Front. Comput. Sci., 2016, 10(2): 353–369
DOI 10.1007/s11704-015-5045-6

RESEARCH ARTICLE

D-Ocean: an unstructured data management system for data ocean environment

Yueting ZHUANG, Yaoguang WANG, Jian SHAO (✉), Ling CHEN, Weiming LU, Jianling SUN, Baogang WEI, Jiangqin WU

College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China

Abstract Together with the big data movement, many organizations collect their own big data and build distinctive applications. In order to provide smart services upon big data, massive variable data should be well linked and organized to form *Data Ocean*, which specially emphasizes the deep exploration of the relationships among unstructured data to support smart services. Currently, almost all of these applications have to deal with unstructured data by integrating various analysis and search techniques upon massive storage and processing infrastructure at the application level, which greatly increase the difficulty and cost of application development.

This paper presents D-Ocean, an unstructured data management system for data ocean environment. D-Ocean has an open and scalable architecture, which consists of a core platform, pluggable components and auxiliary tools. It exploits a unified storage framework to store data in different kinds of data stores, integrates batch and incremental processing mechanisms to process unstructured data, and provides a combined search engine to conduct compound queries. Furthermore, a so-called RAISE process modeling is proposed to support the whole process of Repository, Analysis, Index, Search and Environment modeling, which can greatly simplify application development. The experiments and use cases in production demonstrate the efficiency and usability of D-Ocean.

Keywords unstructured data, storage, analysis, index, search, RAISE process modeling

1 Introduction

Nowadays, data is being generated at an astounding rate by everything around us at all times. Together with the big data movement, many organizations try to collect their own big data and build distinctive applications [1]. Due to the *variety* characteristic of big data [2], these applications have to deal with unstructured data, such as texts, images, videos, and audios, whose content does not have a specific, pre-defined data model.

Taking digital library application as an example, today, the application maintains not only a great amount of digital books, but also enrichments in various forms, e.g., pictures, calligraphy, paintings, as well as the relationships between them. NoSQL databases (e.g., HBase and MongoDB[1]) and distributed computing frameworks (e.g., Hadoop and Storm[2]) are adopted to build massive storage and processing infrastructure. Various unstructured data analysis and search techniques (e.g., UIMA [3], Solr[3]) are integrated to support complex content based queries. In this way, massive variable data are well linked and organized to form *Data Ocean* [4], which specially emphasizes the deep exploration of the relationships among unstructured data to support smart services,

Received January 29, 2015; accepted June 12, 2015

E-mail: jshao@zju.edu.cn

[1] http://hbase.apache.org/, and http://www.mongodb.org/
[2] http://hadoop.apache.org/, and https://storm.incubator.apache.org/
[3] http://lucene.apache.org/solr/

e.g., knowledge discovery, cross media retrieval, and personalized recommendation.

However, due to the lack of system support, almost all of these applications have to deal with unstructured data at the application level (integrating various unstructured data analysis and search techniques upon massive storage and processing infrastructure), which greatly increase the difficulty and cost of application development. Additionally, such solutions cannot well support the representation of links between data, which is crucial to form Data Ocean.

Among these big data related applications, there is a great need of unstructured data management system (UDMS), which can store and use all forms of unstructured data. To the best of our knowledge, there is no formal definition for UDMS. In this paper, we define UDMS as a fundamental software platform for managing unstructured data, with core functionalities of storage, analysis, index, and search.

In the environment of Data Ocean with the issues beyond volume, variety, and velocity of big data to the special emphasis on the deep exploration of the relationships among unstructured data to support smart services, several requirements are imposed over each functionality in a UDMS, which are listed as follows:

1) Storage Requirement

The underlying storage infrastructure should be scalable and agile to deal with large volume of data and various types of data. Specifically, it should support different storage requirements of unstructured data, including raw data, low-level features, and semantic features. In addition, in order to adapt to the quick changes of unstructured data, the storage should support evolution of the repository model over time and extensibility of new types of unstructured data.

2) Analysis Requirement

The system should provide a set of analysis tools for deriving value from various types of unstructured data. Besides, the system should not only execute analysis tasks in an incremental manner to deal with growing datasets, but also need to re-analyze the overall datasets using batch processing. Moreover, the analysis tools also need to be customized by users for the various types of data and analysis tasks.

3) Index Requirement

The types of features of unstructured data are manifold. It may be primitive, such as numerical value and strings, or high-dimensional vectors from extracted features. In order to efficiently retrieve unstructured data according to different features, the system should not only support several index types, such as B+tree index, inverted index, and hash index, etc., but also support index customization.

4) Search Requirement

Typical searching services, such as range query, keyword query, exemplar query, and cross-media query, etc., and the compound query, a combination of all or some of the above simple queries, should be provided. In addition, the search strategies, such as filtering, ranking, merging and feedback, should be customized.

