Experimental Evaluation of a Dynamic Cubing System: Workflow, Metrics, and Prototype

Anne Tchounikine*, Maryvonne Miquel*,
Usman Ahmed*

*LIRIS-CNRS UMR 5205, INSA-Université de Lyon, France
firtsname.lastname@liris.cnrs.fr

Abstract. This article originates in the efforts we made in the prototyping stage of a project conducted in the field of new agile BI applications. In this earlier work, we defined a model and algorithms for handling dynamic cubes that are updated as single fact comes in. The proposal includes the definition of a tree structure, called DyTree, used to store these cubes. A comprehensive experimental evaluation was conducted to observe the behavior of the proposal under some varying circumstances or settings, to help in understanding its working and to provide feedback to improve the solution. In this paper, we describe the workflow which has been used for this experimental evaluation. Performance metrics and behavioral metrics are defined as output of the experimental evaluation workflow. Different types of data sets and parameters for algorithms are the input used to configure the experiments. Using this environment and the prototype, some examples of experimental studies are presented. This shows how different scenarios can be simply constructed and how the experimental results can be used to understand the behavior of our proposal and to evaluate performance.

1 Motivation and Positioning

This article addresses the issue of experimental evaluations related to a research work in OLAP. It originates in the efforts we made in the prototyping stage of a project conducted in the field of new agile BI applications operating in fast evolving environnement. In this work, a solution is discussed in order to provide better data freshness and reduce analysis latency (Ahmed et al. (2010)). This solution allows on-the-fly insertions of facts and members by means of frequent atomic insertions, thus leading to fast aggregate updates. We provided the definition of a Dynamic Cube based on a multidimensional un-ordered and multi-level data space, enabling its evolution. A tree structure incrementally stores detailed data and aggregates for the densest regions of the data space thanks to a split strategy that promotes refinement of aggregates at increasing lower level. This proposal was implemented in a prototype. This prototype consists of a suite of tools starting from facts loading to OLAP navigation in the dynamic cube. This prototype aims at functional testing and helps us to demonstrate the feasibility of the solution. However, it is also a means for evaluating the performance of the proposal and studying its behavior in time. In academia, new models or algorithms always gain to be formally demonstrated and qualified. In a complementary way, experimental evaluation allows to observe their behavior under some varying circumstances or settings, helps in understanding their working and provides feedback to improve the solution and allows to experiment some formal postulates. This requires the settings of run-time environments to be tightly controlled. The collected results must be relevant and interpretable, ie. the observation of the system requires the definition of a set of metrics representing its state and its evolution: know what has changed, and how this change can be decisive for the system's behavior. In addition to functional testing, two types of experimental evaluation can be carried: one can vary the run-time settings (data set and/or scenario and/or parameters of algorithms) and compare obtained results in order to tune the system or, more simply, to understand its behavior. Alternately, one can compare the results obtained by competing related systems under a same execution context in order to conduct a comparative study or benchmark. Benchmarking is often more difficult to achieve: it needs to have good insights of actual implementation of competing solutions in order to breed conditions required for fair comparison. Moreover, it is also sometimes difficult to make sense of comparison studies, the models and algorithms often being designed to meet issues in very specific contexts (data of a specific type: multimedia, stream, graphs, etc. or applications in a specific context: real-time, mobility, web etc). In the specific field of OLAP and data warehousing, multidimensional data sets generators (as TPC-D, TPC-H) are proposed paired with workflows offering scenarios of ETL or OLAP queries. Common metrics are memory usage and execution times for loading, cleaning, aggregation, response for OLAP queries, etc.

In this article we try to highlight how the use of a rigorous approach can promote experimental evaluations. Therefore we advocate our experience of a workflow that enables to: (1) list and characterize input elements that determine the execution context and are determinant for the test performed, (2) formally define metrics that are relevant for the evaluation of results, (3) define a worfklow which allows the design of scenarios, tuning of inputs, collection and interpretation the outputs.

This approach is used while conducting our research project on real-time data warehousing. We believe this experience can contribute to show how experimental evaluations can gain in being designed and adopted from the very beginning and maintained throughout a project. The remainder of this paper is organized as follows: Section 2 briefly motivates our work on Dynamic Cubing and summarizes our earlier contribution in this field. Section 3 describes the workflow of the experimental evaluations performed on our solutions for dynamic cubing, and evaluation metrics used to carried the analysis of experiments. Section 4 lists and defines the input parameters used to customize the settings of an experimental evaluation and Section 5 gives samples of experiment tracks and shows how the input parameters are set to reproduce use cases of test scenario, and how metrics are used to analyze them. Section 6 discusses related work and finally Section 7 concludes.

2 Experimental Evaluation of the DyTree

2.1 Glimpse of our Earlier Work on Dynamic Cubing

Data warehouses are increasingly desired in time critical applications such as supervision applications, fraud/threat detection, energy management, operational intelligence etc. In these

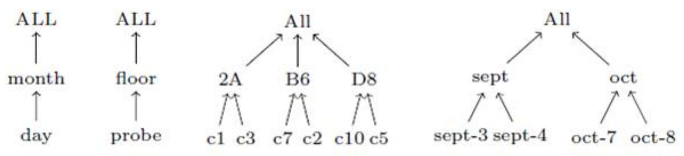


FIG. 1 – *Hierarchies for Building Temperature*

applications, the facts are expected to to be inserted on-the-fly, by means of frequent atomic insertions leading to the requirement of fast aggregate updates in order to maintain the hypercube up-to-date. Moreover, if new dimension members are frequently inserted at run time, then it is clear that the re-arrangement for ordering data is not feasible, leading to a need for fast and dynamic cubing. The special nature of a dynamic cube makes inefficient the usual strategies and data structures. In this article, we will not expand on the topic of dynamic or real-time cubing. We will merely give a brief overview of our earlier work in the followings. The detailed proposal is available in Ahmed et al. (2014). Our earlier contribution in the field of dynamic cubing includes the following propositions: a multidimensional data model is defined that operates in a data space defined by non-ordered hierarchical dimensions. This data space is extensible, i.e. dynamically created at facts arrival. Let us take a very simplified use case of a data warehouse for Building Temperature Control, with two dimensions, time (day < month) and sensor location (probe < floor) (Figure 1). The measure is the temperature with average as aggregation function.

