Chapter 5

Distributed Multidimensional Data

The previous chapter presented related work in the area of data broadcast. Section 4.3 explained why existing systems are ill-suited for the mOLAP domain. This chapter deals with distributed and mobile data warehousing. History has shown that mobile systems began their evolution based on distributed systems. In this sense, we present related work in distributed data warehousing in Section 5.1 and in the broader area of mobile data warehousing in Section 5.2. Section 5.3 begins by defining criteria, based on which mOLAP systems can be evaluated. After that, it presents three direct competitors of *FCLOS*, and finally evaluates them according to those criteria.

5.1 Distributed Data Warehousing

Beyond the classical paradigm of fully centralized data warehousing, many companies nowadays decide to build smaller, flexible data marts dedicated to specific business areas. In order to use cross-functional analysis, there are two options. The first one is to create again a centralized DW for only cross-functional summary data. The other one is to integrate the data marts into a common conceptual schema and therefore create a distributed DW. Issues relevant to web warehousing are clearly beyond the scope of this thesis. [64, 29] provide a good insight in that area.

5.1.1 Distributed Storage and OLAP

This section briefly presents the most representative approaches in the area of distributed data warehousing. *CUBESTAR* [17] is a distributed OLAP system, which focuses on optimization issues concerning aggregation management. It employs dynamic and partial aggregation in order to improve query performance. The proposed algorithms decide which aggregate combinations yield the highest

benefit and thus are the best to be materialized.

Another distributed OLAP system is *DWS-AQA* [27]. Its goal is to use the processing and disk capacity normally available in large workstation networks to implement a DW with reduced infrastructure cost. As DWs typically share computers also being used for other purposes, most of the times only a fraction of the computers will be able to execute the partial queries in time. The approximated answers estimated from partial results have a very small error for most of the plausible scenarios.

The authors of [105] assume that the DW is distributed over a network, and that not all OLAP tasks are supported at all network locations. This implies that only some fragments have to be locally available. A heuristic cost optimization approach is suggested for setting up local DW fragments. The underlying cost model integrates query and maintenance costs. The model shares many characteristics with respective relational database models.

DSQoS [18] deals with the issue of quality of service, in the context of the accuracy/speedup tradeoff. For a given query pattern, it determines the required summary size to guarantee the accuracy targets, and then dynamically selects a set of summaries, distributed in various nodes, which can ensure the QoS constraints.

The system proposed in [19] presents sampling-based techniques for approximate answering of ad-hoc aggregation queries in Peer to Peer (P2P) databases. Given an aggregation query and a desired error bound at a sink peer, it minimizes the cost of an approximate answer that satisfies the error bound.

[45] focuses on data integration in distributed OLAP. The execution of queries that need extraction of information from more than one schema is facilitated with appropriate metadata mappings and query rewriting. [49] deals with aggregate computations in distributed environments, without specifically focusing on OLAP applications. The developed framework focuses on scalability issues.

More recently, distributed data warehousing applications have been examined in grid environments [163, 90]. It is shown how a grid computing infrastructure can be used to store and manage computational expensive data aggregations and to answer OLAP queries in a fully distributed manner.

5.1.2 Caching

Caching of multidimensional data has attracted a lot of attention. *PeerOLAP* [80] is a distributed caching system for OLAP results. In typical client-server architectures, isolated remote clients access DWs and maintain previous results in their local caches. By sharing the contents of the individual caches, *PeerOLAP* constructs a large virtual cache that can benefit all peers. The system is fully distributed and highly scalable as there is no centralized administration point and no central catalog.

In [44], the authors perform a regular decomposition of the multidimensional space into *chunks*, which constitute the smallest piece of cached information. When a query is asked, the system computes the set of chunks required to answer it, and

splits it into two subsets based on whether they are cached or not. To answer the query, the system will request the missing chunks from the DW. Only chunks at the same aggregation level as the query are considered, i.e., no aggregation is performed on the cached results; this option, however, is exploited in an extension of their work [43].

Watchman [135] is a semantic cache manager, which caches the results of the query together with the query string. Subsequent queries can be answered by the cached data, if there is an exact match on their query strings. The authors present admission and replacement algorithms that explicitly consider retrieved set sizes and execution cost of the associated queries in order to minimize the query response time.

