2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

On Managing Very Large Sensor-Network Data using Bigtable

Byunggu Yu
Dept. of Comput. Sci. & Inf. Technol.
Univ. of the District of Columbia
Washington, DC, USA
byu@udc.edu

Alfredo Cuzzocrea
ICAR-CNR &
Univ. of Calabria
Rende, Cosenza, Italy
cuzzocrea@si.deis.unical.it

Dong Jeong, Sergey Maydebura
Dept. of Comput. Sci. & Inf. Technol.
Univ. of the District of Columbia
Washington, DC, USA
djeong@udc.edu;
dc.20007@yahoo.com

Abstract— **Recent advances and innovations in smart sensor technologies, energy storage, data communications, and distributed computing paradigms are enabling technological breakthroughs in very large sensor networks. There is an emerging surge of next-generation sensor-rich computers in consumer mobile devices as well as tailor-made field platforms wirelessly connected to the Internet. Billions of such sensor computers are posing both challenges and opportunities in relation to scalable and reliable management of the peta- and exa-scale time series being generated over time. This paper presents a Cloud-computing approach to this issue based on the two well-known data storage and processing paradigms: Bigtable and MapReduce.**

Keywords: Sensor Networks; MapReduce; HBase; Bigtable; Cloud Computing

I. INTRODUCTION

An increasing number of emerging applications deal with a large number of *Continuously Changing Data Objects* (CCDOs). CCDOs [10], such as vehicles, humans, animals, mobile sensors, nano-robots, orbital objects, economic indicators, geographic water regions, forest fires, risk regions associated with transportation routes, sensor data streams, and bank portfolios (or assets), range from continuously moving physical objects in a two-, three-, or four-dimensional space-time to conceptual entities that can continuously change in a high-dimensional information space-time. A variety of handheld computers and smartphones equipped with a GPS and other sensory devices are already available to consumers. A combined sensor that can detect and report $n \geq 1$ distinct stimuli plots a sequence of data points in the $(n+1)$-dimensional data space-time (i.e., n data dimensions and one time dimension). In GPS applications, the locations of objects can continuously change in the geographic space. In earth science applications, temperature, wind speed and direction, radio or microwave image, and various other measures associated with a certain geographic region can change continuously. In particular, for what regards to images, they can be mapped onto multidimensional points (i.e., vectors) through principal components analysis. In this sense, a sequence of images or video clip associated with a certain ontological entity is a CCDO. In water-resource research, the flow, pH, dissolved oxygen, salinity, and various other attributes can change continuously at each monitoring site. More and more CCDO applications are appearing as the relevant technologies proliferate. There is much common ground among various types of CCDOs: CCDOs produce discrete points representing the breadcrumbs of their trajectories in the information space-time.

Recent advances and innovations in smart sensor technologies, energy storage, data communications, and distributed computing paradigms are enabling technological breakthroughs in very large sensor networks. There is an emerging surge of next-generation sensor-rich computers in consumer mobile devices as well as tailor-made field platforms wirelessly connected to the Internet. Very large sets of CCDOs are growing over time, posing both challenges and opportunities in relation to scalable and reliable management of the peta- and exa-scale time series being generated and accumulated over time.

This clear trend poses several issues from the data management point-of-view, as traditional models, algorithms and schemes (mostly developed in the context of *distributed database systems*) cannot be applied as-they-are to the challenging context of managing massive CCDOs generated by large-in-size sensor networks. On the other hand, recent *Cloud technologies* offer several optimization opportunities that may be used to this end. Cloud computing is a successful computational paradigm for managing and processing big data repositories, mainly because of its innovative metaphors known under the terms "*Database as a Service*" (DaaS) [11] and "*Infrastructure as a Service*" (IaaS). DaaS defines a set of tools that provide final users with seamless mechanisms for creating, storing, accessing and managing their proper databases on remote (data) servers. Due to the naïve features of big data like those generated by CCDOs, DaaS is the most appropriate computational data framework to implement big data repositories [12]. *MapReduce* [5] is a relevant realization of the DaaS initiative. IaaS is a provision model according to which organizations outsource infrastructures (i.e., hardware, software, network) used to support ICT operations. The IaaS provider is responsible for housing, running and maintaining these services, by ensuring important capabilities like *elasticity*, *pay-per-use*, *transfer of risk* and *low time to market*. Due to specific application requirements of applications running over big data repositories, IaaS is the most appropriate computational service framework to implement big data applications. *Bigtable* [4] is the

978-0-7695-4691-9/12 $26.00 © 2012 IEEE
DOI 10.1109/CCGrid.2012.150

compressed database system running over MapReduce and other Google systems like *Google Maps*. It is deployed over a distributed file system setting.