Initially, the multidimensional data space and its dimension axes are empty. When the first temperature is recorded, then its coordinates (day, probe) are plotted in first position on the axes and the point appears in the multidimensional space. Thereafter gradually as the facts come in, their coordinates are added to the axes and the points are placed in the multidimensional space. The data grouping structure is inspired by the R-Tree although operating in a hierarchical and unordered data space. This grouping structure, called Minimum Bounding Spaces (MBS) is defined by the cartesian product of sets of members lying at same hierarchical level. Examples of MBS are illustrated Figure 2 (M1, ..M5). A MBS represents a section of a cuboid. We distinguish the set of points *enclosed* in a MBS from the points *covered* by the MBS and note:

- $|M_\triangle| = \prod_{i=1}^{n} |E_i|$ the number of points covered by the MBS M_Δ,
- $||M_\triangle|| = |\triangle|$ the number of points enclosed in the MBS M_Δ.

A tree structure, named DyTree, is designed to store MBS with associated aggregated measure value (see Figure 2, in this tree |M5| = 3 while ||M5|| = 9). The tree is dynamically built and grows classically by means splitting its nodes. When the first fact arrives, the Dytree stores this fact in its only leaf, and a MBS (all_{D_1},..., all_{Dn}) with associated measures in its root. As other facts arrive, their coordinates are added to the axes, next to each other, and meanwhile added to the tree. When a non leaf node of the tree is full with respect to pre-settled maximum capacity, then it is split. The originality of the approach is that a split generates two new nodes storing aggregates at same or lower level dimension. These new aggregates in turn are plotted in the multidimensional space. The split dimension of the aggregates is chosen based on the current number of facts indexed or density: therefore, in our use case, if temperature measures occur mostly in the same location, location aggregates will be more and more detailed. As a result, the multidimensional space is un-ordered and contains all members whatever their level on its axes ; the DyTree is a multidimensional cube with materialized aggregates for its densest regions. Algorithms are provided to handle the DyTree (split of nodes, aggregation, growth of the tree) and for querying the tree (range, point and group-by queries).

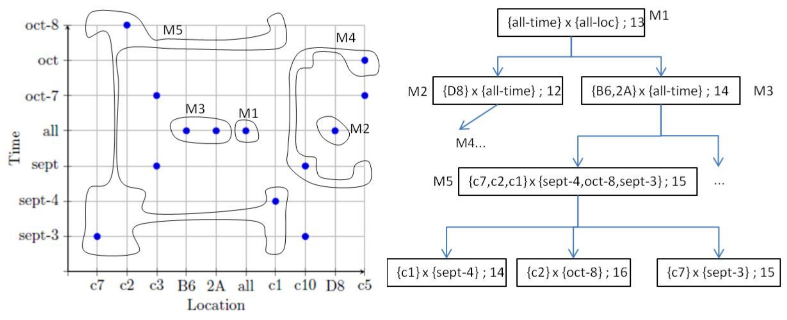


FIG. 2 – *Data Space and associated DyTree*

Two input parameters are used by the construction algorithm of the DyTree. These are directory node capacity and overlap limit. Our implementation of the DyTree uses the following storage strategy: internal nodes that hold the calculated aggregates are stored in main memory while the leaves holding the detailed facts are stored on disk.

2.2 Goals of the Evaluation

The objective is to experimentally assess the efficiency and effectiveness of the DyTree based solution for cubing. For this purpose, a careful experimental process is proposed in order to measure and analyze the efficiency of the structure of DyTree and its associated algorithms. These experiments let us investigate the answers to the following questions:

– **Question 1**: Are the DyTree and associated algorithms efficient ?
This question addresses the performance of the solution. Our criteria for performance evaluation are based on both execution time and memory space usage. Here, execution time corresponds to the insertion time of a new fact, which includes aggregation and eventual splits, and response time of an OLAP query. The objective is of course to minimize both times. As for execution time, efforts are made to decrease the memory space usage for storing the Dytree. Let us recall that the Dytree is the whole hypercube.

– **Question 2**: How does the DyTree behave under different circumstances ?
This question addresses the effectiveness of the solution by observing the behavior of the DyTree. The behavior can be analyzed by observing the shape and contents of the tree and its nodes, which may change with the characteristics of the data set or Dytree maintenance algorithms configuration. This may allow us to observe our postulates, such as dense regions holds more aggregates (i.e. MBS) than sparse ones.

In order to answer to these two questions, we have to define performance metrics (Question 1) and behavior metrics (Question 2). The workflow which is proposed allows to define and control the runtime environments and to collect the evaluation metrics.

