Managing a Large Shared Bank of Unstructured Data by Using Free-Table*

Xiao ZHANG[1,#], Xiaoyong DU[2,++], Jinchuan CHEN[3], Shan WANG[4]

Renming University of China

Key Lab. of Data Engineering and Knowledge Engineerging, MOE, China
59 Zhongguancun Street, Haidian District, Beijing, China

{[1]zhangxiao,[2]duyong, [3]jcchen, [4]swang}@ruc.edu.cn

***Abstract*—This paper presents a reference framework, called *BUD*, to manage a large shared bank of unstructured data. This paper lists several important issues on managing or maintaining the unstructured data in BUD. BUD stores and manages the ever-growing unstructured data by introducing a novel technique called *free-table*, which is a conceptual view for end-users and a physical entity maintained by transactional storage manager of BUD. Free-table is *cell*-oriented but not column-oriented as relational table. It can store various types of unstructured data in cell with different versions. Additionally, we study two cases, VMP and PXRDB, to show that our proposal is feasible and tractable.**

I. INTRODUCTION

Unstructured data is pervasive now and contains various potential values out of human's imaginations[1]. Meanwhile, its volume are increasing explosively[2]. It has become necessary to establish an extensible infrastructure to manage the ever-growing unstructured data and then extract more valuable information from them continuously. In this paper, we propose one reference framework, called *BUD*, refering to the ***B***ank of ***U***nstructured ***D***ata, and discuss the key issues while constructing an instance of BUD.

A BUD system has to manage voluminous and various formats of unstructured data. Its data model must be universal and extensible, i.e., adaptive to any existing or future emerging unstructured data. Unlike structured relation data, unstructured data has no explicit schema to represent its various internal structures, while its descriptive metadata is structured relational data. So it is one natural approach to manage and describe them by combining unstructured data with structured relational data in a uniformed platform. There is no such a uniformed platform now. In practice, in relation databases, the unstructured data is stored in LOB and is interpreted by applications. Object model seems to be an appropriate choice. Unfortunately, it bases upon Abstract Data Type (ADT) theory and an ADT is a pre-defined data type, which is difficult to do run-time optimization. In addition, an object can only maintain the current state of the data while unstructured data might have many versions with time going.

In this paper, we propose a new data model, *Free-Table* (FT in short), to meet this requirement and overcome the drawbacks of the existing models. At a first glance, researchers who are familiar with this area might take BUD as only a coined word and another appearance of wide-table[3, 8] or a bigtable[4]. But, FT is much different from the wide-table and object as well. *First*, FT is cell-oriented table or relation while wide-table is still column-oriented, in other word, domain-oriented as the traditional relational model. *Second*, FT is much more "*free*" than XML. FT allows any format of data to be saved in a cell and its meaning or content depends on the operators affiliated to the cell. *Third*, it is not the traditional object. A cell has different versions beside the current state. The operations of a cell are optimizable that is very different from object model.

A. Organization of this paper

Section II describes our reference framework BUD and lists some important research problems related to management of unstructured data, including system architecture, data model, quality of services, query language, distributed TSM, metadata repository, optimization and versioning.

We present what an FT is in detail in Section III. Next, two use cases of FT, VMP(*Video Management Platform*) and PXRDB(*Pure XML-Relational Database*), are studied in Section IV that show our proposal is feasible. We conclude this paper in Section V.

II. A REFERENCE FRAMEWORK---BUD

We will introduce a layered reference framework, *BUD*, to manage the unstructured data in a database approach. BUD can be illustrated as in Figure 1.

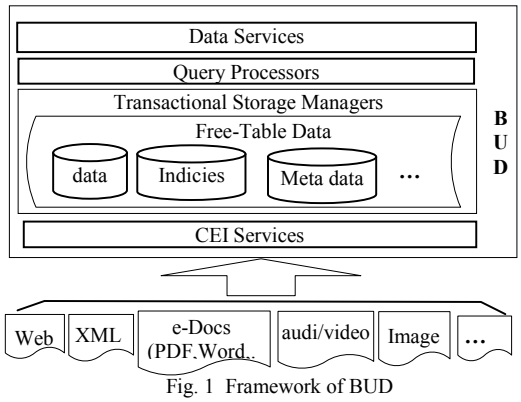

Fig. 1 Framework of BUD

BUD has four core components: *Data Services*, *Query Processors*, *Transactional Storage Managers* and *CEI Services*, i.e., USD Capture, Extraction and Integration. All

*Partly supported by National 863 High Tech. Project (No. **2009AA01Z149**),
Partly Supported by Shanghai Key Lab. of Intelligent Information Processing, China. (No. **IIPL-09-018**),
++ Partly Supported by National Science Fundation of China. (No. **60873017**).