5) Other Requirements

Several easy-to-use tools for system management and monitoring should be provided. In addition, programming interfaces and language are quite valuable for system development. Specifically, if the system can model the whole process of unstructured data storage, analysis, index and search, it could also reduce the complexity of application development.

Motivated with above requirements, we present D-Ocean, an unstructured data management system for data ocean environment, which has the following technical virtues:

- A data ocean oriented open architecture with the ability of extensible data type, storage, and processing for different applications.
- A unified storage framework, which integrates different kinds of data stores to support efficient storage of both structured and unstructured data.
- A generalized processing framework, which supports both batch processing and incremental processing under a unified task management.
- A set of pluggable unstructured data process elements, which supports analysis, index, and simple search, as well as a combined search engine for compound queries on diverse data types.
- A so-called RAISE process modeling method, which supports the whole process of Repository, Analysis, Index, Search and Environment modeling, as well as a SQL-like Unstructured Query Language UQL to streamline application development.

The rest of this paper is organized as follows. Section 2 introduces the previous work relevant to our D-Ocean system. Section 3 describes the architecture and design of D-Ocean, whose details are then elaborated in Section 4. In Section 5 and Section 6, we present experimental results and use cases in production obtained from the current system snapshot and applications. We conclude in Section 7.

2 Related work

Our work related to a number of areas such as unstructured data model and query language, massive data storage and processing, unstructured data analysis and search.

2.1 Unstructured data model and query language

Unstructured data model and query language have been studied both in the multimedia field and the database field. The MPEG-7 standard [5] provides a rich set of multimedia description schemes, which can be used to describe various types of multimedia data and their complex relationships. Aiming at facilitating and unifying access to multimedia data, MPEG Query Format [6] is proposed as a part of MPEG-7 standard, which is essentially an XML-based query language, however, it appears to be too complicated for SQL users. SQL/MM [7] introduces structured user-defined types and associated methods for texts, images, and spatial data. UnQL [8] is proposed to query semi-structured data, which mainly focuses on the tree abstractions for structured and unstructured data sources. These earlier works give inspiration to the design of our object based unstructured data model and SQL-like query language for D-Ocean.

Dataspace [9] is proposed to manage a large number of diverse and interrelated data sources, which could support data model to be evolved in a “pay-as-you-go” fashion. It is implemented in iMeMax Data Model (iDM) [10], which uses resource views that linked to each other in arbitrary directed graph structure. The data model of D-Ocean also supports the evolution, e.g., adding new features to an existing data model to meet the requirements of new analysis activities.

2.2 Massive data storage and processing

Great changes are taking place in the massive data storage and processing fields. Data storage is changing from SQL databases to NoSQL, NewSQL [11], and multistore systems [12]. Data processing is shifting from batch processing [13] to streaming processing (e.g., Storm), iterative processing [14], and hybrid processing systems (e.g., Summingbird [15]).

From the perspective of data-intensive applications, it is desirable to integrate storage and processing infrastructures together to reduce the cost of data transfer. Hadoop is a representative distributed computing system that incorporates a distributed file system and MapReduce processing framework, which can handle the data volume challenge successfully. However, Hadoop does not deal with the data variety well due to the limited programming interfaces and data processing model. An extensible and scalable system epiC, which integrates a file system federation, an elastic storage system, an elastic execution engine and a flexible query language [16], is proposed to address the data variety challenge. So far many big companies provide all kinds of storage and computing products. Some of them deploy these products in the cloud and users have to pay for what they use, such as Goolge Cloud Platform and Baidu Open Cloud[4)], while others provide free products and sell services, like Hortonworks and Pivotal[5)]. D-Ocean is designed to free developers from all these different tools and simplify application development. D-Ocean is designed with an open and scalable architecture, which integrates various data stores and processing frameworks with Hadoop. Specially for unstructured data, a RAISE process modeling method is proposed to free developers from all these different tools and reduce the complexity of application development.

2.3 Unstructured data analysis and search

Analysis and search techniques for unstructured data have been widely investigated, such as natural language processing for information retrieval [17], content based multimedia retrieval [18], and complex event processing for data streams [19]. Accordingly, a lot of tools and services for unstructured data analysis and search are developed. Many open source tools can be readily used to set up our own server or cluster for processing, such as SOLR, LIRE [20], and Cayuga [21]. In addition, some web services also can be invoked for remote processing over cloud, such as AlchemyAPI and Amazon CloudSearch[6)].