Dynamat [85] is another OLAP cache manager, which stores fragments instead of arbitrarily shaped query results. Fragments are aggregate query results in a finer granularity than views, since they may include equality selections on some dimensions. They may be further aggregated to answer more general queries, but the data from multiple fragments cannot be combined.

5.2 Mobile Data Warehousing

The research community has rather sporadically dealt with the area of mobile data warehousing. [150] copes with disconnections in hierarchical DWs, discussing a variety of architectures for mobile views, from proxy based to non-proxy based systems. The concept of a mobile source is introduced. Architectural alternatives ranging from the use of proxies for mobile systems to architectures where every source and the DW can be mobile are discussed.

In [69], an intelligent cache mechanism for mobile data warehousing is proposed. In order to prefetch appropriate data in the cache, the system employs a query mechanism with cache management and a prediction model. The prediction model uses data mining techniques to determine the query pattern from a record of previous queries.

Hand-OLAP [39] is a system specifically designed for providing OLAP functionality to users of mobile devices. This proposal focuses mainly on the drawbacks of mobile devices, with emphasis in the small storage space and the frequent disconnections. To cope with this issue, this approach focuses on a solution for storing locally, in the mobile device, a compressed and highly summarized view of the data that can be more efficiently transmitted from the OLAP server than the original ones. However, in spite of identifying fundamental mOLAP issues, *Hand-OLAP* fails to provide a really applicable solution. The main problem is that the compression comes at the cost of approximation. A further drawback is that the technique applies for 2-dimensional views only.

Advanced OLAP visualization techniques, targeting devices with small screens is the focus in [109]. The authors introduce *CPM*, a presentational model for OLAP screens. The fundamental idea is to separate the logical part of a data cube

computation from the presentational part of the client tool. *CPM* is mapped on the *Table Lens*, which is an advanced visualization technique from the HCI area, particularly tailored for cross-tab reports. *CubeView*, using *CPM* and *Table Lens*, is a pilot academic platform enabling OLAP visualization both for desktops and mobile devices.

5.3 Mobile OLAP Architectures

This section presents three systems that are directly comparable with *FCLOS*, and can be regarded as state of the art in the mOLAP domain. The evaluation of *FCLOS* in the following chapters is performed against these three systems. Before presenting these systems, we describe the criteria, based on which, Section 5.3.5 compares them.

It is important to underline that the entire concept of mOLAP systems is founded on providing offline functionality. This is crucial when considering that portable devices are not permanently connected to a network. Therefore, clients are aware of the schema metadata, can consequently locally store received data and perform local processing in order to enable subsequent analysis. Naturally, if offline functionality is not a requirement, mobile clients can act as web clients, letting the server perform any necessary computations and just receive the (visualized) results. In the later case, complex mOLAP architectures are not necessary.

5.3.1 Criteria

We choose the following quality criteria, based on which existing mOLAP architectures can be evaluated:

Performance

In the mOLAP domain, typical performance hard metrics are query access time, energy consumption overhead for the mobile client, and generated traffic by the server. Usually, these metrics are related to each other, although some tradeoff may arise.

Offline Functionality

This is a critical criterion, since the entire concept of mOLAP is founded on it. It refers to the mobile client's functionality in case of network disconnections. In other words, it refers to the analysis capabilities of client relying exclusively on its local data.

Scalability

This measures the extent to which an architecture can handle growing number of incoming requests. This might be the effect of growing client population or of

increasing request rate.

Self Adaptiveness

Wireless broadcast is usually designed under specific workload assumptions. However, workloads are not static. This criterion evaluates the degree in which an architecture can adapt itself to workload changes.

Load Balancing

This refers not only to the degree in which the computational load is distributed across server and clients, but to the available ways to achieve that as well.

Complexity

This measures the computational complexity on server and clients. The server's complexity is mainly related to the employed scheduling algorithms. The client's complexity is related to the necessary local processing.

Fairness

This criterion is related to the stretch metric, as described in Section 3.2.2. The stretch of a job translates more directly to user-perceived performance. Intuitively, clients with larger jobs should be expected to wait in the system longer than those with smaller requests.

Maintenance Overhead

This encompasses the overhead of reconfiguring the architecture and its parameters either under different network and data settings or in order to optimize specific scenarios.

