Efficient Cube Construction for Smart City Data[*]

Michael Scriney & Mark Roantree
Insight Centre for Data Analytics, School of Computing, Dublin City University, Dublin 9, Ireland
michael.scriney@insight-centre.org, mark.roantree@dcu.ie

ABSTRACT

To deliver powerful smart city environments, there is a requirement to analyse web produced data streams in close to real time so that city planners can employ up to date predictive models in both short and long term planning. Data cubes, fused from multiple sources provide a popular input to predictive models. A key component in this infrastructure is an efficient mechanism for transforming web data (XML or JSON) into multi-dimensional cubes. In our research, we have developed a framework for efficient transformation of XML data from multiple smart city services into DWARF cubes using a NoSQL storage engine. Our evaluation shows a high level of performance when compared to other approaches and thus, provides a platform for predictive models in a smart city environment.

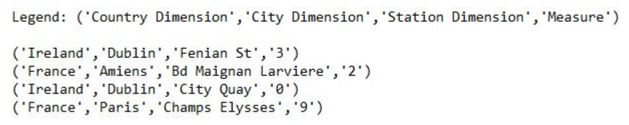
Keywords

Smart City, Data Streams, Cubes, XML Analytics

1. INTRODUCTION

Many smart city applications are fed information from their online services and repositories through XML and JSON data objects. As part of our research, we are seeking to maintain up to date cubes of information taken from multiple sources in a European capital city (Dublin) in order to make predictions about the usage of various services. The data streams in our research include car parks, bicycle sharing schemes, online auction data, air quality sensor data, and sales data. While some are not directly associated with the smart city project, they may influence decision making and are thus, included in our data cubes. Our data cubes adopt the DWARF approach [12] which was shown to offer significant savings in terms of both storage and computational efficiencies for relational data. Previously, we demonstrated how the DWARF model processes XML data [2], [3] with the same levels of efficiency. However, with the migration of many large data repositories to the NoSQL model, we investigate if DWARF cubes based on the NoSQL model can deliver similar efficiencies and thus, allow a canonical approach to managing XML and JSON smart city data streams.

[*]This work is supported by Science Foundation Ireland under grant number SFI/12/RC/2289



```
Legend: ('Country Dimension','City Dimension','Station Dimension','Measure')

('Ireland','Dublin','Fenian St','3')
('France','Amiens','Bd Maignan Larviere','2')
('Ireland','Dublin','City Quay','0')
('France','Paris','Champs Elysses','9')
```

Figure 1: Sample DWARF input

Contribution. Our overall research goal is the maintenance of data cubes, fused from the multiple sources listed above. For this paper, we limit ourselves to development and performance testing the DWARF cubes creation times and storage size as this provides the fundamentals for our system. Our contribution is the development of a DWARF to NoSQL (bi-directional) mapping mechanism, which facilitates the creation of a DWARF cube from XML data and its storage in a NoSQL database for future retrieval and querying. Our evaluation takes one of these datasets (the bicycle sharing scheme) and uses a variety of cube definitions to robustly test and compare its performance.

Structure. This paper is organised as follows: Section 2 outlines the structure and implementation of DWARF cubes; Section 3 describes DWARF cube storage; Section 4 details the transformation of the in-memory DWARF cube to a NoSQL database; Section 5 presents our evaluation; related research is presented in Section 6; and in Section 7, conclusions are presented.

2. STRUCTURE AND IMPLEMENTATION OF DWARF CUBES

The DWARF [12] compression algorithm produces data cubes of a compressed form. While clustering algorithms [10], use a form of suffix coalescing for storage efficiency, DWARF uses both suffix *and* prefix coalescing to detect duplicate aggregates before they can be computed. The result of this process is a DWARF cube which takes input in the form of a series of tuples which are used to create a DWARF. Each tuple takes the form:
`(dimension_1,dimension_2,...,dimension_n,measure)`.
For example the input in Fig. 1 would produce the DWARF cube in Fig. 2.

