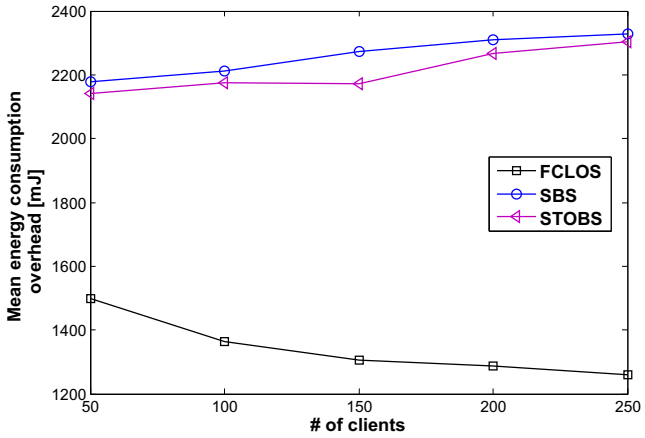

Figure 6.11: Mean energy consumption overhead (E_{all})

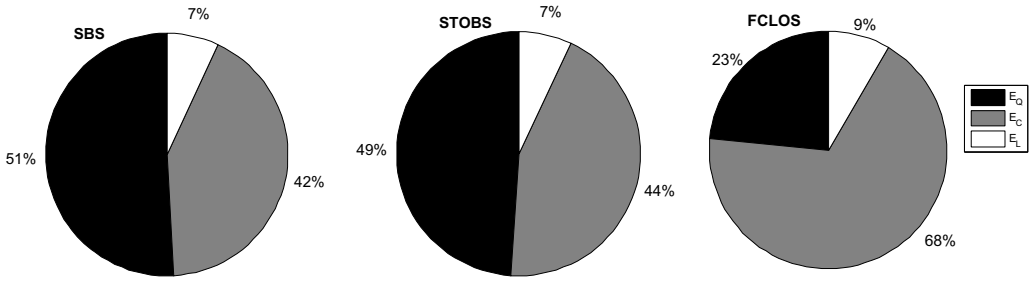

Figure 6.12: Percentage of E_Q, E_C and E_L to E_{all}

In Fig. 6.14 on the contrary, where the per broadcast generated traffic is depicted, a completely different behavior can be observed. *FCLOS* transmits clearly more data per broadcast. Despite the emerged contradiction with Fig. 6.13, the results are absolutely consistent. The intuition behind *FCLOS* is exactly to transmit bigger data cubes and thus serve more clients, hence requiring fewer broadcasts. Indeed, in this experiment *FCLOS* had to broadcast almost three times less than its competitors. In the same context, Fig. 6.15 shows the average queue length, and the number of requests served and not served per broadcast. Evidently, not only the average queue length of *FCLOS* is smaller, but every transmission manages to serve around 50% of the pending queries.

Stretch

The stretch for a request i is $stretch_i = \frac{AT_i}{ST_i}$, as defined in Section 4.1.1. In mOLAP, if request i is the only system job, then T_Q=0, T_C is the transmission time for exactly the required dataset (no subsumption), and T_L=0. Figure 6.17 demonstrates a weakness of *FCLOS*. Evidently, *FCLOS* exhibits a poor perfor-

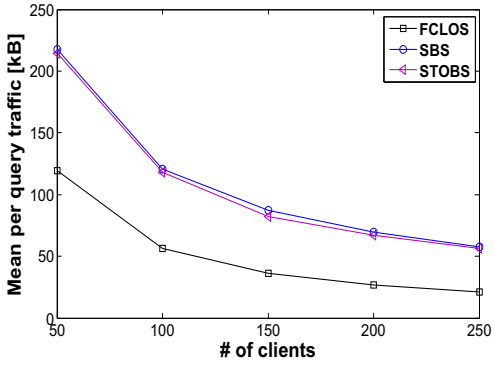

Figure 6.13: Mean per query generated traffic (Tr_q)

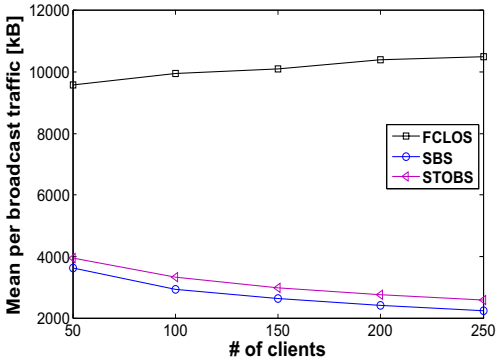

Figure 6.14: Mean per broadcast generated traffic (Tr_b)