Application Generality

It represents the genericity of the architecture. In our context, it indicates if the architecture can be used in application domains other than mOLAP.

Extendability/Modularity

This criterion represents the ability to replace or extend specific modules of the architecture without influencing its entire concept.

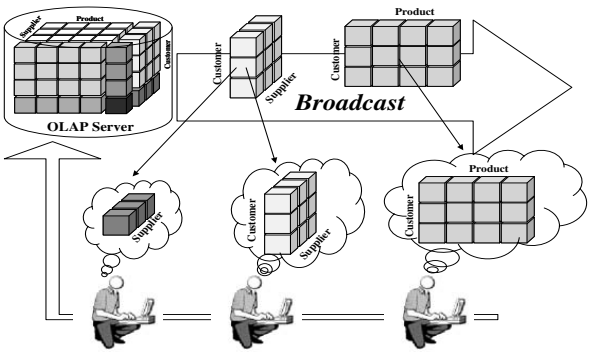

Figure 5.1: STOBS OLAP System (Source: [140])

5.3.2 STOBS

In [140] and subsequently in [139], the authors present an on-demand scheduling algorithm for mOLAP called Summary Tables On-Demand Broadcast Scheduler (STOBS). Assuming a traditional OLAP server basic functionality, the authors argue that an on-demand broadcast architecture as shown in Fig. 5.1 is the most suitable for supporting mOLAP query processing.

STOBS is an approach explicitly dealing with dissemination of multidimensional information, where scheduling decisions take into consideration additional parameters such as energy consumption. It exploits derivation among OLAP STs and tries to maximize the aggregated data sharing between mobile users. An optimizer is used to control the tradeoff between experienced access time and energy consumption overhead.

STOBS consists of two components. The first component is the prioritizing function based on the popular queue metric $\frac{R \times W}{S}$, as described in Section 4.1.1, where R is the number of requests for a specific sub-cube, W is a factor computed by the time a request has already waited in the queue, and S is the size of the sub-cube.

The idea is quite straightforward. Initially, the $R \times W/S$ metric is computed for each element of the queue and then the sub-cube j^* with the maximum metric value is selected for transmission:

$$\forall j \in Queue \quad K_j = \frac{R_j \times W_j}{S_j}, \quad j^* := arg\ max(K_j) \tag{5.1}$$

The second component controls the degree of flexibility when trying to derive subsumptions. This is done by the a-optimizer. BCL is the group or cluster of requests that are going to be served by the broadcast. D_{j^*} and D_i stand for the

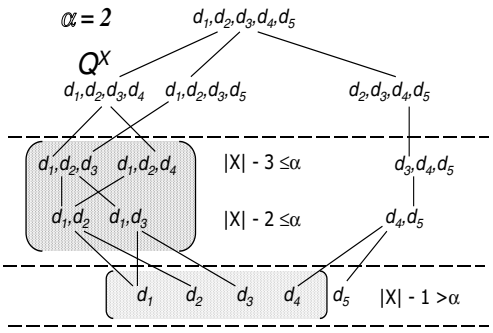

Figure 5.2: STOBS: Flexibility in DCL (Source: [140])

dimensionality of sub-cubes j and i, respectively.

$$\forall i \in Queue : i \neq v \ , \ if \ \ j^* \succeq i \ and \ D_{j^*} - D_i \leq a \ \ add \ \ i \ \ to \ \ BCL$$
$$where \ 0 \leq a \leq D$$

When $a = 0$ there is no flexibility in using STs and the client access is restricted to exact match. In this case, *STOBS-0* is equivalent to $\frac{R \times W}{S}$. At $a = D$, it is the case of extreme flexibility in which a client can use any subsuming matching table.

Assume that the search lattice nodes shown in Fig. 5.2 are the tables for which there exists at least one request. Also, assume that Q^X is a request to the 4-dimensional table T^X:(d_1, d_2, d_3, d_4) that is selected to be broadcast next. All shaded tables can be derived from T^X. These are (d_1), (d_2), (d_3), (d_4), (d_1, d_2), (d_1, d_3), (d_1, d_2, d_3) and (d_1, d_2, d_4). However, if we assume that $a = 2$, then clients' requests for tables (d_1, d_2), (d_1, d_3), (d_1, d_2, d_3) and (d_1, d_2, d_4) will be satisfied by T^X and hence the requests for these tables will be discarded, whereas the requests for tables (d_1), (d_2), (d_3) and (d_4) will remain in the queue.