This paper presents a Cloud-computing approach to this issue based on the following well-known data storage and processing paradigms: Bigtable [4] and MapReduce [5]. Section II provides an overview on sensor-network data management proposals appearing in literature. The conceptual basis of CCDO is discussed in Section III. Section IV presents a Bigtable model designed for very large sets of CCDOs generated by sensor networks. Section V focuses the attention on the physical implementation of our Bigtable model. Section VI discusses scalability issues of the proposed Bigtable approach. In Section VII, we provide and discuss open problems and actual research trends related to the issue of managing very large sensor-network data over Bigtable. Finally, Section VIII concludes the paper and proposes future work to be considered in this scientific field.

II. Related Work

The problem of efficiently managing massive sensor-network data generated by large-in-size sensor networks has received significant attention during the last decade [20]. In [21], *compression paradigms over Grids* are exploited to obtain efficiency during indexing and querying actives. In line with this, [22] further extends this work towards a more effective *event-based lossy compression*, with OLAP [23] querying capabilities. OLAP is also a way to compress sensor-network data via *multidimensional summarization* [24]. *Uncertainty* and *imprecision* are also significant aspects of sensor-network data processing. A solution is discussed in [25].

Traditional and well-recognized proposals in the wide sensor-network data management context are the following. *Cougar* [16] is an approach that views the sensor-network data management problem in terms of a *distributed database management problem*, and supports distributed query processing over such a database. *TinyDB* [17] is an *acquisitional and aggregate query processing system* over large-scale sensor networks that also ensures energy efficiency. To this end, TinyDB relies on sophisticated *probabilistic models*. *BBQ* [18] is a further refinement of TinyDB where probabilistic models are enhanced via elegant *statistical modeling techniques*, even with *approximation metaphors*. *PicoDBMS* [19] is a database platform powered by features that allow *smart-card data* (which clearly resemble sensor-network data) to be managed efficiently.

Among more recent initiatives, we recall: [13], which studies the challenges of managing data for the *Sensor Web*, and defines open issues in this field; [14], which proposes the system *StonesDB* that support indexing and management capabilities for sensor-network data via well-known *flash memory* technologies; [15], which presents *PRESTO*, a two-tier sensor data management architecture that comprises proxies and sensors *cooperatively acting* to support acquisition and query processing functionalities.

III. CCDO Concept: A Generic Ontology

Sooner than later, more complex and larger applications that deal with higher-dimensional CCDOs (e.g., a moving sensor platform capturing multiple stimuli) will become commonplace – increasingly complex sensor devices will continue to proliferate alongside their potential applications.

To support large-scale CCDO applications, one requires a data management system that can store, update, and retrieve CCDOs. Importantly, although CCDOs can continuously move or change (thus drawing continuous trajectories in the information space-time), computer systems cannot deal with continuously occurring infinitesimal changes. Thus, each object's continuously changing attribute values (e.g., position in the information space and the higher-order derivatives of the changing positions) can only be discretely updated. Hence, they are always associated with a degree of uncertainty.

Considering an observer who records (or reports) the state of a continuously moving object as often as possible, the object is viewed as a sequence of connected segments in space-time, and each segment connects two consecutively reported states of the object. Hence, an *Ontology*[3]-based CCDO concept can be modeled in terms of an abstracted multi-level point-type CCDO concept, which is composed by the following elements:

- *Trajectory*. A trajectory consists of dynamics and can be modeled as a function f: *time* → *snapshot*, where *time* is a past, current, or future point over time.
- *Snapshot*. A snapshot is a probability distribution that represents the (estimated) probability of every possible state in the data space at a specific point over time. Depending on the dynamics and update policies, the probability distribution may or may not be bounded.
- *State*. A state is a tuple: $\langle P, O \rangle$, where P is a location (position) in the data space-time (i.e., a point in the space-time), and O is an optional property list including zero or more of the following: orientation, location velocity vector (and rotation velocity vector, if orientation is used), location acceleration vector (and rotation acceleration vector, if orientation is used), and even higher derivatives at the time of P.
- *Dynamics*. The dynamics of a trajectory is the lower and upper bounds of the optional properties of all states of the trajectory.