978-0-7695-4012-2/10 $26.00 © 2010 IEEE
DOI 10.1109/APWeb.2010.78

components are required to be extensible and scalable. Each layer can be a cluster of distributed service units.

A. Data Service

Data Services(DSs) are a collection of services that accept the requests from users and feed back desired data with specific formats. DSs allow users to harvest various kinds of information through specific service(s). We recommend four types of services to manage the unstructured data.

- *Uniform query service*. User can retrieve different types of data, such as well-formed relational data, xml data, e-docs, video, audio and so on, in a single query.
- *Content retrieval service*. It enables user to do semantic search over the unstructured data with no idea of their real format.
- *Import/Export/Transformation service*. User can execute them to convert data among FT data, relational data, xml data and others.
- *Analyticcal and mining service*. They run built-in analytical and mining algorithms to accomplish classification, aggregation, association rule finding and so on.

DS layer also provides users with many DBA utilities.

B. Query Processor

Query Processors(QPs) are invoked by any DS to act as the brokers to search various data in *FT* through accessing the available Transactional Storage Managers(TSMs).

A QP unit has those functionalities similar to the query processor in a SQL engine. It consists of parser, optimizer, plan generator and executor. Each component processes its task in a distributed manner. For example, the optimizer evaluates which plan is best according the pre-defined rules. Also, the generator translates the computation primitives into operators adaptive to distributed infrastructures.

More importantly, a set of operators are implemented in the QP for unstructured data and treated as the built-in first-class citizens like relational operators. These operators include keyword-search, approximation-query, content-based retrieval and so on. Moreover, QP is highly extensible so that BUD allows user to register new operator(s) for unstructured data.

QP in BUD enhances the result processing through several post-retrieval operations. QP can accept users' online feedback and then adjust the query policy taking into account users' satisfaction to afford adaptive and personalized query service.

Another important issue in BUD's query processing is the management of uncertain data. In BUD, semantics, contents and even some metadata, e.g., information schema, of unstructured data are not pre-defined. They can be automatically extracted from unstructured data according to users' requirements. Hence these data are intrinsically uncertain. Techniques such as probabilistic queries [12] should be incorporated in QP for providing reasonable query services over uncertain data.

C. Transactional Storage Manager

TSMs are the kernel of BUD framework since various formats of unstructured data might emerge in the future, BUD must be capable of managing them. We design a concept and technique of *Free-Table* to build the TSMs.

TSM mainly manipulates three types of data: *user data* in FT, *indices* used for optimization and summary retrieval, and *metadata repository* with incremental information schema for unstructured data. It has capabilities of: 1) multi-version concurrency control; 2) distributed task scheduling over new computational environment, for instance, cloud environment; 3) security assurance based on certificates; 4) versioning data that maintains the different versions of the original and extracted data; 5) incremental maintenance of data extraction in a *pay-as-you-go* manner that makes BUD able to explore more valuable information and refines them with more approaches; 6) allowing users to build indices for FT. Index can improve system performance and leverage the QoS for managing unstructured data.

D. CEI Services

Capture, Extraction and Integration(CEI) layer plays very important role in BUD because it acts as the preprocessor before unstructured data are loaded into TSM. CEI can: 1) collect or crawl unstructured data from web according to the policies; 2). execute declarative query to extract information including structured description, information and schema, so that more effective analysis can be deployed. 3). classify the input data by computing the similarity of extracted features, and infer the global schema thereafter.

E. Open Research Problems

As a reference framework, there are many open problems to be solved. We list some of them and our consideration on these issues.

1. *How to build a fully extensible architecture*?