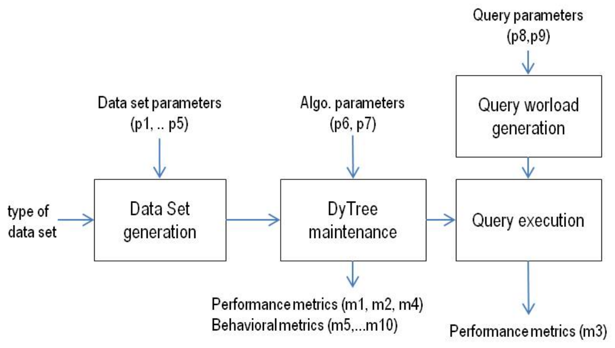


FIG. 3 – *Workflow of experimental evaluation*

3 Workflow for Experimental Evaluation

The experimental evaluation workflow consists in 3 phases. A set of parameters are defined in order to characterize the inputs: data sets, query workloads, and algorithms configuration. Then, we define a set of metrics to capture and qualify the outputs. These are performance and behavioral metrics that qualify either the process of DyTree maintenance or query execution. At last, experiments are performed to study:

– performance and behavioral metrics obtained while varying characteristics of the data sets,
– performance and behavioral metrics obtained while varying algorithms configuration,
– performance metrics obtained using our solution against other methods.

Flow diagram presented in Figure 3 summarizes the overall process of experimental evaluation. For every experiment, a data set is used that is generated on the basis of a use case schema and various parameters detailed in the next section. Query workloads in turns are generated according to the generated data set, and desired types of queries. Finally parameters for algorithms configuration are set, and this ends the parametrization stage. Then the simulation stage begins and facts are loaded one by one while the tree is constructed and updated; Meanwhile, queries are triggered against the tree. Metrics are collected all along the process for in line and/or later analysis.

The experimental evaluation is performed thanks to various tools that are plugged around the DyTree proptotype as shown in Figure 4. The Data Generator Tool computes data sets and query workloads based on input parameters. A Fact Injector is responsible for atomic loading of facts in the DyTree Engine, under a user-defined periodicity. The DyTree Engine handles the dynamic DyTree, it performs algorithms for insertion into the tree, split, aggregation and storage of the nodes. The OLAP Engine supplies query interface for OLAP navigation and/or query workloads and uses the DyTree as entry and a navigation GUI. The DyTree Monitoring system provides features that allow the storage and the analysis of output metrics. These features comprise: (1) run-time visualization of performance and behavioral metrics collected while building the DyTree, (2) run-time visualization of the Dytree, (3) off-line 3D visualization of aggregates (i.e. nodes of the DyTree), (4) off-line plotting of sets of metrics.

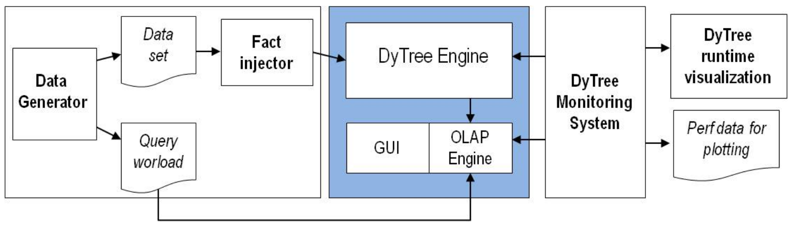
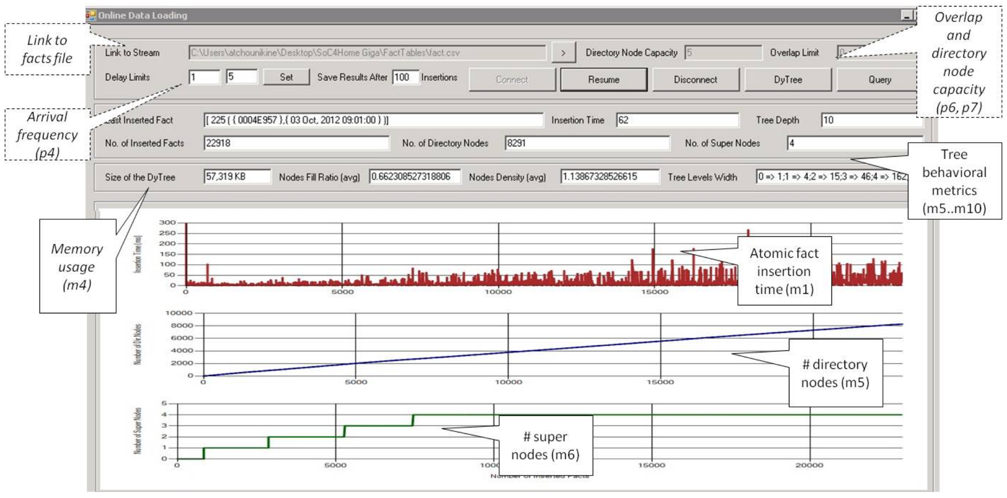


FIG. 4 – *Architecture of the suite*


Online Data Loading

Link to Stream: C:\Users\atchounikine\Desktop\SoC4Home Giga\FactTables\fact.csv
Directory Node Capacity: 5
Overlap Limit: 0
Delay Limits: 1 5
Save Results After 100 Insertions
Last Inserted Fact: [225 ({ 0004E957 },{ 03 Oct, 2012 09:01:00 })]
Insertion Time: 62
Tree Depth: 10
No. of Inserted Facts: 22918
No. of Directory Nodes: 8291
No. of Super Nodes: 4
Size of the DyTree: 57,319 KB
Nodes Fill Ratio (avg): 0.662308527318806
Nodes Density (avg): 1.13867328526615
Tree Levels Width: 0 => 1;1 => 4;2 => 15;3 => 46;4 => 162

FIG. 5 – *Screenshot of the DyTree Monitoring System*

Outputs of the experimental evaluation workflow are metrics that can be grouped in two categories, performance metrics and behavioral metrics. These metrics can be visualized at run-time thanks to the monitoring tool GUI (Figure 5) and/or stored for later studies and plotting.