However, due to various processing interfaces and running environment requirements, it is not easy for developers to integrate above tools and services in one system. Researchers concentrate more on designing a framework to solve the generalized unstructured data analysis and search problems. They take reliability, scalability and processing efficiency into their consideration and to satisfy the processing for large-scale unstructured data. UIMA [3] is a framework that focuses on providing unstructured data analysis services to the end users. SAPIR[7)] is an unstructured data search framework providing the audio-visual content search solution under peer-to-peer environment. Compared with UIMA and SAPIR, our D-Ocean is a more generic framework support-

4) https://cloud.google.com/, and http://bce.baidu.com/
5) http://hortonworks.com/, and http://pivotal.io/big-data/pivotal-big-data-suite
6) http://www.alchemyapi.com/, and http://aws.amazon.com/cn/cloudsearch/
7) http://sysrun.haifa.il.ibm.com/sapir/

ing both the analysis and search for unstructured data. Recently, AsterixDB [22] has been developed as an open-source platform. The goal of AsterixDB is to ingest, manage, index, query and analyze mass quantities of semi-structured data, which is quite similar to D-Ocean. However, they have different design decisions. AsterixDB provides a semi-structured data model and an expression language, while D-Ocean exploits SQL-based query language and supports plugins management and task status management for unstructured data. The important motivation of D-Ocean is to provide a full-function UDMS and simplify application development on different kinds of unstructured data.

3 D-Ocean architecture and design

In this section, we present the architecture of D-Ocean, the data model, the query language, and the design philosophy of architectural extensibility.

3.1 D-Ocean architecture

The D-Ocean system is designed to provide fig:3a comprehensive unstructured data management solution. Figure 1 illustrates the architecture of D-Ocean, which consists of three layers: a tools layer, a core layer and an infrastructure layer.

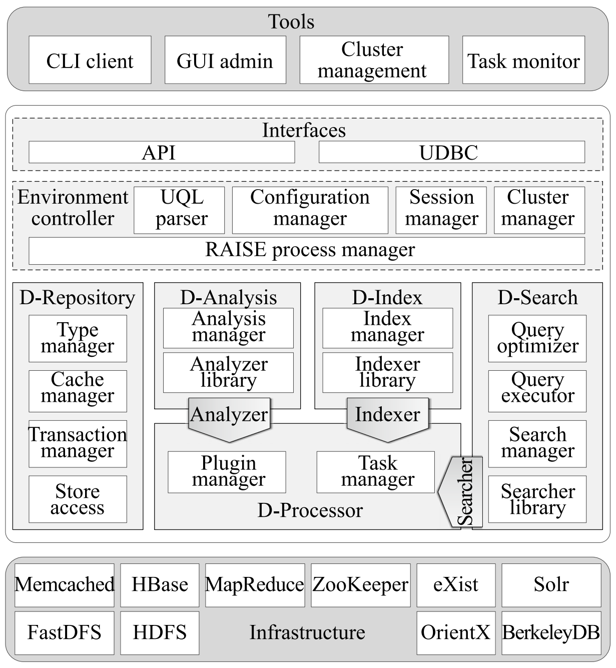


Fig. 1 The D-Ocean architecture

The **tools layer** offers users to manage and monitor the whole system via auxiliary tools, e.g., command line interface (CLI) client and graphical user interface (GUI) administration. A *cluster management tool* is used to deploy and maintain the system, while a *task monitor* is in charge of task status.

The **core layer** comprises of seven major modules: Interfaces, Environment controller, D-Repository, D-Analysis, D-Index, D-Search and D-Processor.

To interact with the underlying system, users can utilize either any above-mentioned tools or two kinds of interfaces, *API* and *UDBC* (UDMS Database Connectivity). We provide java APIs for modeling, storing and analyzing unstructured data, while the entire functionalities can be obtained via UDBC. The key part of UDBC is a SQL-like unstructured query language, UQL. The environment controller module handles all the activities between developers and the system. For instance, the *session manager* maintains user connections and the *UQL Parser* is responsible for parsing UQLs. The *configuration manager* takes charge of environment configurations, e.g., batch job configuration and cluster configuration. The *cluster manager* collects the cluster status and sends to the cluster management tool. Meanwhile, it receives commands and controls the server status. The most important module is the *RAISE process manager* which takes charge of all requests. If the request is clear and concise, it is delivered to the appropriate module directly. Otherwise, the RAISE process manager encapsulates required configurations for process modeling. The details will be introduced later.

Data are fed into the system via the D-Repository module which provides CRUD operations to manipulate data objects. These operations are organized as a unified interface for further data processing in D-Processor. According to the process modeling, D-Processor downloads and instantiates needed plugins from the corresponding plugin libraries, fetches data from repository and executes the tasks in parallel. Currently, plugins are divided into three categories including *Analyzer*, *Indexer* and *Searcher*, maintained in D-Analysis, D-Index and D-Search, respectively. Once analysis and index work is completed, D-Search supports unstructured data search, including a wide range of simple queries and compound queries.