For any CCDO that draws a continuous trajectory in space-time, only a subset of the object's states can be stored in the database. Each pair of consecutively reported states of the CCDO represents a single trajectory segment. Any number of unknown states can exist between the two known states. In order to support queries referring to the trajectories without false dismissals, one needs an estimation model, called the *uncertainty model* that covers all possible unknown states (conservative estimation of all possible unknown states).

Spatio-temporal uncertainty models reported in [6,8,9] formally define both past and future spatio-temporal

uncertainties of CCDO trajectories of any dimensionality. In these next-generation models, the uncertainty of the object during a time interval is defined to be the overlap of two spatio-temporal volumes, called funnels or tornados. A higher-degree model presented in [9] substantially reduces the uncertainty by taking into account the temporally varying higher-order derivatives, such as velocity and acceleration.

IV. CCDO BIGTABLE

This Section proposes our Bigtable-based approach to the problem of managing very large sets of time-growing CCDOs. A *CCDO Bigtable* is a Bigtable schema designed for very large sensor networks collectively generating peta- and exa-byte data over time. This paper proposes a modified version of the Bigtable model constructs by means of the following components:

- *Row Key*. It is a unique ID of the individual CCDO.
- *Column Keys*. They can be of the following column families [4]:
    - *CCDO.description*. It is an XML description of the CCDO. This column family may have only one column key without qualifiers. This family represents the most sparse column family of the CCDO Bigtable.
    - *Trajectory.dynamics*. This family may have multiple column keys with low- to high-order dynamics as the qualifiers. The qualifier format is of kind: *sensor_name.time_series_name*. *dynamics_name*, where *sensor_name* is the logical identifier of the target sensor, *time_series_name* is the logical identifier of the reference time series, and *dyamics_name* is the logical identifier of the trajectory dynamics. Examples of value for this family are: Trajectory.dynamics:gps.longitude.max_velocity, Trajectory.dynamics:accelerometer.y.max_acceleration, Trajectory.dynamics:temperature.max_rate_of_change.
    - *Reported_State*. This family has a distinct column key for each distinct sensor time series. It may represent the least sparse among all the families. The qualifier format is of kind: *sensor_name.time_series_name*, where *sensor_name* is the logical identifier of the target sensor, and *time_series_name* is the logical identifier of the reference time series. Examples of value for this family are: Reported_State:gps.longitude, Reported_State:accelerometer.y, Reported_State:temperature.

Note that, in a CCDO Bigtable, each entry (or cell) can contain multiple versions of the same data or time series with timestamps.

Actually, *Apache Hadoop* [1] and *Apache HBase* [2] are the most prominent and popular open source implementations of the Google's proprietary Cluster technology and Bigtable, respectively. Through the rest of this paper, we assume the Hadoop and HBase environment as the reference computational environment.

Let us now focus on more details on Bigtable. A Bigtable is a sorted table. The key structure comprises row key (most significant), column family, qualifier, and timestamp (least significant and sorted in reverse). Each ⟨*row_key*, *column_key*, *timestamp*⟩ exactly specifies a certain cell value at a certain point in time, and having an unbounded number of time-stamped values of the same cell is possible. As of this writing, HBase recommends less than or equal to three column families and abbreviated row key and column key names for file size [2].

A growing time series can be stored in a single cell via the built-in "versioning" (by using the attribute *HColumnDescriptor*). Alternatively, one can adopt "rows approach" [2], in which the timestamp becomes part of the row key in each column family. For example, one can add timestamp suffix to the row key of the CCDO Bigtable, in our implementation. In this alternative approach, the maximum number of different updates that can happen at the same time in the same column family should be defined for each column family set as the maximum number of versions in the attribute *HColumnDescriptor*. Yet another alternative is to add the timestamp as suffix to the column qualifier. This does not incur any versioning issues. However, it is important to note that different approaches will result in different significance of the timestamp in the key order and carefully programmed temporal logic in the application program will be required.