BUD should not only be easy to add more components, but also enable users or DBA to register a new type of unstructured data that will be as accessible as other existing types in BUD. So BUD needs to be fully extensible in both functionality and inclusion of new data type.

2. *What is the universal data model for diverse unstructured data*?

BUD manages all sorts of data no matter they are collected, extracted or self-derived. So it is a key issue to provide one effective technique to model, in particular, those unstructured data so that any operator can be designed for certain general-purpose but not arbitrarily. To solve this problem, we propose a novel technique, *Free-Table*, to model all data in BUD. FT will be discussed thoroughly in Section III.

3. *How to guarantee the quality of data services(QoSs)*?

Both content retrieval and analytical mining services are subject to users' expectation. In term of the content understanding, to assure the QoS is to make them able to "understand" the semantics of unstructured data precisely. This needs more efforts in both data services and CEIs.

4. *Long way to standardize uniform query language.*

One *goal* of BUD is to free users from learning more complicate query languages and understanding the certain specific operations on USD. Users can construct one query in a uniform language to manipulate diverse types of data

including relational data, semi-structured data and unstructured data. One feasible way is to support a collections of user-defined functions(UDFs). The problem remains that it will take a long time to reach the standardization of specifications of UDFs.

5. *How can TSMs manage distributed USD?*

Distributed TSMs are necessary to cope with ever-emerging unstructured data and computing environment, such as grid computing or cloud computing. Even for an enterprise environment, its partners might spread over several areas or a whole country. We propose a data portal model in this paper to organize the TSM as shown in Fig. 2.

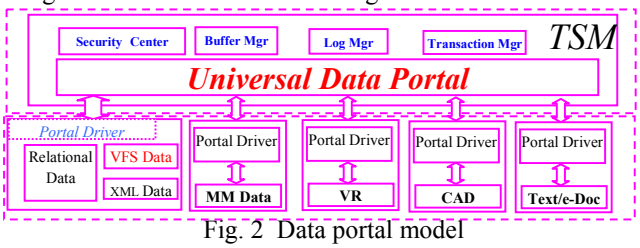

Fig. 2 Data portal model

TSM is adaptive to the content of unstructured data. Mostly, it is difficult to determine the general optimization rule or policy applying to the unstructured data before processing it. Some hidden schema or structural information might be discovered with processing the data. Therefore, TSM needs to adjust its prior storage plan to a more efficient organization.

6. *How to organize the metadata repository and evolve it?*

There is no bypass to manage unstructured data without metadata repository. Whether users or BUD has no idea about the unstructured data if no any metadata for them, that is, the metadata repository is the "open sesame" to the treasure cave of unstructured data. So, building a metadata repository is one of critical tasks to deploy a BUD instance. Furthermore, it will evolve with time going since BUD is extensible and the contents understanding of USD will become more complete.

In literature, [5] presents an incremental approach to extract schema. "Pay-as-you-go" or "best-effort" [6,7,1] are similar ideas to accumulate the knowledge of schema. Unfortunately, the up-to-date methods mainly solve the problems related to unstructured text data. Lot of works have not been carried out to build the metadata repository for other sorts of unstructured data, such e-docs, xml data, video, audio, image and so on.

7. *Optimization must NOT be neglected.*

Effectiveness and efficiency are two elementary metrics to evaluate whether an instance of BUD can manage its data as required. Partly different from DBMS, some operators in BUD are data-intensive and computational.

For example, version comparison for a large volume of documents will take long time to execute in a naive way. The computational cost, however, can be cut down dramatically if change log is available. In this case, much more efficient query will be executed on log instead of the whole document.

On the other hand, many contents and semantics cannot be extracted exactly from unstructured data. User can only be provided some approximate result. The effectiveness of the query result, for instance, recognizing face of the wanted, will significantly influence the user's decision. In general, BUD has to trade off the effectiveness of result and the efficiency of execution under different environment. Optimizer will cope with this challenge.

8. *Version Management.*

Many applications involved in unstructured data are not OLTP-oriented. Update generates the versioned data but not the replacement of the previous ones. Probably, a new version might be derived with more effectiveness and/or resulting in the efficient execution.