The **infrastructure layer** provides necessary storage and computing infrastructure to the upper layers. As for storage infrastructures, we now integrate various kinds of data stores, such as NoSQL databases (e.g., HBase) and distributed file systems (e.g., HDFS and FastDFS[8]). As for computation in-

[8] http://hadoop.apache.org/, and http://code.google.com/p/fastdfs/

frastructures, a Hadoop MapReduce cluster is deployed. In addition, indexing service (e.g., Solr) and coordination service (e.g., ZooKeeper[9]) are also supported.

3.2 Data model

The complexity of unstructured data is in two folds. First, unstructured data has rich semantic info and features, which are extracted from raw data. Second, there are complicated relationships between unstructured data objects, e.g., nesting and inheritance. Take image as an example, after analyzing an image, we get color features, shape features, semantic info, etc. Images can be classified into different categories, e.g., people, cars, aeroplanes, etc. Those categories may have inheritance relations. Traditional data models are hard to fully describe these info, thus an unstructured data model is proposed for D-Ocean.

Figure 2 shows the data model of D-Ocean. In D-Ocean, every instantiated unstructured data is called a UObject. A UObject is uniquely identified by its ID. The ID can be generated by the system, or concatenated by user-specified primary key(s). The ID allows users to quickly get the object. UObject contains one or more Features, which contain values. UObject and Feature have corresponding UType and FeatureType, which are described as follows.

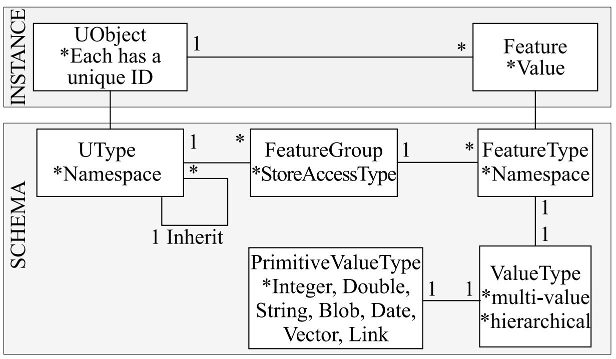


Fig. 2 D-Ocean data model

1) UType

A UType is a named set of FeatureGroups, and has a qualified name consisting of namespace and an actual name. Each UObject is associated with an UType. Some UTypes have inheritance relations which describe the relationship between two types of UObjects, e.g., a teacher is one kind of person and an image crawled from the Internet is a special image. For better description of these relations, D-Ocean supports inheritance functionality. Thus, a UType Teacher is defined by inheriting UType Person and UType WebImage inherits UType Image. If an UType B inherits from UType A, B will have all FeatureGroups that A has.

2) FeatureGroup

A FeatureGroup consists of FeatureTypes which share the similar characteristics: they are always read and written together. FeatureGroup is proposed to make better use of this characteristic. A FeatureGroup has a name, one or more FeatureTypes, and StoreAccessType. StoreAccessType defines where to store the features, e.g., key-value stores, file systems, and document stores. Features in the same FeatureGroup will be stored closely in the same storage engine, which can improve I/O performance.

3) FeatureType

Each FeatureType has a Namespaced name and a ValueType. The reason we propose the concept of FeatureType is that we want make it reusable: different UTypes can have same FeatureTypes. For example, A FeatureType named ImageBlob, can be used by an UType named Image, and an UType named Keyframe. The design of FeatureType improves system's reusability, and users do not need to create similar things over and over.

4) ValueType and PrimitiveValueType

The built-in types of D-Ocean consist of ValueType and PrimitiveValueType. PrimitiveValueType includes Integer, Double, String, Blob, Boolean, Vector, Date, DateTime, and Link. Link type is used to store the relationships between UTypes, and it can also support delink, the opposite direction of link. A ValueType can be defined by specifying a PrimitiveValueType and an option of singleton, hierarchical, or multi-value. Both hierarchical and multi-value options can be used to support multiple values. The difference is that hierarchical can maintain the order while multi-value cannot. For example, to ValueType phone numbers, the order of them is not important, while to ValueType paper authors, the order of them is important. Thus, the ValueType phone numbers should be multi-value and the ValueType paper authors should be hierarchical.

Note that, D-Ocean does not support the join operation for the performance consideration. Instead, the PrimitiveValueType Link and the hierarchical/multi-value options are employed to store the 1:N relationship between UTypes. It is important for applications. For example, we extract key frames from videos. They are represented as two UTypes, Video and Keyframe. The UType Video has a feature named *keyframes*, whose PrimitiveValueType is Link (with hierarchical option and links to the UType Keyframe). When users get a video of

9) http://zookeeper.apache.org/

UType Video, they can quickly get all its key frames by the feature of Link type, avoiding the join operation.