STOBS arguably exhibits a superior performance than a point-to-point architecture. The scalability of the system is also satisfactory, since increasing workloads incur a relatively low loss of performance. However, there is a tradeoff between the optimization of access time and energy consumption. Although this tradeoff can be controlled by the a-optimizer, this inevitably causes an additional maintenance overhead. The server's computational complexity is relatively low because the scheduling algorithm just needs to compute the value for the employed metric. Then, after having decided the element to be transmitted, the discovery of subsumptions needs just a single scan over the queue.

The a-optimizer provides a fairly satisfying flexibility. Despite that, the algorithm heavily relies on a metric that does not at all take into account the nature of the transmitted data, which is an OLAP ST. In addition to that, the two components (prioritizing and optimizing) are completely independent. The clustering

of requests succeeds, only after the sub-cube to be transmitted has been already selected. As a result, clustering is rather loose. This is a consequence of the fact that smaller in size sub-cubes have generally increased priority. When smaller sub-cubes are selected from the first component to be transmitted, the possibility of creating bigger clusters in the second diminishes.

The scheduling policy of *STOBS* has further consequences. Since subsumptions are considered in a second optimizing step, a lot of optimization space has already been wasted by the prioritizing function, before subsumptions can be examined. Thus, received datasets are not optimal for offline usage. Finally, the resulting system is not extendable. *STOBS* transmits STs. A selection of another physical structure will directly and unpredictably influence the scheduling decisions, since the used metric ($\frac{R \times W}{S}$) includes the size of the data items, which depends on the physical structure.

A fundamental problem of *STOBS* is that it targets a very limited set of queries. As explained in Section 2.6, selections or clauses might not necessarily be posed on the values of the fact table but possibly on values or attributes of the dimension tables. *STOBS* broadcasts only fact tables and consequently fails to address this issue.

5.3.3 SBS

The Subsumption-Based Scheduler (SBS) [141] is an approach very similar to *STOBS*. *SBS* consists of two similar components. Instead of $\frac{R \times W}{S}$, the prioritizing function is based on the *LTSF* algorithm, described in Section 4.1.1. In *LTSF*, the data item that has the largest total current stretch, i.e., the sum of the current stretches of all R pending requests for the item j, is chosen for broadcast. Particularly, the current stretch of a pending request is the ratio of the time the request has been in the system thus far to its service time.

$$\forall j \in Queue \quad K_j = \frac{\sum_{r=1}^{R} W_{r,j}}{ST_j}, \quad j^* := arg\ max(K_j)$$

In addition to that, the a-optimizer is defined in a different way, considering the sub-cubes' size and not the dimensionality.

$$\forall i \in Queue : i \neq j^*\ ,\ if\ \ j^* \succeq i\ and\ \frac{|S_{j^*}| - |S_i|}{|S_{j^*}|} \leq a\ \ add\ \ i\ \ to\ \ BCL$$
$$where\ a \in [0,1]$$

When $a = 0$ there is no flexibility in using STs and the client access is restricted to exact match. In this case, *SBS-0* is equivalent to *LTSF*. At $a = 1$, it is the case of extreme flexibility in which a client can use any subsuming matching table.

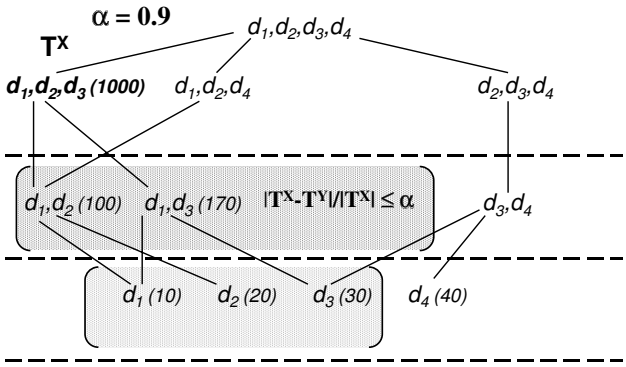

Figure 5.3: SBS: Flexibility in DCL (Source: [141])

As an example, consider the partial search lattice shown in Fig. 5.3, in which nodes are STs and the number between braces is the table cardinality in units of size. Assume that the search lattice nodes shown are the tables for which there exist at least one request and $a = 0.9$. Also, assume that the 3-dimension table T^X (d_1, d_2, d_3) is selected for broadcast. Then, clients' requests for tables (d_1, d_2) and (d_1, d_3) will be satisfied by T^X. While clients who requested tables (d_1), (d_2) and (d_3) will have to wait for the next broadcast cycles.