3.1 Performance Metrics

Performance metrics are used to evaluate and analyze the performance efficiency of the solution. They are numbered `mi` in the followings ; they are summarized in Table 2 ; the visual interface for some of them can be seen on the screenshot (Figures 5, 7 and 8).

Execution Times `Atomic fact insertion time (m1)` is the time needed to insert an individual fact in an existing DyTree. This time is calculated as the average time recorded for a pre-settled number of atomic insertions. `Tree construction time (m2)` is the time to build a tree from scratch, i.e. the total insertion time needed to insert a pre-fixed number of facts. Finally, we record the `Query Response Time` (m3) of OLAP queries.

Memory Usage Let us recall that the Dytree is the whole hypercube with partial materialization. The detailed data is in the data nodes stored on disk, while the aggregated data is in directory nodes stored in memory. This strategy could affect the performance evaluation if the growth of the tree becomes out of control. The `memory usage (m4)` is the effect of the directory nodes present in the DyTree witch are the result of nodes splitting.

3.2 Behavioral Metrics

The behavioral metrics are defined to characterize the different aspects of the DyTree that are found to be useful in understanding its behavior. These metrics are grouped into two categories: metrics for tree and metrics for nodes of the tree.

Metrics for Tree Height, width and number of nodes that directly impact the performance of insertion and query algorithms are classicaly observed.

`Number of directory nodes (m5)` is an indicator of number of splits that occurred. It monitors the number of materialized aggregates and gives an idea about the evolution of the tree. Super nodes are internal nodes with a non limited capacity. They are created when no suitable split dimension is found at the time of a directory node's splitting. A larger `number of super nodes (m6)` may indicate the wastage of memory and/or disk space and could cause the degradation of the efficiency of the search and insert algorithms. The height and width of the DyTree are indicators of the level of aggregates created. In DyTree, the `height of the tree(m7)` grows when the root node is split. The higher the tree is, the more aggregates have been materialized, favoring OLAP queries but increasing memory usage. We also record `average value of the widths (m8)` at each level in the tree. A DyTree with greater value of average tree levels width indexes a larger number of nodes at the same level and penalizes search. A lower value of average tree levels width, therefore, is expected to result in better performance of the algorithms.

Metrics for Nodes Each directory node contains an aggregate, and indexes aggregates at lower level. Facts are stored in the leaves. Both the number of aggregates indexed by a directory node (i.e. children) , and the number of leaves indexed by it (i.e. descendants at lower level) are measured.

– `Node Fill Ratio (m9)`: The fill ratio of a node determines the extent to which the node is filled. Let $node$ be a directory node of a DyTree:

$$fillRatio(node) = \frac{number\ of\ children(node)}{directory\ node\ capacity}$$

In our experiments, we record the average value of nodes fill ratio of all the directory nodes in the DyTree. A lower value of average nodes fill ratio means that the nodes are not filled up to the capacity and hence indicates the wastage of memory.

– `Density (m10)`: The density of a node measures the sparsity of its MBS M. For its calculation, we use drill-down operator applied on each element of the edge of M. Let be N the MBS obtained by a drill down on the elements of each edge of M, at the lower level of detail, then we can define:

$$volume(M) = |\, N \,| \text{ and } density(node) = \frac{||\, N \,||}{volume(M)}$$

A lower value of a node's density means that the node covers a large volume without indexing a sufficient number of data nodes under it. It monitors the accuracy of an aggregate.

4 Tailorization of Run-time Environments

In order to test and analyze the performance and behavior of our solution, we need to collect the metrics previously described under different circumstances, i.e. when the run-time environment varies. The tailorization of run-time environments is done by means of input parameters, among them the parameters that drive the generation of multidimensional data sets and query workloads, as well as the algorithms configuration. We first describe the different data sets used, and then come into the details of the parametrization.

4.1 Multidimensional Data Sets

Our multidimensional data sets are made of a multidimensional schema, instances of dimensions stored in dimension tables, and one fact table. Our first data set is a standardized benchmark and benefits from a community recognized specification and allows fair comparisons between related work. However, using a benchmark makes it hard to reproduce use cases exposing the issues that the solution specifically addresses. Thus, we also define and use custom synthetic Data Sets in order to be able to customize the characteristics we postulate relevant for the hypothesis to test. Finally, we use a real world data set, whose data were obtained as part of a building monitoring system deployed in our lab.

Star Schema Benchmark The Star Schema Benchmark (SSB) O'Neil et al. (2009) is a modified version of the famous TPC-H benchmark TPC (2011). SSB proposes a "Sales" data set based on star schema.

Custom Synthetic Multidimensional Data Sets We propose to generate Custom Synthetic Data Sets following the conceptual schema shown Figure 6. The schema is star shaped, with one numerical measure that will lately be aggregated with function SUM. Each dimension Di comprises 2 to 5 levels Lj. Their attributes are reduced to auto-generated keys and random labels (DiLjKey and DiLjName). Schema and data generation is driven by the input parameters described in next section.