3.3 Query language

Relational databases usually use SQL to manage data. However, SQL cannot deal with unstructured data well. Similarity search is quite popular to unstructured data. For example, users input a text file and want the system to return similar text files, ordered by the similarity score. SQL cannot support such kind of query, which constrains its application in UDMS. Therefore, we design a SQL-like query language: unstructured query language (UQL). Users can use it to develop applications conveniently, instead of invoking complex APIs. UQL has simple grammar form, complete functionalities and strong query support. The functionalities of UQL are divided into three groups:

1) Model definition

Model definition contains four categories that interact with models: Storage, Index, Analysis, and Import/Export. Storage UQLs manage Namespace, FeatureType, and UType. Index UQLs manage index definitions, while analysis UQLs manage analysis definitions. Import/Export UQLs are used to load/save models from/to XML files.

2) Data management

Data management UQLs provide CRUD operations. The most important feature is data query, which includes sorting, aggregation, condition query, XML query, similarity search, and full-text retrieval. Note that, the join operation in D-Ocean is not supported and replaced by link instead. The provided UQLs are as follows:

- Support results ordering by some features;
- Support restriction on the size of results with keyword TOP or LIMIT;
- Support basic aggregation functions with keyword AVG, COUNT, MAX or MIN;
- Support text search with keyword CONTAIN;
- Support exemplar query with keyword LIKE;
- Support XML query with keyword XQUERY;
- Support relevance feedback with keyword FEEDBACK;
- Support search with link. We use "->" to search for linked UType. For example, "*select from Video where keyframes->image like @'path_to_the_pic'*" is a UQL query for videos whose key frames are similar with the provided picture. In this UQL statement, the UType Video stores videos' info like name, video blob, size, keyframes, etc. The Feature named *keyframes* is a Link type that links to a UType Keyframe, which has a feature named *image*.

3) Environment control

Environment control UQLs not only support traditional data control languages, such as user's privileges, roles, audit and transactions, but also operations on plugins management and task status management. We give several examples of the latter as follows:

- Support plugins management with keywords ADD/DELETE PLUGIN ANALYSIS/INDEX;
- Support task status update with keywords UPDATE ANALYSIS ACTIVE/DISABLE.

3.4 Extensibility

As big data solutions are constantly emerging, we argue that it is of great significance to make a system evolve gracefully. At the beginning of D-Ocean, we took a lot of effort to design an open, scalable and extensible architecture. We present three main aspects of the extensibility of D-Ocean as follows.

1) Extensible type

In the data model of D-Ocean, two extensible types are FeatureType and UType. First, developers may multiplex the existing accessible FeatureTypes or define their specific FeatureTypes. The features with the same FeatureType can be processed in the same way. Second, a new UType can be declared by inheritance from other UTypes.

2) Extensible storage

To manage structured data and binary objects, we define two different interfaces, namely *StoreAccess* and *BlobStoreAccess*. The former is responsible for the majority of value types except the blob type, which is handled by the latter. It is easy to replace or enhance the existing storage components, in which the other modules and data flow are minimally or not affected.

3) Extensible processing

As mentioned before, D-Processor utilizes three kinds of plugins to execute incremental and batch tasks. We design unified interfaces for each. These interfaces enable users to extend the D-Processor's capabilities.

3.5 Other features

As a distributed unstructured data management system, D-Ocean has several fundamental features. Some of them are shown as follows.

1) High availability

High availability usually signifies that how to guarantee the data is accessible and service is available when failure happens. In order to protect data, we leverage data replication techniques of some storage engines, such as HDFS and FastDFS. If MySQL is specified as the primary data store, we also use HBase as a secondary data store, which will be accessed only when the primary store has no response. When uploading a large binary object, we exploit a reference table to first log the identifier of UObject which the object belongs to. The object is visible after it is completely persisted and the log in the reference table is removed.

2) Fault tolerance

We discuss three kinds of failures in D-Ocean. If no response is received from a D-Ocean server, the set of alive nodes maintained in Zookeeper is automatically updated and the client needs to reconnect to D-Ocean. For worker failure during task processing, it is different for two processing modes. In the batch mode, the job needs to be re-executed in the presence of failures. Note that it is correct to re-execute the analysis task by overwriting the analyzed results, while the incomplete indexing results have to be removed before re-execution. In the incremental mode, we use a task log to handle failures. All failed UObjects with reasons are persisted and used for manual re-execution at the granularity of UObject instead of the whole task. Finally, data node failure is masked by data replication techniques.