Data versions can consume space remarkably, for example, in summarization of video clips. Here, CEI service will extract representative clips from videos based on different algorithm and users' feedback. These versions are not same but relevant, and some might be able to be derived from others while some might be accessed frequently. So BUD should enhance the version control mechanism to compress those redundant versions and make the frequently used version able to be accessed efficiently.

Additionally, version management is the basis of the multi-version concurrency control policy in BUD.

To our knowledge, the problems mentioned above are very limited for this research area and its list will definitely go on and on. Much more efforts are required to find the solutions. In this paper, for space reason, we mainly focus on the second problem, i.e., modeling the data in BUD. We propose a novel concept, *Free-Table*, as a universal data model used by BUD. The next section will describe this model in details.

III. FREE-TABLE

In this section, we try to model unstructured data into a *free-table* that is used to store not only the originally-captured or extracted unstructured data but also the derived or versioned.

Definition. 1. *Free-table*. A free-table is a table with one or more attributes among which some may have symbolic type, *cell*, devised for storing and accessing the unstructured data.

The distinguished characteristic of the Free-table is that the cell type can be assigned to attributes. A free-table reduces to a relation if there is no any celled attribute. We have the definition of cell type as below:

Definition. 2. *Cell Type*. Cell is a variable by-instance virtual interface type for modeling the unstructured data. Cell is a symbolic type.

For an FT tuple, any cell-type attribute contains data that can have no same internal structure for all of its instances. Given a free-table, for example, *newsitem*, it has the following schema:

NewsItem(*Title* TEXT, *Data* DATETM, *Comment* CELL)

For all NewsItems, the column *Comments* is cell typed and can have various forms, such as text, xml, e-docs, video or audio, among different tuples and even in different versions of a same tuple. It is a distinct feature that makes a table free from rigid type constraints.

We will discuss some important issues related to the implementation techniques on cell type next.

A. Properties of a Cell

In our proposal, cell type is represented as a quadplex as

Cell=(byteData, rdfContent, pfOperator[], byteVersion)

First of all, a cell must be a container of unstructured data. The first property, *byteData*, is the container of the captured/extracted unstructured data w/wo transformation from its original source. *byteData* can be a byte stream.

rdfContent is in form of RDF or other structured form. It keeps the semantics and information for the current version. Also, it does maintain full version of knowledge accumulated from the first version.

The third one is *pfOperator*. pfOperator is a set of operators applicable to byteData to access the unstructured data in this cell. Meanwhile, this set includes the operators manipulating the rdfContent and byteVersion. In general, it contains one operator to show information of all operators in this set.

The last one is *byteVersion* property. Its values can be used to keep track of the version history of the unstructured data and its change log of its content or semantics information extracted.

To meet the requirement of large volume of unstructured data, the space limitation of a cell is up to the capacity of storage media. The output of CEI can be stored in the byteData and extracted information is saved in the rdfContent. The byteVersion will be kept at same time.

B. User's View

Users can access the celled attributes as same as the others with non-cell type, e.g., integer. In most case, the attribute name stands for its value no matter whether it is virtual type or a built-in data type. In BUD, we comply with this user's view of the cell-type attribute.

Take the free-table NewsItem for example again, BUD allows user to submit an SQL query like:

SELECT title, comment FROM NewsItem;

QP parses it and executes it by transforming the comment in the select-list into its appropriate internal form according to the default rule(s). The default rule(s) can be used to determine the default operator and which property of the cell will be used to generate the query result. In particular, the value of the comment column in a tuple will generate a NULL value if no default operator assigned to this attribute.

The default operator belongs to the pfOperator set.

C. Operator Set

Operators are keys to unstructured data in a cell. Any properties of a cell can be accessed through predefined and/or user-defined operators if the appropriate privilege is granted. Operator set contains elementary operators that are primitive operators for all cell instances in a free-table. Minimally, the set should provide functionality as below:

- *Get/SetData*: the most fundamental operators. They are responsible for reading from and writing into byteData respectively. Particularly, SetData must be assigned to the cell before its data is loaded.
- *Search*: it is an entry-point operator to locate the portion of byteData based on the search argument(SARG). Search can also find the information in rdfContend if user requests to do so.
- *EvalCost*: the basic operator is used to report the cost prediction to optimize the query execution. It evaluate the cost of a specific operation on the cell so that the optimizer chooses an efficient plan.
- *GetInfo*: it retrieves the metadata repository to find the description for the specific or all operators attached to the cell instance or a cell attribute.