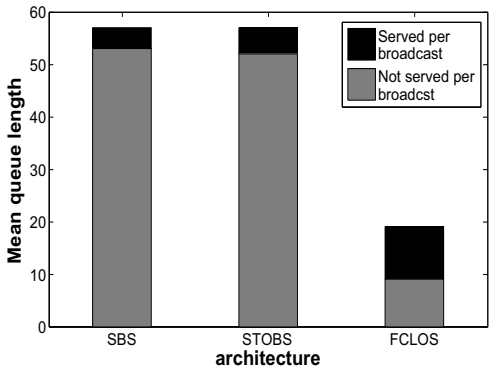

Figure 6.15: Mean queue length and number of elements served or not per broadcast

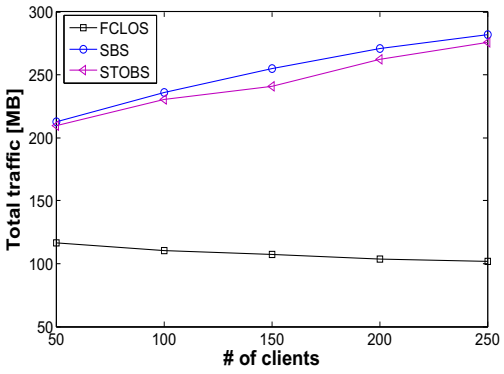

Figure 6.16: Total amount of generated traffic (Tr_{sum})

mance in comparison with its competitors. This comes without surprise though. *FCLOS* fundamental design principle is to serve as many clients as possible with each broadcast. Inevitably, queries targeting smaller datasets frequently have to deal with bigger datasets and increased local processing. This inevitably incurs increased stretch.

Nevertheless, stretch is an indicator of fairness, not performance. In order to investigate how different query classes are treated, we partitioned the set of all sub-cubes into four classes according to size. Class A represents the smallest in size sub-cubes and class D the biggest in size sub-cubes (classes B, C accordingly in between). Figure 6.18 reveals that despite its poor stretch, *FCLOS* outperforms its competitors for each query class. We argue that the improvement achieved by *FCLOS* is for each class so strong, that it is essentially of minor importance if the stretch deteriorates.

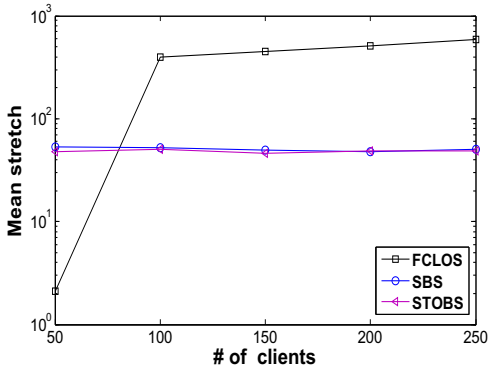

Figure 6.17: Mean stretch

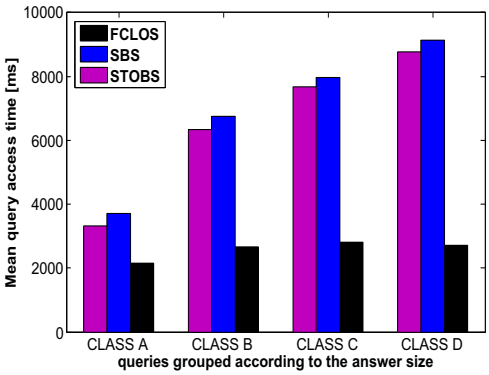

Figure 6.18: Mean query access time (T_{all}) for different job classes

a-optimizer

Presenting *STOBS* in Section 5.3.2, we explained its flexibility in searching for ancestors of broadcast clusters. This flexibility is controlled by the a-optimizer. This is a very important configuration parameter for *STOBS*, since in [139], the authors identify an inherent tradeoff between the optimization of energy consumption and access time, which can be controlled through the a-optimizer. Although theoretically this could be employed by *FCLOS* as well, this is not necessary as revealed by Fig. 6.19. For both approaches we implemented extensions using the values 1, 3 and 5 for a. Value a=5 results in full flexibility (because the used dataset consists of 5 dimensions), exactly how default *FCLOS* operates. In accordance with the results presented in [139], higher value of a reduces *STOBS*' access time, but at the cost of increased energy consumption. On the contrary, the value a=5 guarantees optimization of both metrics in *FCLOS*. Therefore, *FCLOS* not only avoids one additional configuration parameter, but optimizes both metrics as well. *FCLOS* does not need any kind of tunable parameter to control the exploitation of subsumptions. Note that every previous experiment involving *STOBS* assumes a=5. Due to the similarity of the definition of a in *SBS* and the produced results, its results are omitted in this chart.

6.6.3 Further Evaluation

The performance of mOLAP systems is subject to many parameters. In order to ensure a complete evaluation, the following paragraphs analyze the performance under different influencing factors: dataset, query distribution, query rate and bandwidth.