3) Data consistency

In most cases D-Ocean leverages single-row atomicity of HBase to guarantee consistency except the Blob data. As mentioned before, we use a reference table to avoid inconsistent binary objects. In extreme cases, we need consistency across multiple UObjects. D-Ocean exploits a strict two-phase locking protocol to guarantee transactional multi-row access. The transaction cost is also reduced by an optimized single-phase commit protocol [23].

4 D-Ocean implementations

In this section, we present the implementations of D-Ocean. Specifically, we show how we implement a unified storage management and how to integrate two types of data processing modes on top of it. After that, we describe how we perform search on unstructured data. Finally, we detail the RAISE process modeling.

4.1 Unified storage management

D-Repository is responsible for receiving clients' requests and providing a unified storage framework over different kinds of data stores [24, 25]. Figure 3 shows the major aspects of a UDMS node and the unified storage framework.

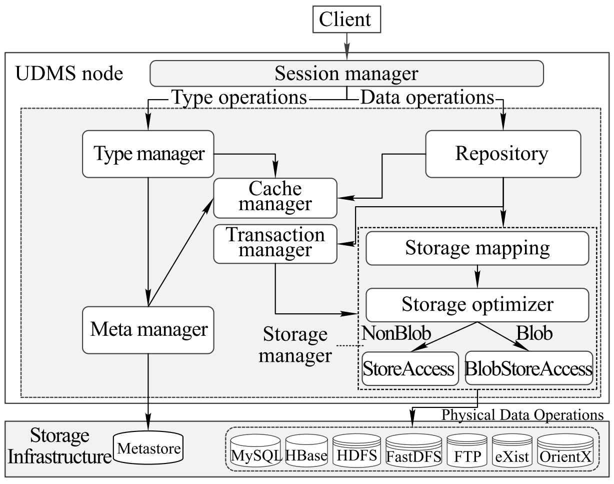


Fig. 3 The unified storage framework of D-Repository

There are two types of operations to be handled by D-Repository: *type operations* and *data operations*. After a connection is established, these two types of clients' requests are forwarded to the *type manager* and *repository*, respectively. The type manager takes charge of type definition and maintenance, including UType, Namespace, FeatureType, etc. Since UTypes are frequently accessed, the *cache manager* keeps the consistent structures of UTypes in memory. The *meta manager* is used to flush the type information into persistent storage, which is currently implemented by a key-value store and other databases are easy to replace it. As a significant submodule of D-Repository, the repository handles all data operations. It acquires logical structures of UTypes from the cache manager, delivers the operations to the *transaction manager*. The transaction processing module [23] in D-Ocean exploits a strict two-phase locking protocol to guarantee transactional multi-row access. It also reduces the transaction overhead by an optimized single-phase commit protocol. In fact, the transaction manager handles transaction processing logic, such as lock and log management, while data operations are routed to the *storage manager*. In addition, repository will deliver the operations directly to the storage manager in case of single row operation or the disabled transaction capability.

The storage manager plays a critical role in D-Repository. Only it knows the storage decisions that where the UObjects are placed. The storage mapping component is responsible for mapping features to key-value stores or relational databases since multiple data stores are allowed to co-exist. Administrator is able to specify the mapping rules through scripts or implementing a pluggable interface. In addition, the storage optimizer is used for choosing the best data store to place data. We provide two dedicated access interfaces for structured and unstructured data storage. The features with blob type are handled by the *BlobStoreAccess*, otherwise *StoreAccess*. Both of them have several implementing classes, each of which is used for executing physical data operations with one data store. All implementing classes are pluggable and configurable, making the whole storage framework extensible to integrate new data stores. All in all, clients have a unified storage interface, the repository, regardless of the heterogeneity of underlying data stores.

4.2 Generalized processing framework

D-Processor provides a run-time environment in which developer can plug in their process elements and with which they run processing tasks in both incremental mode and batch mode. As shown in Fig. 4, it uses a master-slave distributed computing as its parallel implementation. Each UDMS node has a *plugin manager* and a *task manager*.

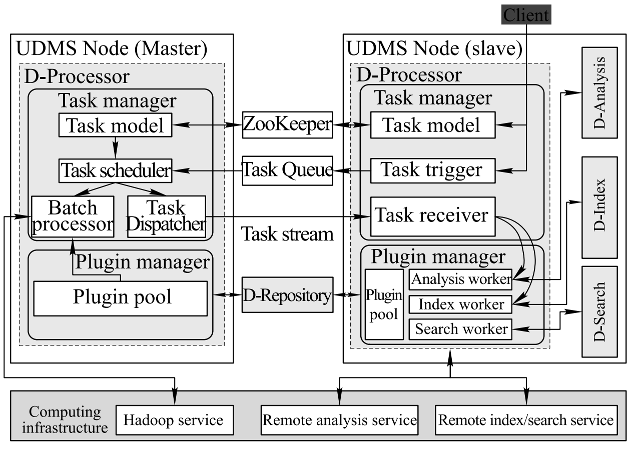


Fig. 4 The generalized processing framework of D-Processor

The plugin manager is responsible for uploading, deleting and instantiating process elements. Three types of process elements are defined in D-Processor: *Analyzer*, *Indexer* and *Searcher*. Figure 5 shows unified interfaces for each category.