There are more useful operators not listed above, such as Index, Match, etc. The elements of operator set can be created and imported if necessary for enhancing its extensibility.

D. Extensibility

Cells in a free-table might have lots of unrecognized information so that the functions for a cell can grow on and on. It means that cell type have very high extensibility.

Any operator of a cell supports to overload its implementation so as to extend its functionality. On the other hand, BUD enables users to register the operator set for new type of unstructured data, for example, type T and then type T of data can be loaded into BUD after processing of CEI.

For a cell instance, its pfOperators[] is never fixed. Users can easily add new element into it to meet new need.

Regarding storage, byteData can be organized with a more complex internal storage structure so that it can receive large volume of unstructured data and their versions. The internal structure is understood by the Get/SetData and other operators.

E. Open Research Issues

Besides our comments above, some important research issues must be paid more attentions

- *Adaptive storage mechanism for a free-table*

A cell-type attribute has no same built-in data type for all instances, but all instances have the same or similar semantics, e.g., comments for the free-table NewsItem. One FT can have more cell attributes and/or be a wide sparse table [3]. These cell attributes should be stored in BUD more efficiently. This problem must be focused more intensively while designing and implementing the TSM. The storage mechanism also needs to be adaptive to the run-time environment or infrastructure so as to fully exploit the available resources.

- *Indexibility*

It seems difficult to index the unstructured data, but it is nearly impossible to improve the performance without any index. BUD must build some indices to speed-up the access to the desired byteData.

This is a challenging problem. One clue is there might be a way while combining the unstructured data with structured data including those metadata or extracted information.

- *Version Purging*

To restate the above opinion here, the first three properties of a cell except the byteVersion can be involved into version management. For example, a new content-understanding method is implemented as a new version of an operator. Consequently, a new version of rdfContent is generated and probably some portions of derived byteData are spawned as well. After a pretty long period, lots of versions will be accumulated for a cell instance. The version purging needs more research.

- *Uncertain Data Management*

Semantics, contents and even some metadata, e.g., information schema, of unstructured data are not pre-defined. They can be automatically extracted from unstructured data according to users' operations and feedbacks. These data are uncertain while they are valuable for managing unstructured data. So, management of uncertain data in BUD is a big issue.

As the basis of the BUD in this paper, there are more research or implementation issues. We could not enumerate all of them for space reason. We will present two use cases of FT briefly in next section.

IV. CASE STUDIES

In this section, we discuss the possible solution to manage two typical sorts of unstructured data in two cases. One is for video data and the other is for XML data.

A. Case Study---Video Surveillance

Video is typical unstructured data used everyplace and contains lots of information than a bundle of plain text. In this case study, we will demonstrate how to manage the video surveillance in a Video Management Platform (VMP).

VMP continues to capture surveillance video through cameras installed in, for example, a retail store. One interesting scenario is :

Some customers can be recognized if they had been the store and bought something before. Once identifying them, VMP alerts salespersons to provide them better shopping experience, eg. special offers, discount or recommendation.

VMP uses a group of relations. One of them is a free-table *ftVideo* maintaining video data and another is *ftKeyframe* used to record the photos of the recognized customers in the store.

ftVideo(title, date, length, ***vseg***,...)

ftKeyframe(id, ***keyframe***, position,...)

In the above two FT, both attributes *vseg* and *keyframe* are cell type and the others are annotations or descriptions related to them. Its byteData is well organized and can be accessed more easily by executing the affiliated operators.

- Storage Organization

Comparatively, it seems easier to store keyframe than vseg because vseg consumes more space. We store the byteData of both in LOB and split the large volume of LOB into smaller segments (see Fig. 3) to improve the performance[9]. The segmentation supports multi-granularity, such as shot, scene, frame and so on.

cellId	seg#	byteData
110	1	<preprocessed>
110	2	<preprocessed>
... ...	... ...	... ...

Fig. 3 . Storage solution

For purpose of tracking people and locating a specific video clip for a specific person or event, VMP allows user to create the multi-summary index(MSI) on surveillance video based on keyframe photo as in Fig. 4.