A **DWARF cube** is a tree-like structure which contains

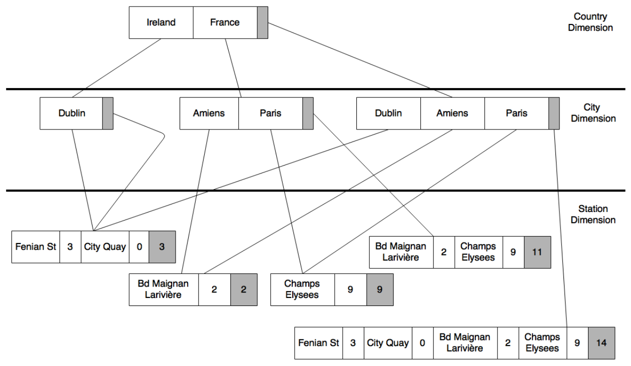


Figure 2: A sample DWARF cube created by Fig. 1 input

two main structures: *a DWARF cell* and a *DWARF node*.

A **DWARF node** is a container for groups of cells which share the same parent. At the top level of the tree in a DWARF cube there is a root node. This contains the top cells of the tree at the highest dimension level.

A **DWARF cell** represents a single item in a DWARF cube. It has a reference key and points to a DWARF node which contains all of its child cells. The cell itself is contained in a DWARF node. The value for a particular cell can be retrieved by starting at the top of the tree and following the cell paths until the target cell is found. The value of a DWARF cell is synonymous with its child's aggregate cell. If a DWARF cell does not point to a DWARF node further down the tree, it is at the bottom level of the tree and is called a *leaf cell*. A leaf cell is the smallest structure in a DWARF cube. The value of a leaf cell is derived from the `measure` item in the tuple list supplied to create the DWARF cube. This is identical to the measure found in a traditional Fact Table.

3. A NOSQL DWARF MODEL

The terminology and notation used in columnar NoSQL databases provide the basis of our NoSQL DWARF model. Briefly, a columnar NoSQL database is composed of keyspaces and column families. A keyspace can be roughly equated to a database in a relational schema and column families are similar to tables in a RDBMS.

In order to correctly model a DWARF Schema, three column families must be created: `DWARF_Schema`, `DWARF_Node` and `DWARF_Cell`. Together, these column families represent the tree-like structure of a DWARF shown in Fig. 2. This model necessitates the construction of a bi-directional model mapper where a full DWARF Schema can be rebuilt from the records stored in the NoSQL database. This is achieved by reading the records of the DWARF cells and nodes from the NoSQL database and joining them based on their unique ids.

- The `DWARF_Schema` column family stores information regarding an individual DWARF Schema. This is to ensure that multiple schemas can be stored in the same NoSQL database and is the entry point for query and traversal functions.
- The `DWARF_Node` column family stores information about a particular DWARF Node such as its parent and child cells.
- The `DWARF_Cell` column family contains information regarding a particular DWARF Cell, specifically the id of its parent and pointer node and the measure associated with the cell.

The role of the `DWARF_Schema` column family in Table 1-A is to record an individual DWARF Schema and its metadata. `node_count` is the amount of DWARF nodes in the schema, `cell_count` is the amount of cells and `size_as_mb` is an approximation of the size taken up on disk by the DWARF Schema. The `entry_node_id` attribute contains the id of the top-level node in the DWARF schema and serves as the entry point for all traversal functions. Finally, the `is_cube` attribute is a flag which indicates whether or not this particular record is a full DWARF Schema or a DWARF cube constructed from querying a DWARF schema. The role of the `DWARF_Cell` column family outlined in Table 1-C is to store information about a **DWARF Cell**. The role of the `DWARF_Node` column family (Table 1-B) is to model all `DWARF Nodes` in a DWARF cube.

4. TRANSFORMATION APPROACH

We now outline the process used to convert an in-memory DWARF Schema into a NoSQL model. This process involves transforming each Node and Cell in the DWARF structure into the relevant NoSQL query which will be used to insert it into the database.