<<Interface>> Analyzer
Attributes: -AnalysisDefinition definition

Methods:
+init()
+analysis(UObject object)
+analysis(Collection <UObject> objects)
+clean()
...

<<Interface>> Indexer
Attributes: -IndexDefinition definition

Methods:
+add(UObject object)
+add(Collection <UObject> objects)
+delete(String uid)
+update(UObject object)
+commit()
+clean()

<<Interface>> Searcher
Attributes: -SearchDefinition definition
Methods: + search(QueryNode query, ResultList results)

Fig. 5 The unified interfaces for Analyzer, Indexer and Searcher

The Analyzer is used for processing unstructured data to extract structured information from low-level features to high-level semantics. It takes UObjects as input and writes back the analyzed results into D-Repository. It can be run in local UDMS nodes or remote analysis services which are managed by D-Analysis. Based on the unified Analyzer interface, numerous concrete analyzer plugins are provided in the library, such as DocumentAnalyzer, ImageAnalyzer, AudioAnalyzer and VideoAnalyzer. Furthermore, a variety of unstructured data analysis techniques, such as natural language processing, multimedia content analysis, topic discovery, can be made use of by choosing appropriate derived classes for further inheritance and implementation.

The Indexer is used for building index for unstructured data. It takes UObjects from D-Repository and puts them into remote indexing services (e.g., Solr) which are managed by D-Index. With a unified Indexer interface shown in Fig. 5, inverted indexer for full-text, b-tree indexer for numeric value, m-tree indexer for multi-dimensional feature, and spectral hashing (SH) indexer for high-dimensional feature have been derived and implemented.

The Searcher is used for retrieving of unstructured data. It receives simple queries from D-Search, then directly conducts search on remote searching service, and returns intermediate result list to D-Search. Similarly, with a unified searcher interface shown in Fig. 5, we have provided full-text searcher, exemplar searcher and numeric searcher.

The task manager is responsible for task definition, scheduling, execution and monitoring. Once task definitions are added to the *task model* in any UDMS node by clients, all other nodes in the cluster will be notified and keep their task models in sync by Zookeeper. The *task scheduler* in master node is in charge of task scheduling according to the defined running mode: batch processing or incremental processing.

For batch processing, the master node will prepare and submit tasks to Hadoop cluster on its own. At first, the *batch processor* in master node gets the added task definition from the task model, and splits the task into several logical partitions for the use of Hadoop. Then, the split information expressed by the start and end IDs of UObjects, together with the task definition information are packaged and submitted to Hadoop Cluster, and plugins defined in task definition will be instantiated in each mapper and used to process the data in D-Repository. For task monitoring, a daemon thread in master node is started to watch the execution status.

For incremental processing, the master node first picks several slave nodes and notifies them to build a message queue based real-time task processing pipeline. After this, once a

UObject is inserted, updated or deleted in a node, the *task trigger* in that node will trigger a task message including UObject ID and definition ID, and deliver it to *task queue*. At the same time, the *task dispatcher* will get task messages from task queue and dispatch them to the chosen nodes where corresponding plugins have been already instantiated at worker nodes, and the workers can consume task messages by the *task receiver* one by one. The aforementioned *task monitor tool* is used to watch the execution status.

In addition, we also adopt some advanced parallel computing techniques for incremental processing. For a collection of UObjects, users may submit multiple task definitions on them. RAISE process manager constructs a directed acyclic graph (DAG) (Storm and Dyrad [26]) according to these definitions. Following that, tasks are distributed to a cluster of workers with different roles to execute analysis or index tasks. In order to deal with changing workload, the number of workers and their roles can be tuned dynamically.

4.3 Combined search engine

Our designed UQL not only supports simple query for UObjects by single feature type, such as keyword query over free text, exemplar query over multimedia content, and range query over numeric property, but also supports compound query for UObjects by multiple features where the compound query is a combination of all or some of the above simple queries, as well as feedback query and link query. A combined search engine, which consists of Query Optimizer and Query Executor in D-Search as shown in Fig. 6, is implemented for UQL query optimization and execution.