A MSI can be built on ftKeyframe by executing the elementary operator *Index* for cell keyframe like:

keyframe.Index(level,...)

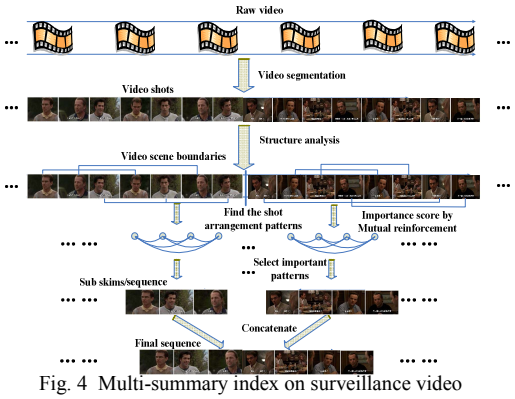

Fig. 4 Multi-summary index on surveillance video

Where *level* indicates the levels of the summaries. It then can be used to locate specific video clips on various levels according to the parametric face image. For example,

keyframe.Search(photo, level#,...)

The result is set of position-pointers to clips of vseg which have index-entries on the *level#* MSI matching the *photo*.

For video and image in VMP, both Index and Search are the elementary operators, we have more to list next.

- Elementary Operators

Elementary operators are users' utilities to access the cell and extract more information in a "pay-as-you-go" manner. VMP has several classes of elementary operators for different purposes:

- *Exactly Find*
  - a) GetVal/SetVal: saves and retrieves byteData.
  - b) GetInfo: retrieves the metadata of the operators and the extracted content as well.
  - c) Locate: finds the specific clip or image according to timestamp and/or other comparable operators for the non-cell attributes.
- *Approximately Search*
  - a) Search: searches video clips or images in IR-style.
  - b) Index: builds the MSI based on face-recognition and other machine-learning methods.
  - c) EvalCost: statistically reports the cost to run a specific video-/image-related operation.
  - d) Match: reports whether two video/image match each other probably.
- *Viedo-oriented/Image-oriented*
  - a) FeatureExtraction:extracts features of a keyframe.
  - b) Convert: converts one video clip from one format to another one, e.g. from mpeg1 to mpeg4.
  - c) Split: splist a large segment into smaller clips.
  - d) Merge: merges a few clips into a larger segment.
  - e) Annotate: adds annotations for a video/image.
- *Version Management*
  - f) SetVer: sets the current data as a real version.
  - g) Purge: compresses the version history. Some obsolete versions will be discarded.

More operators are not included here for space reason. Furthermore, the set can be expanded if necessary.

VMP is designed and implemented based on PostgreSQL. One main goal is to build a RDBMS-oriented instance of BUD and enable the interplay of structured and unstructured data[1]. In VMP, operators run in context of the RDBMS and can easily or even smoothly be combined with the relational structured data. Specially, the elementary operator *EvalCost* is available to the optimizer of the SQL engine so that QP can choose a more efficient plan. We call this cost-awared video management. This is one of significant contributions of VMP.

The services in VMP are put into UIMA, an open source architecture now. We will try to port it to cloud computing environment in the future.

B. Case Study---XML Data with Relation Data

XML data is the semistructured data and its volume keeps growing on and on no matter what arguments or even debates about it. In this section, we will discuss our propsal on how to manage both XML data and relational data universal. This proposal is not to implement a complete instance of BUD now.

- Multi-tiered uniform integration

Integration can be implemented in different layers to combine XML data with relation data referencing BUD.

Tier one: Language layer. We define and support a cooperative query language mixing SQL and XQuery language where SQL statement can invoke processiong of XQuery and vice versa. For users' view, it is uniform.

Tier two: Optimizer and executor layer. A universal optimization model is able to optimize query written in SQL, XQuery or both. In particular, it can generate more efficient plan if storage manager supports this integration.

Tier three: Adaptive storage manager. Storage manager can adaptively select the efficient storage schema according to the data content but not rigidly based on the data type.

We now mainly focus on how to adaptively store XML data. Regarding, for example, a document *customer.xml* with well-formed content much like a table, PXRDB then can store it in a relation rather than in an XML document object.