The first step in this process is the creation of a `DWARF_Schema` item which is inserted into the `DWARF_Schema` column family seen in Table 1-A. The `id` field is obtained by querying the `DWARF_Schema` column family in the NoSQL data warehouse to determine the next id to be used. The fields `node_count` and `cell_count` are obtained by scanning the DWARF structure in-memory. The field `size_as_mb` is populated separately by querying the NoSQL data warehouse to determine the size of the DWARF structure when stored.

In order to map a DWARF Schema to a NoSQL model, each Node and Cell in the DWARF must be visited which involves a full traversal of the DWARF. Starting from the `root node` of the DWARF, the tree is traversed in a breadth-first top-down fashion. Using the DWARF presented in Fig. 2 the `root node` is processed, then the cell `Ireland`. From here the descendant node of `Ireland` is processed. This continues until all leaf cells which are descendant from `Ireland` are processed. Afterwards, the root node is revisited and the cell `France` is visited and all of its descendants are evaluated. However, as a DWARF structure contains multiple-inheritance (Nodes can have multiple parent cells) precautions must be taken in order to prevent Nodes and Cells from being processed multiple times. This is accomplished by a lookup table which records each Node and Cell visited by assigning them a unique ID. Upon visiting a Cell or Node in the DWARF structure, the lookup table is first checked to ensure that is has not already been transformed.

During traversal, the structure of a DWARF Node or Cell is evaluated and the relevant `INSERT` command for a NoSQL database is created. Using Cassandra [6] as a sample NoSQL database and the `Cassandra Query Language` (CQL) as a sample query language, a transformation is presented in Fig. 3 where the values of an in-memory DWARF cell are pre-

DWARF_Schema					
id	node_count	cell_count	size_as_mb	entry_node_id	is_cube
int	int	int	int	int	bool

A: DWARF_Schema column family

DWARF_Node				
id	parentIds	childrenIds	root	schema_id
int	set< int >	set< int >	boolean	int

B: DWARF_Node schema

DWARF_CELL							
id	key	measure	parentNode	pointerNode	leaf	schema_id	dimension_table_name
int	text	int	int	int	boolean	int	text

C: DWARF_Cell schema

Table 1: NoSQL DWARF Schema

Sample DWARF Cell Values	
parentNode:	DWARF Node (id 3)
pointerNode	`null`
key	”Fenian St”
measure	3
id	3

```
INSERT INTO DWARF CELL (id,key,measure,parentNode,
pointerNode,leaf, schema_id, dimension_table_name)
VALUES (3,"Fenian St", 3,3,null,true,1,"Station");
```

Figure 3: Sample DWARF Cell values and CQL query after transformation

sented along with the corresponding NoSQL `INSERT` command which is generated after visiting the cell.

Additionally, if a dimension table is specified in the schema definition, the `dimension_table_name` is also updated to include the name of the dimension table which contains additional information about the DWARF Cell.

As each appropriate NoSQL query is created, it is added a list of Insert expressions. These queries are subsequently executed in a bulk process to populate the column families shown in Table 1. Finally, when all column families have been populated, the NoSQL store is queried to determine the size of the DWARF structure and the `size_as_mb` field in the `DWARF_Schema` column family is updated.

5. EXPERIMENTS

All experiments were run on Ubuntu 14.04 LTS 64-bit with an ASUS Z87-PRO V motherboard, an Intel Core i7 4770K processor and 8GB 1600 MHz RAM. For both MySQL and Cassandra the DWARF cubes were inserted in bulk. All DWARFs contain 8 dimensions however, they differ in the number of source tuples used in construction.