The operation get implemented in HBase allows a direct access, given a table name, a row key and optionally a dictionary (i.e., description) of values or ranges of subsequent column keys and timestamps. The operation scan allows a table access by an ad hoc and arbitrary combination of selected column family names, the qualifier names, timestamp, and cell values of the table by using the class *Filter*.

Our previous work reported in [10] presents a spatio-temporal paradigm of understanding the uncertainties of sensor data streams from a database management perspective with newly found insights into the computational efficiency: a MapReduce approach for parallelizing the uncertainty computation for enhanced performance, resolution, and scalability. The MapReduce-based solution in [10] can be applied to the CCDO Bigtable presented in this paper in a straightforward manner as well.

V. PHYSICAL VIEW

HBase runs on a networked cluster of computers powered by the *Hadoop Distributed File System* (HDFS) [1] and parallel job scheduling (via MapReduce). HDFS is a proprietary Google file system solution for supporting file-oriented, distributed data management operations efficiently. HBase system consists of a master and one or more region

servers running on the cluster. Every HBase table (Bigtable) consists of one or more regions distributed among the region servers.

Each region consists of one or more stores each of which is a physical representation of a distinct column family. In our case, the CCDO Bigtable, each region consists of three stores for the three column families (see Section IV): *CCDO.description*, *Trajectory.dynamics*, *Reported_State*. The row key attribute (see Section IV) is associated with each column family in the corresponding store. This way, empty cells are not stored in the physical stores.

Each store includes an allocated memory space, called *memstore*, of the region server and one or more data files called *StoreFiles*. Each incoming write to the table, having *Put*, *Delete*, *DeleteColumn*, or *DeleteFamily* as key type, is first written in the corresponding region's *memstore*. When *memstore* becomes full, all contents are written as a new immutable *StoreFile* in the distributed file system. Hadoop compaction process merges the immutable *StoreFiles* into a new immutable *StoreFile*, during which multiple writes with different key types regarding the same cell are resolved (deletes are performed only by major compaction running on a regular basis and combining all *StoreFiles* of the store into a single *StoreFile*), and deletes the old *StoreFiles*.

A region split occurs whenever there is a *StoreFile* larger than a predefined threshold (default 256MB). When a compaction produces such a large *StoreFile*, the region (i.e., every store of the region) is split into two. Since each region spans all column families as stores, each split is across the entire width of the table.

As regards indexing aspects, which play a critical role in this context, all records of a Bigtable are in a sorted order by the multiple key: {*row key*, *column family qualifier*, *time stamp*}, with the timestamp being in reverse order (i.e., the most recent first). Hence, the physical *StoreFile* is an indexed file (the *HFile* format in Hadoop [1]) with the index portion loaded in the main memory. This solution efficiently supports column value accesses [4].

VI. Scalability Issues

Once the number of regions has started increasing by region splits, the system starts distributing the regions among different region servers for load balancing. Moving a region from one computer of the cluster to another requires locality consideration. The region server will show the best performance when the region is available on its local disk partition representing a portion of the distributed file system space.

This physical move of a region from one node to another happens via locality-prioritized block writing capability of HDFS. A region server may be assigned a region with non-local *StoreFiles*. When a new *StoreFile* (from the *memstore* or a compaction) is being created, the file blocks are written in the distributed file system. The distributed file system is a collection of the assigned partitions of the disks of the cluster node computers. For each block produced by any region server, the system writes the first replica on the local disk of the region server and the second replica on a disk nearby in the network topology (in the same rack) and the third replica in a different rack (for a better node fault tolerance). Over time, through the compaction process, the region data will become local to the region server, eventually achieving the region-region server locality.

It has been found that this design provides rather balanced (uniform) distribution of regions (data) across the region servers. This can provide almost linear improvement regarding the table data read/write performance with increasing number of cluster nodes and almost constant with increasing data size [4].

Given a table access (e.g., `get` or `scan` operations with filters), a map process (mapper) is created for each region. Multiple mappers can access their regions of the CCDO Bigtable at the same time, resulting in a parallel processing of the query over the distributed regions. The actual degree of the parallelism is naturally limited by the size of the cluster (in terms of the number of computing nodes).

VII. Open Problems in Managing Very large Sensor-Network Data over Bigtable

There are a number of open problems and actual research trends related to the issue of managing very large sensor-network data over Bigtable. In the following, we provide an overview on some of the most significant of them.