It should be clear by now that *SBS* is a very similar approach to *STOBS*. This is also confirmed by our experimental evaluation presented in following chapters. Hence, it exhibits the same advantages and disadvantages as analyzed in the previous section. Although the usage of the *LTSF* algorithm for the prioritizing function results in an increased user-perceived fairness, the overall performance hardly improves. On the contrary, the definition of the *a*-optimizer does no longer depend on the schema metadata, which is independent of the physical structure. Thus, the integration of other physical structures will have even more unpredictable effects, than in *STOBS*.

5.3.4 DV-ES

DV-ES [142] is an extension of the *SBS* architecture. While the scheduling process remains almost intact (apart from a negligible modification of the *a*-optimizer), the difference lies in the transmitted physical structure. *SBS* transmits STs, which are the simplest data cube physical structure. STs are unindexed relations which consist of all the tuples of a corresponding fact table. *DV-ES* exploits the *Dwarf* technology [149]. Dwarf is a highly compressed structure for computing, storing and querying data cubes. The intuition is to transmit the structure which occupies less space in order to reduce generated traffic. Due to the fact that depending on the sub-cube, the corresponding Dwarf or ST might occupy less space, *DV-ES* employs a hybrid selection. Chapter 8 discusses this approach and issues of finding the appropriate physical structure for transmission in much more detail. In the context of this chapter, note that *DV-ES* hybrid scheduling function incurs a com-

Table 5.1: Evaluation of mOLAP architectures

	Point2Point	**STOBS**	**SBS**	**DV-ES**
Performance	- -	0	0	0
Offline functionality	- -	0	0	0
Scalability	- -	0	0	0
Self adaptiveness	-	- -	- -	- -
Load balancing	- -	0	0	0
Complexity	++	-	- -	- -
Fairness	+	-	+	+
Maintenance overhead	++	-	- -	- -
Application generality	++	- -	- -	- -
Extendability/Modularity	- -	-	-	- -

plexity and maintenance overhead, without providing any substantial performance improvement.

5.3.5 Evaluation of mOLAP Architectures

Having presented the state of the art of mOLAP architectures, we can now compare them according to the criteria defined in Section 5.3.1. The criteria are weighed using the symbols of Table 4.1. The results are shown in Table 5.1

Point to point architectures exhibit unlimited application generality, so that any application can be supported without domain specific considerations. Moreover, complexity and maintenance overhead are kept at a minimum. However, these systems fail to provide acceptable performance or scalability. *STOBS* and *SBS* are, as already underlined, almost identical approaches. They exhibit fair performance and scalability, without adding too much complexity. *DV-ES* fails to provide any substantial improvement against its ancestor *SBS*, despite the additional complexity.

5.4 Summary

In the research area of mOLAP, it was soon realized that point to point architectures are unable to provide acceptable performance or scalability. Therefore, existing mOLAP architectures employ broadcast to confront these issues. However, the usage of general broadcast systems for mOLAP is not straightforward, since such systems ignore data semantics and OLAP end user behavior, while assuming thin clients and not considering offline functionality. Moreover, multidimensional data cubes are items order of magnitude bigger than the ones typically assumed.

Existing mOLAP systems are mere extensions of general, on-demand broadcast systems, adding a flavor of subsumption. Despite the considerable performance

optimization compared to point to point architectures (although a tradeoff between the optimization of access time and energy consumption exists), they still remain on-demand architectures, which are notoriously not scalable. While they can efficiently serve a limited number of end users, they are definitely unsuitable as this number increases. In addition to that, subsumption exploitation is designed as an extra component that operates after the determination of the transmission schedule. Since sub-cube semantics are not involved in the scheduling process, the subsumption exploitation is loose and clearly sub-optimal.