As part of our evaluation of NoSQL-DWARF cubes and Cassandra as a storage mechanism for a DWARF cube, we used 4 schema models (as shown in Tables 4 and 5) for the purpose of comparison. The first two schema models MySQL and MySQL-Min, represent a DWARF schema in a relational model and a single table DWARF representation respectively. The next two schema models NoSQL-DWARF and NoSQL-Min represent the DWARF model described here and a single NoSQL table format respectively. The purpose of these schemas is to firstly show how a NoSQL-DWARF compares to its SQL counterpart and subsequently, how an unnormalised DWARF model (for the purpose of speed in MySQL) performs overall. Five DWARF cubes were created each in increasing size. Each one representing a time period of bikes data. The five period chosen were one day (Day), one week (Week), one month (Month), two months (TMonth) and six months (SMonth). All periods chosen use bikes data based on the CitiBikes dataset [7].

	Day	Week	Month	TMonth	SMonth
Size (MB)	2.1	17.1	54.1	113	338
Number of tuples	7358	60102	118934	396756	1181344

Table 2: The datasets used in the experiments

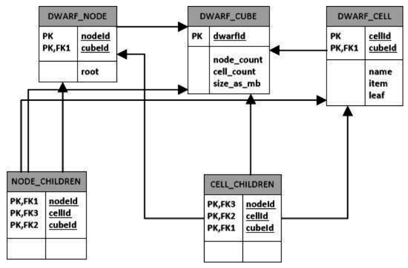


Figure 4: MySQL-DWARF Schema for a DWARF cube

We now briefly describe the comparison schemas used in our evaluation.

MySQL-DWARF.

A diagram of the schema can be seen in Fig 4. This schema was chosen as it most accurately describes a dwarf structure in a relational database. The reason for the NODE_CHILDREN and CELL_CHILDREN tables is that nodes can contain multiple cells and multiple cells can point to the same node. This multi-inheritance like structure is hard to represent accurately in a traditional RDBMS.

NoSQL-Min.

This schema was created to show that the construct of a dwarf node does not need to be stored since the dwarf cells contain the ids of their parent and pointer nodes and these nodes can be rebuilt at a later stage. An outline of the schema can be seen in Table 3.

DWARF_CUBE			
id	node_count	cell_count	size_as_mb
int	int	int	int

DWARF_Cell							
id	item	name	leaf	root	cubeid	parentNodeId	childNodeId
int	int	text	bool	bool	int	int	int

Table 3: NoSQL-Min Schema

Table 4: DWARF storage performance

Size (MB) use to store a DWARF cube					
	Day	Week	Month	TMonth	SMonth
MySQL-DWARF	2	20	80	169	424
MySQL-Min	< 1	8	33	70	178
NoSQL-DWARF	< 1	9	35	73	182
NoSQL-Min	< 1	11	45	96	243

Table 5: DWARF storage time performance

Time (Milliseconds) taken to insert a DWARF cube					
	Day	Week	Month	TMonth	SMonth
MySQL-DWARF	1768	12501	47247	100466	255098
MySQL-Min	1107	5955	22243	47936	121221
NoSQL-DWARF	927	4368	15955	34203	89257
NoSQL-Min	5699	57153	222044	484498	1219887

This schema contains only two tables. DWARF_Cube and DWARF_Cell. DWARF_Cube contains information for a particular DWARF cube inside the database. the DWARF_CELL column family contains information regarding the individual cells of the cube.

MySQL-Min..
This schema was designed to test how well MySQL performs using a schema without joins. It is based on the approach presented in the NoSQL-Min schema.