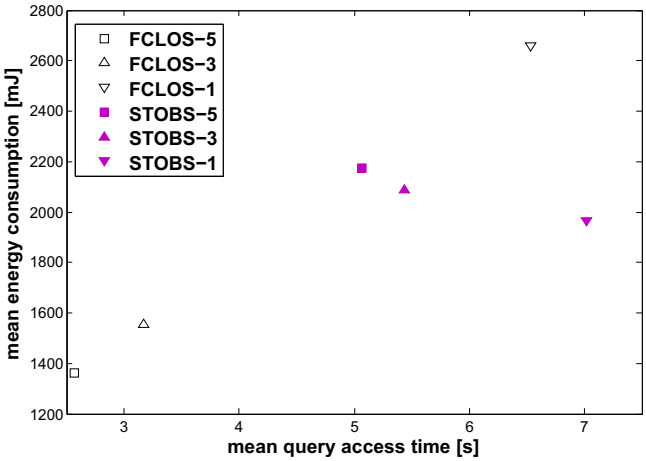

Figure 6.19: Mean query access time (T_{all}) for different values of the a-optimizer

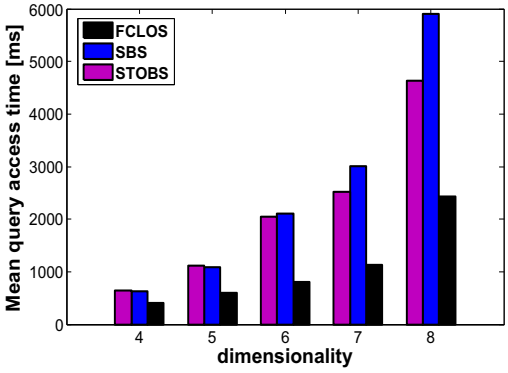

Figure 6.20: Mean query access time (T_{all}) vs. dimensionality

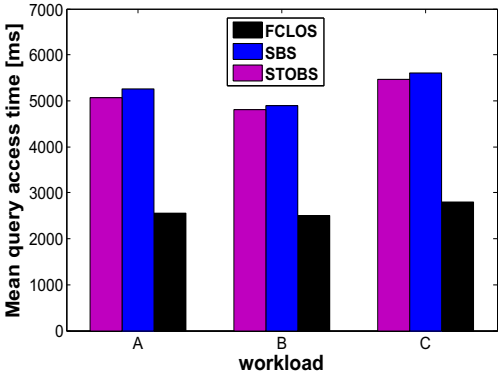

Figure 6.21: Mean query access time (T_{all}) vs. query distribution

Dataset

The following experiment reveals the effect of dimensionality on mOLAP architectures. For this purpose, we use the semi-synthetic dataset described in 6.6.1, keeping the number of tuples to 500K. Dimensionality D is a critical parameter because the number of *DCL* nodes is 2^D, and thus directly influences query mapping. Figure 6.20 confirms the superiority of *FCLOS*, regardless of the number of dimensions. Despite the increase of access time as the number of dimensions grows (which is expected because the size of cubes also increases), *FCLOS* copes with more dimensions better than its competitors. The slight difference between *STOBS* and *SBS*, particularly when the dimensionality becomes higher, can be justified by their similar, but not identical, a-optimizer components.

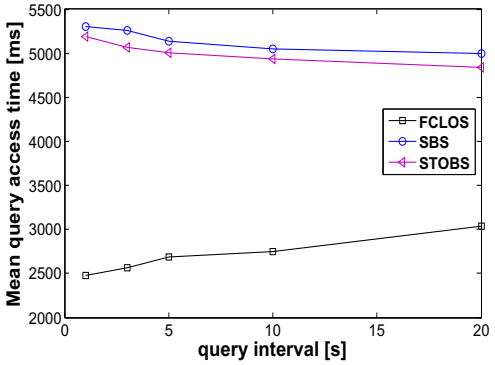

Figure 6.22: Mean query access time (T_{all}) vs. query rate

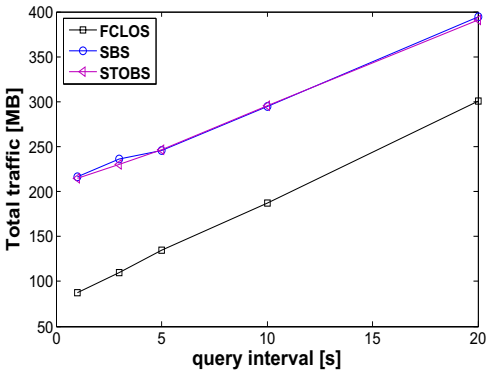

Figure 6.23: Generated traffic (Tr_{sum}) vs. query rate

Query Distribution

In this experiment, we analyze the effect of different types of query distributions on the query access time. Apart from the default workload WL_A, we test two further workloads described in Section 6.6.1. WL_B represents a workload where roll-ups or drill-downs occur more frequently than in WL_A. Exploiting their local resources, all approaches perform better, exhibiting slightly reduced access time, as shown in Fig. 6.21. WL_c represents a workload where roll-ups or drill-downs never occur. Although this is practically unrealistic, we can conclude that although there is an increase in access time, this is relatively low. All approaches are sufficiently resilient as far as query distribution is concerned. This confirms the usefulness of query mapping. *FCLOS* retains its advantage.