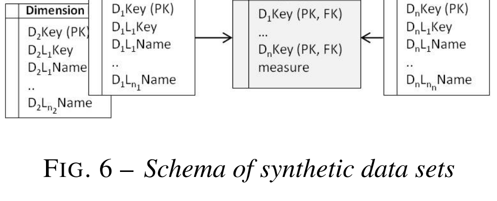


FIG. 6 – *Schema of synthetic data sets*

SoQ4Home Data Set The SoQ4Home project[1] aims at deploying a wireless network of temperature sensors in buildings subject to environmental and energetic control. These temperatures are collected through a data stream oriented web service. One aspect of this project is the warehousing of the collected data and the design of dashboards allowing energetic real-time analysis of the building. This data warehouse holds 2 dimensions, sensor and timestamp. The sensor dimension shows a geographic hierarchy used to locate the probe, from its GPS coordinates to its location in the building (room, floor etc.). Time is hierarchized in second, minute, hour etc.. The measure is, of course, the value of the temperature, aggregated with functions min, max and average. Two buildings of our university campus are equipped with such a network.

Query workloads Query workloads are generated for the SSB benchmark and our Custom Synthetic Data Sets. SoQ4Home data sets are only queried at a glance thanks to the user OLAP interface engine (Figure 4). We generate three types of queries which are usually used in OLAP analysis i.e. point queries, range queries and group-by queries. We represent a point query as a MBS whose all edges are singleton. For range queries, a random range of domain values belonging to the domain sets of the levels of each dimension is selected. The aggregate value is queried for all the points that lie in the subspace formed by the selected ranges of values for each dimension. A range query is represented by a MBS where the edges represent a range of values for each dimension. Group-by queries are generated in a similar way.

4.2 Input Parameters for Run-time Environments

Tailorization of the run-time environment is done thanks to the choice of above described data set, paired with set of input parameters. In what follows, the parameters are numbered `p`*`i`* (see Table 2) ; the visual interface for some of them can be seen in the screenshot Figure 5 ; the visual interface for the others is embedded in the Generator Tool (Figure 4).

Input Parameters for Data Sets The appliance of input parameters obviously depends on data set types (Custom Synthetic, SSB or SoQ4Home). Table 1 summarizes the possible combinations.

The first input controls the size of the data set. The size is determined thanks to the `number of dimensions (p1)` in the cube and the `number of tuples in fact table (p2)`. In our experiments we varied the number of dimensions from 10 to 30 for custom synthetic data sets. Cardinality of the fact table is particularly critical in case of dynamic data warehousing application, as for the major interest lies in determining the insertion efficiency for atomic tuple. In our experiments we let the cardinality vary from 1 to 100 millions facts. All multidimensional data sets are submitted to screw and/or sparsity. The volume of a multidimensional data space represents the number of distinct detailed data points (i.e. possible facts) that can lie in the data space. Let D_1, D_2, ... , D_n be n dimensions of a multidimensional data space S and $l_i^1, l_i^2, ..., l_i^{k-1}$, ALL_{D_i} the levels for D_i.The volume of the data space can be calculated as:

$$volume(S) = \prod_{i=1}^{n} |domain(l_i^1)|$$

1. `http://liris.cnrs.fr/socq4home/`

Input parameter	SSB	Custom synthetic data sets	SoQ4Home
# dimensions p1)	4	x (up to 30)	2
# facts (p2	x	x (up to 100M)	na
density p3	o	x	na
arrival frequency p4	x	x	na
order p5	o	x	na

x : user defined value, na = non applicable, o = not used

TAB. 1 – *Input parameters for data sets*

If $||domain(l_1^1) \times domain(l_2^1) \times ... \times domain(l_n^1)||$ is the number of existing data points at the lowest level of each dimension of S (i.e. the facts in the fact table) then the density of the data set or the S is given by:

$$density(S) = \frac{||domain(l_1^1) \times domain(l_2^1) \times ... \times domain(l_n^1)||}{volume(S)}$$

Alternatively, to volume of the fact tables, the `density (p3)` can be used to parametrize the data generation. In this experimental evaluation, by varying the density of data sets, we aim to analyze if the density of data sets affects the density of the the constructed data partitions i.e. the nodes of the DyTree and the overall performance of the DyTree. For now, our data sets are generated following uniform distribution, unless a screw is manually simulated. If not user valued, the density is calculated.

As stated before, we use a tool named Fact Injector (Figure 4) to simulate the real time arrival of one fact in the DyTree. This tool is parametrizable by a time interval in milliseconds which determines the maximum delay between the arrival of 2 facts. This value of `arrival frequency (p4)` can be modified at run-time to accelerate or slow down the progress of the loading stage thus stressing the update of materialized aggregates and growth of the tree. Lastly, remember our multidimensional model considers the dimensions as hierarchical but non ordered. However, dimensions can appear to be naturally ordered, for example, time dimension is obviously a naturally ordered dimension. Thus we want to test data sets where facts arrives following chronological order or with some percentage of tuples delayed. We expect our model to naturally gather the temporally close facts together in same node of the tree, thanks to the split algorithm that groups together members that require smallest extension of the node. We postulate that this grouping will favor group-by and range queries. Percentage of facts delayed is the parameter `order (p5)` and can vary from 5% to 100%.