5.1 Results and Analysis

Storage Space. In terms of storage size our work outperforms that presented in [1] where the authors stored a DWARF containing 400,000 tuples with 8 dimensions in 200MB using their standard DWARF implementation and 260MB using their recursion clustering method. Conversely using Cassandra and our DWARF implementation we were able to store a DWARF cube of 1,181,344 tuples across 8 dimensions in 182MB. However, it is important to note that as different datasets were used the degree of compression provided by the DWARF algorithm differs which can affect the resulting size on disc. The MySQL-Min schema performed best for the small datasets with $<$ 1 MB for the smallest dataset and 70MB for the TMonth dataset. However, for the largest dataset the NoSQL-DWARF has a smaller resulting size (182MB for NoSQL-DWARF and 185MB for MySQL-Min). MySQL-DWARF (Fig. 4) performed worst overall with a size of 2MB for the smallest dataset and 424MB for the largest. This is due to its relational design. A DWARF Node can have many child cells. Each individual relationship between a node and a cell must be individually recorded in the `Node_Children` table. Cassandra does not encounter such problems as these relations can be stored in a single set datatype. The MySQL-Min schema had the smallest size with $<$ 1MB for the smallest cube and a size of 178MB for the largest cube. This schema had small cube sizes due to the absence of the DWARF Node construct. This significantly reduces the size on disc as the relationships between a Node and a Cell (which has greatest impact on the MySQL-DWARF schema) need not be stored. However, we anticipate the absence of a DWARF Node construct will have a significant impact on query times as DWARF Node reconstruction is required.

Storage Time. The time taken to insert a DWARF cube into each schema is outlined in Table 5. The NoSQL-Min schema performed worst overall with an insertion time of 1220 seconds for the largest dataset. This is due to the number of indexes present on each schema. With the NoSQL-DWARF schema having one index per table containing the id of the DWARF Cell, Node and Cube respectively, the absence of a DWARF Node table in the NoSQL-Min schema necessitates the addition of two secondary indexes on the DWARF Cell table `parentNodeId` and `childNodeId`. The presence of these indexes increase the resulting insertion time and size of the cube. The MySQL-DWARF schema had the second largest insertion time for the largest dataset with 255 seconds to insert. This is due to the relational nature of the schema, where each relationship between a Node and Cell must be recorded and thus, a large volume of inserts is necessary to represent the DWARF cube. Conversely, with Cassandra, this construct can be described using a set datatype which can complete in one insert operation. The NoSQL-DWARF schema performed best with an insertion time of 89 seconds for the largest dataset.

6. RELATED RESEARCH

In [5] and [8], the authors describe the creation of an OLAP cube from XML sources. Both methods involve combining the XML data sources with data obtained from a relational database, where the data obtained from both sources is abstracted. In [5], they model an XML cube using an UML Snowflake diagram. A similar method of integrating XML and UML for data warehousing was presented in [13]. The data is combined using a data integrator to determine which data source is to be queried. The end user of the system queries the OLAP server using standard OLAP query languages e.g. MDX queries. The results are presented in relational form to the user. A similar method of abstracting the data from its source format is used in our approach to create the DWARF cube and provide the functionality for a DWARF cube to be constructed from multiple source formats. In [8], they propose an XML based representation of a resulting data cube but do not address the underlying problems associated with OLAP cubes (construction time, storage space). In addition, the source data must be kept intact in its original format which could lead to issues when updating data.

In [4] and [9], they also store data cubes in native XML format similar to our NoSQL-DWARF model. However, their goals are primarily aimed towards interoperability between data warehouses as opposed to querying or data mining.

In [1], the authors employ a DWARF clustering model for storing DWARF cubes on a hard disk. They propose a Node indexing approach where a DWARF Node or DWARF Cell does not contain a pointer to its sub Nodes/Cells but instead contains the unique ID of its children. This method is adopted in our Cassandra schema. They propose two algorithms for storing a DWARF cube as a flat file; a hierarchical clustering model optimised for range queries on a DWARF structure and a recursive algorithm which is optimised for point queries on a DWARF (the recursive algorithm is also adequate for range queries). However as our approach uses Cassandra as a storage model instead of a flat file we do not encounter these problems.

Finally, in [14], the authors use MapReduce to improve cube construction time but this is only true for the most common queries with the rest being constructed in real-time so there is potentially a large overhead in cube construction time depending on the query. In [11], they propose extending DWARF to include dimensional hierarchies, a property of OLAP cubes which is not present in DWARF. [5] states that a hierarchical structure of a cube is necessary for a cube constructed from XML sources. They propose storing partial DWARFs (where all views are not materialised) to contain information pertaining to a dimension hierarchy. The result of this is a modified DWARF structure which can use the traditional OLAP operations ROLLUP and DRILL DOWN. The resulting Hierarchial DWARF structure is similar to a traditional DWARF structure (Fig 2). However, a DWARF Node can also point to another Node occupying the same dimension level, where this node would contain the dimensional hierarchy. An extension of the DWARF_Node schema outlined in Table 1-B where it contains the id of a pointer node would accommodate this functionality.