Query Rate

One of the requirements defined in Section 6.2 is that the system can efficiently operate under relatively low and high incoming loads. The incoming load is directly related to the query rate. After receiving an answer, clients issue a new query after a uniformly distributed time interval of $[\frac{t}{2}, t]$s. This uniform distribution is used to ensure that no artificial concurrent requests arrive, which would enable the server to build bigger broadcast clusters. Figure 6.22 shows the results. On the one hand, as the query interval increases, both *STOBS* and *SBS*, principally being on-demand systems, perform slightly better. On the other hand, *FCLOS* performs slightly worse because its objective of fully exploiting subsumptions is hampered by lower number of pending requests. In other words, the size of *BCL*s decreases. However, the increase is relatively low and *FCLOS* proves fairly self adaptive.

Apart from the access time, it is important to examine the amount of total generated traffic with several incoming loads in order to evaluate the scalability of *FCLOS*. Figure 6.23 depicts an unconventional behavior for all approaches. As the

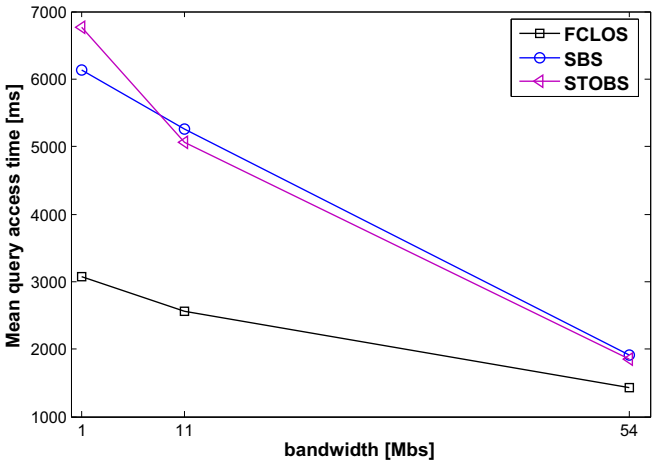

Figure 6.24: Mean query access time (T_{all}) vs. bandwidth

query interval increases, and consequently the incoming load is less concurrent, all approaches perform worse. This is justified by the fact that when requests are less concurrent, subsumptions become less possible and thus each broadcast, regardless of the mOLAP architecture, serves fewer requests.

Bandwidth

The final experiment reveals the impact of available bandwidth on mOLAP architectures. As already explained, mOLAP architectures try to cope with the bottleneck caused by wireless bandwidth. Naturally, as shown in Fig. 6.24, as bandwidth increases, all approaches exhibit enhanced performance, since T_C directly and T_Q indirectly (when the transmissions last shorter, indirectly pending requests will experience shorter queue time) decrease. Nevertheless, *FCLOS* maintains its superiority.

As seen, the performance of mOLAP systems is influenced by many factors. For reasons of completeness, in Appendix A we use parallel coordinates [75] in order to give an even more detailed insight into the effect of these factors.

6.7 Summary

This chapter presents the fundamental components of the mOLAP architecture *FCLOS*. *FCLOS* addresses both server and client architectural issues. The main architectural idea is to exploit semantic dependencies between broadcast sub-cubes in order to serve multiple clients with one transmission, even though the initiating queries are not identical. To achieve that, clients receive fact table data and perform the necessary processing.

Although the semantic dependencies between sub-cubes are given, it is unclear

how these can be exploited in the mOLAP domain. *FCLOS* addresses this issue by intelligent scheduling. Essentially, the scheduling algorithm is its fundamental component. Contrary to the well established notion of prioritizing smaller in size items, *FCLOS* does exactly the opposite. Although this might be not a good choice in general broadcast systems, it is perfectly suited for the mOLAP domain. Subsumption exploitation cannot be maximized when prioritizing smaller subcubes.

The experiments revealed the superiority of *FCLOS* on all relevant criteria: query access time, energy consumption overhead and total generated traffic are reduced by over 50%. Not only does our system significantly improve performance, it also exhibits remarkable scalability and robustness. Apart from that, it enables enhanced online and offline functionality.