Input Parameters for Algorithm Configuration Two input parameters are used to configure the construction algorithms of the DyTree. These are directory node capacity and overlap limit. `Directory node capacity(p6)` value has a great impact on both the performance of the algorithms and the tree shape. A larger capacity directory node means that a larger number of nodes are indexed sequentially under a directory node. This could degrade query performance, but on the other hand it minimizes the invocation of nodes splitting algorithm which is a costly algorithm and is called once a directory node overflows. At last, since the `overlap limit (p7)` controls the shared area of two or more MBS, it could require

	Input parameters
p1	# dimensions
p2 or p3	# facts or density
p4	arrival frequency
p5	order
p6	directory node capacity
p7	overlap
p8	type of queries

	Output metrics
m1	time for atomic fact insertion
m2	total tree construction time
m3	time for query response
m4	memory usage
m5	# directory nodes
m6	# super nodes
m7	tree height
m8	tree width
m9	node fill ratio
m10	node density

TAB. 2 – *Summary of input parameters and output metrics*

Scenario	Data Set	p1	p2	p3	p4	p5	p6	p7
Varying overlap	SSB	4	10M	calc.	na	na	15	0 to 15
Varying dir. cap.	SSB	4	10M	calc.	na	na	5 to 75	0
Scaling # facts	SSB	4	up to 10M	calc.	na	na	15	5
	synth	10	up to 100M	calc.	na	0%	15	5
	Soq4Home	2	na	na	na	na	15	5
Scaling # dim.	synth	2 to 30	up to 100M	calc.	1 to 5 ms	0%	15	5
Varying density	synth	10	calc.	0.2 to 0.6	1 to 5 ms	0%	15	5
Delayed arrival	synth	10	up to 100M	calc.	1 to 5 ms	5 to 80%	15	5

TAB. 3 – *Examples of scenarios*

searching algorithm to verify the searching criteria against less/more nodes and consequently affects the performance of both construction and querying algorithms of the DyTree.

Input parameters for Query Workloads Input parameters for the generation of query workloads, beside the target data set, are `type of queries (p8)` (range, group-by or point) and `number of queries generated by type (p9)`. All three types of queries are generated using randomly chosen aggregation level.

5 Examples of Experimental Studies

Next step is to value the input parameters so as to create run-time environments for testing our hypothesis. In the case of the DyTree, we may want for example to verify if dense regions produce more aggregates than sparse ones, or evaluate the effect of scaling the number of facts on the performance of the solution (see Table 3).

Each scenario is completed with a set of 50 queries of each types (Group-by, range, point). After the setting of input parameters, generation of data sets and query workloads, the Fact Injector tool is triggered. In the following paragraphs we show how the Monitoring System can be used to conduct analyses of test scenarios.

5.1 Run-time Monitoring

As shown in Figure 5, the suite authorizes run-time monitoring thanks to graphs that dynamically display some of the collected metrics. For example, a graph representing the time for atomic fact insertion (m1) can be seen in the top frame, just above the graph displaying the number of directory nodes (m5) and supernodes (m6). Some of the tree behavioral metrics are also displayed dynamically, for example tree height and width, fill ratio and density.

5.2 Late Analysis

Run-time collected metrics are also stored in flat files. These files then are used as input for plotting functions. Figure 7 shows the plotting obtained in a comparison study against a related work solution (Ester et al., 2000). The results for dynamic fact insertion are presented

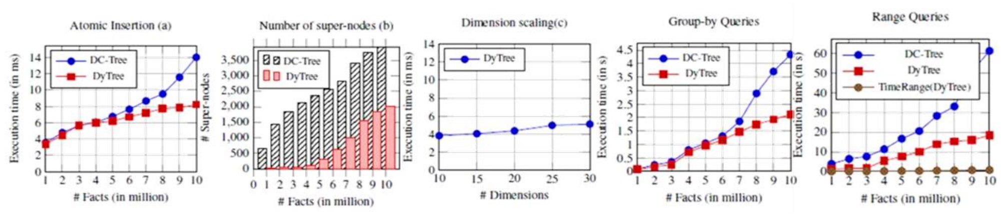
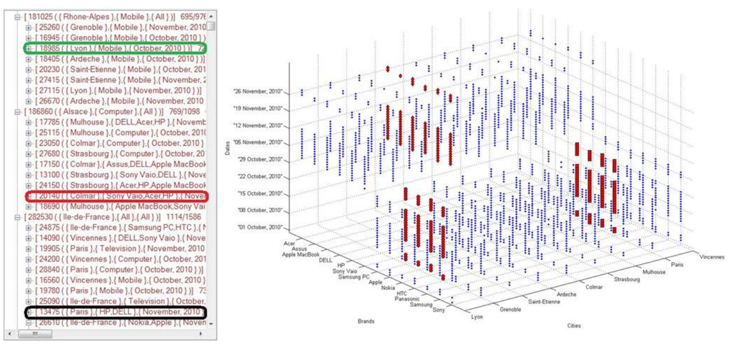


FIG. 7 – *Plotting metrics*

in figure 7. This time is calculated as the average time recorded for 100 atomic insertions in DyTree and DC-Tree storing 1, 2 up to 10 millions facts from SSB. The results in 7(a) show that as the size of data set grows, the insertion time in the DC-Tree starts to increase rapidly. The detailed analysis of the results have shown that this behavior is due to the appearance and increase in size of super nodes (see figure 7(b)). Results in 7(c) show that the performance of DyTree slightly suffer with scaling of dimensions. In figure 7(d, e), the comparison results for group-by and range queries are given. Both show improvement with DyTree solution. The grouping of temporally closed values together in same MBS allows a significant gain which is even more significant when range queries operate on ordered dimension. This is highlighten by the last graph figure 7(e) where range queries are defined solely on time dimension. Our prototype also offers a graphical user interface (see figure 8) to visualize the data in a three dimensional data space. This tool is helpful in viewing the distribution and density of data. Thanks to this GUI, the user just selects (clicks on) a node in DyTree in order to visualize its data at detailed level in the data space. He can observe if the data encapsulated in a node originates of a same cluster or is scattered in the data space. This way, he can visually verify that the DyTree naturally gathers data from dense regions and close data into directory nodes.