In addition, the above three tiers is backward compatible, i.e., Tier three integration supports both tier one and two.

- Keyword search over XML and relation Data

Keyword search is an important data service for non-expert end user to search any data [10,1,3,5, 11]. In order to support keyword search over XML and relation data in a same query, the search engine is built at the QP level. Semantics might be lost in part if it is up-moved to search service simply.

An ranking model is the key to search the meaningful result. In this case, a novel evaluation model must be designed.

- Adaptive storage schema

Integration can be implemented in storage manager. It is an interesting and challenging problem in PXRDB. The novel storage policies are deployed to make XML data adaptively stored according its content as mentioned above.

Indexing techniques and transaction management also need to change to adapt to storage schema.

PXRDB is a Pure XML-Relational Datbase system. It is RDBMS-oriented and implemented based on PostgreSQL, too. We also want to manage more data besides relational data in this system. The operations on XML data can be respecified and implemented as operators affiliated to XML type. Furthermore, XML type can be degraded to be cell type. In this part, PXRDB is an instance of BUD, too.

V. Conclusions and Futrue work

Data, especially the unstructured data, are becoming the wealth for a person, community, society or the whole world. Plenty of information can be extract from unstructured data and stored in structured data. It is natural and necessary way to manage both simultaneously. We propose a reference framewrok, *BUD*, to address this basic issues in this paper.

Extensible architecture and data model are the first two most important issues. The extensibility of BUD makes it open to new formats, and to effectively manage all known forms of data. As for data model, we present a new model, *Free-Table*, which manages unstructured data by *cell*. Cell is not same as traditional data type or object. It is a virtual by-instance type, that is, any instance of an attribute in an FT has peculiar format of its own, and operators. In addition, cell supports data versioning.

Two cases, VMP and PXRDB, show that BUD with FT is feasible. VMP is an instance of BUD to manage surveillance video data while PXRDB is a DBMS adaptively managing XML data. VMP supports various type of video and image data stored in cells witch specific elementary operators. As to PXRDB, its XML type can be as a cell, we believe PXRDB can provide the capabilities of BUD.

As we said, BUD is reference framework based on FT. More open issues will emerge and need more efforts to solve them with FT technique maturing.

Acknowledgment

Thanks so many to our colleagues and students for discussion with and explanation to them.

References

[1] Rakesh Agrawal and etal,*The Claremont Report on Database Research*, http://db.cs.berkeley.edu/claremont/claremontreport08.pdf, 2008

[2] A. Szalay and J. Gray. *Science in an exponential world*. Nature, 440, March 23 2006.

[3] E. Chu, J. Beckmann, J. Naughton. *The Case for a Wide-Table Approach to Manage Sparse Relational Data Sets*. SIGMOD 2007

[4] Fay Chang, Jeffrey Dean, et al. *Bigtable: A Distributed Storage System for Structured Data*. Google. OSDI 2006

[5] E. Chu, J. Naughton, et al. *A Relational Approach to Incrementally Extracting and Querying Structure in Unstructured Data.* VLDB 2007

[6] Shawn R. Jeffery, Michael J. Franklin, Alon Y. Halevy. *Pay-as-you-go user feedback for dataspace systems*. SIGMOD 2008

[7] Warren Shen, Pedro DeRose, Robert McCann, AnHai Doan, Raghu Ramakrishnan. *Toward best-effort information extraction*. SIGMOD 2008

[8] J. L. Beckmann, and J. F.Naughton, et al. *Extending RDBMSs to support sparse datasets using an interpreted attribute storage format*. ICDE 2006.

[9] Wenjing Zhou, Xiangwei Xie, Hui Li, Xiao Zhang, Shan Wang, *A Database Approach for Accelerating Video Data Access*, APWeb/WAIM WCMT Workshop 2009

[10] Gaurav Bhalotia, Arvind Hulgeri, et al. *Keyword Searching and Browsing in Databases using BANKS*. ICDE 2002

[11] B. Yu, G. Li, B. Ooi, L. Zhou. *One Table Stores All: Enabling Painless Free-and-Easy Data Publishing and Sharing*. CIDR 2007

[12] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, *Evaluating probabilistic queries over imprecise data*, SIGMOD 2003