(*a*) *Data Source Heterogeneity and Incongruence*. Very often, distributed sources producing sensor-network data of interest for the target application (e.g., legacy systems, Web, scientific data repositories, sensor and stream databases, social networks, and so forth) are *strongly heterogeneous and incongruent*. This aspect not only conveys in typical integration problems, mainly coming from active literature on *data and schema integration issues*, but it also has deep consequences on the kind of analytics to be designed.

(*b*) *Filtering-Out Uncorrelated Data*. Due to the enormous size of sensor-network data repositories, dealing with large amount of data that are *uncorrelated* to the kind of application to be designed occurs very frequently. As a consequence, filtering-out uncorrelated sensor-network data plays a critical role, as this heavily affects the *quality* of final Bigtable applications to be designed.

(*c*) *Strongly Unstructured Nature of Distributed Sources*. In order to design meaningful Bigtable applications, it is mandatory that input sensor-network data are transformed in a suitable, structured format, and finally stored in the HDFS. This poses several issues that recall classical ETL processes of Data Warehousing systems, but with the additional challenges that distributed sources alimenting Bigtable repositories are *strongly* unstructured (e.g., social network data, biological experiment result data, and so forth) in contrast with less problematic unstructured data that are popular in traditional contexts (e.g., XML data, RDF data, and so forth). Here, transformations from unstructured to structured format should be performed on the basis of the analytics to be designed, according to a sort of *goal-oriented methodology*.

(*d*) *High Scalability*. High scalability is one of the primer features to be ensured for a very large sensor-network data

management system over Bigtable. To this end, exploiting the Cloud computing computational framework seems to be the most promising way to this end [12], like we do in our CCDO Bigtable based implementation. The usage of the IaaS-inspired Cloud computing computational framework is meant with the aim of achieving some important characteristics of highly-scalable s systems, among which we recall: (*i*) *"true" scalability*, i.e. the effective scalability that a powerful computational infrastructure like Clouds is capable of ensuring; (*ii*) *elasticity*, i.e. the property of rapidly adapting to massive updates and fast evolutions of big data repositories; (*iii*) *fault-tolerance*, i.e. the capability of being robust to faults that can affect the underlying distributed data/computational architecture; (*iv*) *self-manageability*, i.e. the property of *automatically* adapting the framework configuration (e.g., actual load balancing) to rapid changes of the surrounding data/computational environment; (*v*) *execution on commodity machines*, i.e. the capability of scale-out on thousands and thousands of commodity machines when data/computational peaks occur.

VIII. Conclusions and Future Work

Inspired by recent trends in the context of effectively and efficiently managing very large sensor-network data, in this paper we have focalized on the issue of managing datasets modeling massive CCDOs generated by large-in-size sensor networks. We introduced an innovative Cloud-computing-based solution that combines two well-known data storage and processing paradigms, namely Bigtable and MapReduce, to gain effectiveness, efficiency and reliability during this so-engaging task. This has conveyed in the so-called CCDO Bigtable concept, which extends traditional Bigtable. We provided an Ontology-based conceptualization of CCDO Bigtable objects, along with its physical implementation, and discussed scalability issues related to so-defined objects.

Future work is mainly oriented towards improving the performance of search and retrieval capabilities in CCDO Bigtable, by means of a suitable empirical study. Also, we are currently integrating the CCDO Bigtable within the web-based and database-supported three-tier sensor cloud system *Smart Sensor Command and Control System* [26].

Acknowledgement

This research was supported by the U.S. National Science Foundation (NSF) Grants 0911969 and 0940393.

References

[1] Apache Hadoop, http://hadoop.apache.org

[2] Apache, HBase, http://hbase.apache.org

[3] M. Uschold, M. Gruninger, M., "Ontologies: Principles, Methods and Applications". Knowledge Engineering Review 11(2), pp. 93-155, 1996.

[4] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, R.E. Gruber, "Bigtable: A Distributed Storage System for Structured Data". ACM Transactions on Computer Systems 26(2), art. 4, 2008.

[5] Dean, J., Ghemawat, S., "MapReduce: Simplified Data Processing on Large Clusters". Communications of the ACM 51(1), pp. 107-113, 2008.