6 Related Work

Testing DB or DW systems has consistently been a major preoccupation in academia and well as in industry. Generation of reference data is a cornerstone of this difficult task. Different research works explore models and methods to synthesize, in a reasonable delay, large volume of data (TPC (2011)). The generation of large volumes of data according to relational scheme including referential integrity constraints poses specific issues including scalability, robustness or parallelisation (Gray et al. (1994); Houkjær et al. (2006)). (O'Neil et al. (2009)) proposed



```
⊟ [ 181025 ( { Rhone-Alpes },{ Mobile },{ All } )]   695/976
   ⊞ [ 25260 ( { Grenoble },{ Mobile },{ November, 2010
   ⊞ [ 16945 ( { Grenoble },{ Mobile },{ October, 2010 } )
   ⊞ [ 18985 ( { Lyon },{ Mobile },{ October, 2010 } )]   7
   ⊞ [ 18405 ( { Ardeche },{ Mobile },{ October, 2010 } )]
   ⊞ [ 20230 ( { Saint-Etienne },{ Mobile },{ October, 201
   ⊞ [ 27415 ( { Saint-Etienne },{ Mobile },{ November, 2
   ⊞ [ 27115 ( { Lyon },{ Mobile },{ November, 2010 } )]
   ⊞ [ 26670 ( { Ardeche },{ Mobile },{ November, 2010
⊟ [ 186860 ( { Alsace },{ Computer },{ All } )]   769/1098
   ⊞ [ 17785 ( { Mulhouse },{ DELL,Acer,HP },{ Novemb
   ⊞ [ 25115 ( { Mulhouse },{ Computer },{ October, 201
   ⊞ [ 23050 ( { Colmar },{ Computer },{ October, 2010 }
   ⊞ [ 27680 ( { Strasbourg },{ Computer },{ October, 20
   ⊞ [ 17150 ( { Colmar },{ Assus,DELL,Apple MacBook
   ⊞ [ 13100 ( { Strasbourg },{ Sony Vaio,DELL },{ Nove
   ⊞ [ 24150 ( { Strasbourg },{ Acer,HP,Apple MacBook
   ⊞ [ 20140 ( { Colmar },{ Sony Vaio,Acer,HP },{ Nove
   ⊞ [ 18690 ( { Mulhouse },{ Apple MacBook,Sony Vai
⊟ [ 282530 ( { Ile-de-France },{ All },{ All } )]   1114/1586
   ⊞ [ 24875 ( { Ile-de-France },{ Samsung PC,HTC },{ N
   ⊞ [ 14090 ( { Vincennes },{ DELL,Sony Vaio },{ Nove
   ⊞ [ 19905 ( { Paris },{ Television },{ November, 2010
   ⊞ [ 24200 ( { Vincennes },{ Computer },{ October, 201
   ⊞ [ 28840 ( { Paris },{ Computer },{ October, 2010 } )]
   ⊞ [ 16560 ( { Vincennes },{ Mobile },{ October, 2010 }
   ⊞ [ 19780 ( { Paris },{ Mobile },{ October, 2010 } )]   73
   ⊞ [ 25090 ( { Ile-de-France },{ Television },{ October,
   ⊞ [ 13475 ( { Paris },{ HP,DELL },{ November, 2010 }
   ⊟ [ 26610 ( { Ile-de-France },{ Nokia,Apple },{ Noven
```

FIG. 8 – *3D visualization of a DyTree nodes' data*

the benchmark called Star Schema Benchmark (SSB) based on TPC-H which is designed to measure the performance of classical data warehousing applications. The SSB proposed modifications in normalized schema of TPC-H to model it into a star schema. However, SSB lacks features to constraint the distribution of synthesized data. (Darmont et al. (2007)) propose a tailorized benchmarh generator that provides synthetic data warehouses and workloads. Complementary works focus on defining models and techniques in order to produce data sets with more statistically sound data. Synthetic data then can expose characteristics specified by means of cardinality, distribution or sparsity constraints designed to suit individual testing needs (Binnig et al. (2007), Arasu et al. (2011), Haddadin and Lauer (2010)). Taking multidimensionality concerns, (Rabl et al. (2013)) and (Torlak (2012)) propose solutions that aim at populating dimension tables with non standard hierarchies, and fact tables with non uniform distribution. Focus is on specification of constraints definition languages and constraints solving methods. At last, let us note that in (Rabl et al. (2013)), the authors propose a new version of SSB with data skew in dimension instances and facts.

Works that design and develop testing scenarios are less numerous.(Vassiliadis et al. (2007) and Wyatt et al. (2009)) propose a set of test suites in order to experiment ETL processes in different circumstances. (Kersten et al. (2011)) develop a suite and collect a set of metrics to evaluate performance of data warehousing systems. They identify scenarios of growth evolution of the evaluated system, build query workloads and stress the system in rounds of tracks.

7 Conclusion

In this paper, an approach is exposed to implement the experimental validation of a dynamic cubing, the DyTree. Our proposal is a workflow for which we define the set of input parameters and output parameters. The input parameters are used to configure the experiment. These parameters relate to the configuration of the initial dataset, algorithms parameters and configuration of test queries. The output parameters consist in performance and behavior metrics. These parameters are integrated into a prototype. A graphical interface allows to control input parameters and to analyze metrics, in real time or offline. This proposal advocates for a formalization of the experimental validation step. We basically focus on the workflow, the prototype, and the definition and manipulation of the parameters and the metrics. This work must be completed in both directions. First, improving the workflow relies on the generation of different data sets. It is necessary to generate multidimensional datasets with certain characteristics of density, cardinality and data distribution with non-standard hierarchies. Clearly, the synthetic datasets should be as close as possible to real datasets while being configurable. The second way of improvement surely concerns the generation of set of queries in order to test the performance of querying. Currently the generation of test queries is basic and does not allow the tuning of the dataset. The introduction of new metrics on the query execution, such as the number of nodes explored and the number of the MBS selected for contributing to a query is worth considering the improvment of the dynamic cube analysis.