7. CONCLUSIONS AND FUTURE WORK

Today's cities have an important new resource, their knowledge infrastructure, which is generated from low level sensor networks, public databases and online information services. Before this knowledge can be exploited in order to generate real impact, we require a number of layers in the technology stack for the smart city. Above the data harvesting level, it is necessary to read and transform data streams and to create the structures (cubes) that higher level applications such as data mining can exploit. In this paper, we presented our novel cube generation approach which takes web generated data and efficiently constructs data cubes. Our usage of the DWARF and NoSQL models delivers new layers of efficiency in terms of speed and compression. Our current focus is on cube updates through efficient query primitives for our DWARF cubes.

8. REFERENCES

[1] Bao, Y.; Leng, F.; Wang, D. & Yu, G. A Clustered Dwarf Structure to Speed up Queries on Data Cubes. JCSE, 2007, 1, 195-210

[2] Gui, H. & Roantree, M. Topological XML data cube construction International Journal of Web Engineering and Technology, Inderscience Publishers, 2013, 8, 347-368

[3] Gui, H. & Roantree, M. Using a pipeline approach to build data cube for large xml data streams Database Systems for Advanced Applications, Springer Berlin Heidelberg, 2013, 59-73

[4] HÃijmmer, W.; Bauer, A. & Harde, G. XCube: XML for data warehouses Proceedings of the 6th ACM international workshop on Data warehousing and OLAP, 2003, 33-40

[5] Jensen, M. R.; MÃÿller, T. H. & Pedersen, T. B. Specifying OLAP cubes on XML data Journal of Intelligent Information Systems, Springer, 2001, 17, 255-280

[6] Lakshman, A. & Malik, P. Cassandra: a decentralized structured storage system ACM SIGOPS Operating Systems Review, ACM, 2010, 44, 35-40

[7] Marks, G., Roantree, M., Smyth, D.: Optimizing queries for web generated sensor data. In:Proceedings of the Twenty-Second Australasian Database Conference-Volume 115. pp. 47-56.Australian Computer Society, Inc. (2011)

[8] Niemi, T.; NiinimÃd'ki, M.; Nummenmaa, J. & Thanisch, P. Constructing an OLAP cube from distributed XML data Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP, 2002, 22-27

[9] Nguyen, B. T.; Tjoa, A. M.& Mangisengi, O. Meta Cube-X: An XML Metadata Foundation for Interoperability Search among Web Data Warehouses. DMDW, 2001, 8

[10] Roantree M. and Liu J. A heuristic approach to selecting views for materialization. In Software: Practice and Experience 44(10): pp. 1157-1179, 2014.

[11] Sismanis, Y.; Deligiannakis, A.; Kotidis, Y. & Roussopoulos, N. Hierarchical dwarfs for the rollup cube Proceedings of the 6th ACM international workshop on Data warehousing and OLAP, 2003, 17-24

[12] Sismanis, Y.; Deligiannakis, A.; Roussopoulos, N. & Kotidis, Y. Dwarf: Shrinking the petacube Proceedings of the 2002 ACM SIGMOD international conference on Management of data, 2002, 464-475

[13] Trujillo, J.; LujÃạn-Mora, S. & Song, I.-Y. Applying UML and XML for designing and interchanging information for data warehouses and OLAP applications Journal of Database Management (JDM), IGI Global, 2004, 15, 41-72

[14] Wang, Y.; Song, A. & Luo, J. A mapreducemerge-based data cube construction method Grid and Cooperative Computing (GCC), 2010 9th International Conference on, 2010, 1-6