[6] B. Yu, "A Spatiotemporal Uncertainty Model of Degree 1.5 for Continuously Changing Data Objects", Proceedings of ACM SAC Int. Conf., pp.1150-1155, 2006.

[7] B. Yu, T. Bailey, "Processing Partially Specified Queries over High-Dimensional Databases". Data & Knowledge Engineering 62(1), pp.177-197, 2007.

[8] B. Yu, S.H. Kim, "Interpolating and Using Most Likely Trajectories in Moving-Objects Databases". Proceedings of DEXA Int. Conf., pp.718-727, 2006.

[9] B. Yu, S.H. Kim, S. Alkobaisi, W.D. Bae, T. Bailey, "The Tornado Model: Uncertainty Model for Continuously Changing Data", Proceedings of DEXA Int. Conf., pp.624-636, 2007.

[10] B. Yu, R. Sen, D.H. Jeong, "An Integrated Framework for Managing Sensor Data Uncertainty using Cloud Computing", Information Systems, doi:10.1126/ j.is.2011.12.003, 2012.

[11] H. Hacigumus, B. Iyer, S. Mehrotra, "Providing Database as a Service". Proceedings of IEEE ICDE Int. Conf., pp. 29-38, 2002.

[12] D. Agrawal, D. Das, A. El Abbadi, "Big Data and Cloud Computing: Current State and Future Opportunities". Proceedings of EDBT Int. Conf., pp. 530-533, 2011.

[13] M. Balazinska, A. Deshpande, M.J. Franklin, P.B. Gibbons, J. Gray, M.H. Hansen, M. Liebhold, S. Nath, A.S. Szalay, V. Tao, "Data Management in the Worldwide Sensor Web". IEEE Pervasive Computing 6(2), pp. 30-40, 2007.

[14] Y. Diao, D. Ganesan, G. Mathur, P.J. Shenoy, "Rethinking Data Management for Storage-centric Sensor Networks", Proceedings of CIDR Int. Conf., pp. 22-31, 2007.

[15] M. Li, D. Ganesan, P.J. Shenoy, "PRESTO: Deedback-Driven Data Management in Sensor Networks". IEEE/ACM Transactions on Networking 17(4), pp. 1256-1269, 2009.

[16] Y. Yao, J.E. Gehrke, "The Cougar Approach to In-Network Query Processing in Sensor Networks". SIGMOD Record 31(3), pp. 9-18, 2002.

[17] S. Madden, M. Franklin, J. Hellerstein, W. Hong, "TinyDB: An Acqusitional Query Processing System for Sensor Networks". ACM Transactions on Database Systems, pp. 122-173, 2005

[18] A. Deshpande, C. Guestrin, S. Madden, J.M. Hellerstein, W. Hong, "Model-Driven Data Acquisition in Sensor Networks". Proceedings of VLDB Int. Conf., pp. 588-599, 2004.

[19] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, J. Heidemann, R. Govindan. "Multi-Resolution Storage in Sensor Networks". ACM Tranasctions on Storage 1(3), pp. 277-315, 2005.

[20] A. Cuzzocrea, "Intelligent Techniques for Warehousing and Mining Sensor Network Data". IGI Global, 2009.

[21] A. Cuzzocrea, F. Furfaro, G.M. Mazzeo, D. Saccà, "A Grid Framework for Approximate Aggregate Query Answering on Summarized Sensor Network Readings", Proccedings of GADA Int. Conf., pp. 144-153, 2004.

[22] A. Cuzzocrea, S. Chakravarthy, "Event-Based Lossy Compression for Effective and Efficient OLAP over Data Streams", Data & Knowledge Enginering 69(7), pp. 678-708, 2010.

[23] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, H. Pirahesh, "Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub Totals". Data Mining and Knowledge Discovery 1(1), pp. 29-53, 1997.

[24] A. Cuzzocrea, "CAMS: OLAPing Multidimensional Data Streams Efficiently". Proccedings of DaWaK Int. Conf., pp. 48-62, 2009.

[25] A. Cuzzocrea, "Retrieving Accurate Estimates to OLAP Queries over Uncertain and Imprecise Multidimensional Data Streams", Proceedings of SSDBM Int. Conf., pp. 575-576, 2011.

[26] UDC Projects, http://informatics.udc.edu/projects.php