References

Ahmed, U., A. Tchounikine, and M. Miquel (2014). Dynamic cubing for hierarchical multidimensional data space. *Journal of Decision Systems 23*(4), 415–436.

Ahmed, U., A. Tchounikine, M. Miquel, and S. Servigne (2010). Real-time temporal data warehouse cubing. In *Proceedings of the 21st international conference on Database and expert systems applications: Part II*, DEXA'10, Berlin, Heidelberg, pp. 159–167. Springer-Verlag.

Arasu, A., R. Kaushik, and J. Li (2011). Data generation using declarative constraints. In T. K. Sellis, R. J. Miller, A. Kementsietsidis, and Y. Velegrakis (Eds.), *SIGMOD Conference*, pp. 685–696. ACM.

Binnig, C., D. Kossmann, E. Lo, and M. T. Özsu (2007). Qagen: generating query-aware test databases. In C. Y. Chan, B. C. Ooi, and A. Zhou (Eds.), *SIGMOD Conference*, pp. 341–352. ACM.

Darmont, J., F. Bentayeb, and O. Boussaid (2007). Benchmarking data warehouses. *International Journal of Business Intelligence and Data Mining* 2(1), 79–104.

Ester, M., J. Kohlhammer, and H.-P. Kriegel (2000). The DC-tree: A Fully Dynamic Index Structure for Data Warehouses. In *Proceedings of the 16th International Conference on Data Engineering (ICDE)*, pp. 379–388.

Gray, J., P. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger (1994). Quickly generating billion-record synthetic databases. *SIGMOD Record 23*(2), 243–252.

Haddadin, K. and T. Lauer (2010). Efficient online aggregates in dense-region-based data cube representations. In A. Hameurlain, J. Küng, and R. Wagner (Eds.), *Transactions on large-scale data- and knowledge-centered systems II*, pp. 73–102. Berlin, Heidelberg: Springer-Verlag.

Houkjær, K., K. Torp, and R. Wind (2006). Simple and realistic data generation. In U. Dayal, K.-Y. Whang, D. B. Lomet, G. Alonso, G. M. Lohman, M. L. Kersten, S. K. Cha, and Y.-K. Kim (Eds.), *VLDB*, pp. 1243–1246. ACM.

Kersten, M. L., A. Kemper, V. Markl, A. Nica, M. Poess, and K.-U. Sattler (2011). Tractor pulling on data warehouses. In G. Graefe and K. Salem (Eds.), *DBTest*, pp. 7. ACM.

O'Neil, P., E. O'Neil, X. Chen, and S. Revilak (2009). In R. Nambiar and M. Poess (Eds.), *Performance Evaluation and Benchmarking*, Chapter The Star Schema Benchmark and Augmented Fact Table Indexing, pp. 237–252. Berlin, Heidelberg: Springer-Verlag.

Rabl, T., M. Poess, H.-A. Jacobsen, P. E. O'Neil, and E. J. O'Neil (2013). Variations of the star schema benchmark to test the effects of data skew on query performance. In S. Seelam, P. Tuma, G. Casale, T. Field, and J. N. Amaral (Eds.), *ICPE*, pp. 361–372. ACM.

Torlak, E. (2012). Scalable test data generation from multidimensional models. In W. Tracz, M. P. Robillard, and T. Bultan (Eds.), *SIGSOFT FSE*, pp. 36. ACM.

TPC (2001 - 2011). TPC Benchmarks. `http://www.tpc.org/information/benchmarks.asp`. Accessed: 03/16/2012.

Vassiliadis, P., A. Karagiannis, V. Tziovara, and A. Simitsis (2007). Towards a benchmark for etl workflows. In V. Ganti and F. Naumann (Eds.), *QDB*, pp. 49–60.

Wyatt, L., B. Caufield, and D. Pol (2009). Principles for an etl benchmark. In R. O. Nambiar and M. Poess (Eds.), *TPCTC*, Volume 5895 of *Lecture Notes in Computer Science*, pp. 183–198. Springer.

Résumé

Cet article présente la phase de prototypage et d'évaluation d'une proposition faite dans le contexte de nouvelles applications BI en environnement fortement évolutif. Dans un précédent travail, nous avons défini un modéle et des algorithmes pour le traitement de cubes dynamiques mis é jour dés qu'un nouveau fait survient. Cette proposition comprend la définition d'une structure d'arbre, appelé DyTree, utilisée pour stocker des parties de l'hypercube. Afin de mieux comprendre le comportement de notre solution et d'améliorer son fonctionnement, nous avons ensuite mené une évaluation expérimentale compléte pour observer ses performances en fonction de différentes situations ou paramétres, que nous avons faits varier. Dans cet article, nous décrivons le worflow utilisé pour cette évaluation expérimentale. Les sorties attendus du workflow sont des mesures de performance et des mesures de comportement que nous définissons de faéon formelle. Les entrées sont composées de différents types de jeux de données paramétrables et des paramétres de configuration des algorithmes de construction de l'hypercube. Nous présentons différentes études expérimentales menées avec cet environnement et le prototype réalisé. Nous illustrons ainsi comment différents scénarios peuvent étre construits et comment les résultats expérimentaux peuvent étre utilisés pour comprendre le comportement de notre proposition et évaluer ses performances.