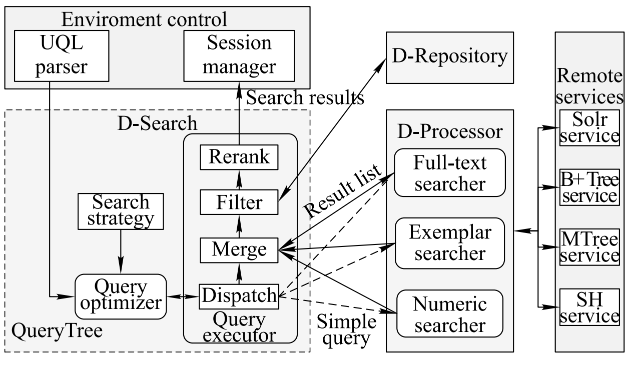


Fig. 6 Combined search engine in D-Search

The *UQL parser* receives UQL requests and transforms them into a unified xml-style search format by extending the standard MPEG-7 query format [6] for unstructured data objects. The search format encodes three kinds of information, which are further organized as an original query execution plan using a *QueryTree* structure with three types of nodes: root, branch, and leaf.

- Root node contains search environment information about the whole query, such as Namespace name and UType name, the type of aggregation function if exists and the weights of sub-results from sub-queries.
- Branch node contains search hierarchy information, which describes relationships among sub-queries including AND, OR, Weighted-AND and Weighted-Sum.
- Leaf node contains search execution information, which describes each concrete sub-query including the feature(s), the operator, the search value, the upper and lower bound of the range query, the keyword of the full text query or the example of the exemplar query, etc. This information can be delivered to search workers in D-Processor for actual search.

The *query optimizer* gets the original query plan tree and re-organizes it through two sets of optimization rules. One is the traditional database optimization rules, such as conjunctive selection, commutative selection, conjunctive projection and commutative projection [27]. Another is the heuristic rules predefined by D-Ocean system or specially added by application developers, such as the grouping of keyword queries together when they use the same remote searching service, the grouping of exemplar queries on multiple features when these features use a hybrid index and user-defined search hierarchy by exploiting the prior knowledge of application.

The *query executor* adopts the divide-and-conquer scheme to conduct a hierarchical invocation of dispatching and merging actions to get merged result list. Although the indexing techniques are adopted, the execution time for each simple search still varies dramatically. For example, the query time for a keyword-based full-text search may be hundreds of times faster than the one for content-based image search. Therefore, the filter strategy is used to accelerate the combined search by executing the short-time search first to get a candidate result list and afterwards executing the long-time search within this candidate list. After that, the filtered result list will further re-ranked. D-Search provides some optional re-ranking strategies, such as the simple threshold algorithm, no random access algorithm [28]. Besides, in order to exploit the various relationships among UObjects, a complex re-rank strategy based on hypergraph manifold ranking is also integrated in our system [29].

4.4 RAISE process modeling

Traditional databases only provide repository modeling.

However, due to the complexity of unstructured data, repository modeling alone cannot satisfy all the needs. Thus, we propose the RAISE process modeling, which supports whole process modeling, including Repository, Analysis, Index, Search, and Environment. One of the advantages of RAISE is that each sub-process couples loosely, such that users can organize each sub-process flexibly according to different requirements. The details of each modeling process are shown as follows.

1) Repository modeling

Repository modeling is based on the data model presented in Section 3. Also, referring to MPEG-7 [5], we build an unstructured data type library which contains types of text, image, video, audio, etc., which can speed up repository modeling.

The results of repository modeling are outputted as XML files. Figure 7 gives an example, the repository model of Image. Here we define an UType named Image, which has features *height*, *width*, *encoding*, *cedd*, and *image*. The first four are in a FeatureGroup named *basic*, while the other one is in a FeatureGroups named *file*. In the feature tag, we specify the feature's type, while in featureGroup tag, we specify the StoreAccessType. If we want to create an UType Keyframe, which has similar features with Image, we can create it as the inheritance of Image, as shown in Fig. 8.

```
<UType name=”Image”>
    <feature name=”height” featureType=”Integer”/>
    <feature name=”width” featureType=”Integer”/>
    <feature name=”encoding” featureType=”String”/>
    <feature name=”cedd” featureType=”CEDDVector”/>
    <feature name=”image” featureType=”ImageBlob”/>
    <featureGroup name=”basic” storeAccessType=”HBase”>height,
width, encoding, cedd</featureGroup>
    <featureGroup name=”file” storeAccessType=”HDFS”>image</
featureGroup>
</UType>
```

Fig. 7 Repository modeling for Image

```
<UType name=”Keyframe”>
    <inherit>Image</inherit>
    …
</UType>
```

Fig. 8 An example of Inheritance

2) Analysis modeling

In order to extract features from UObjects, we need to do analysis modeling. Analysis modeling is based on UType, and defines how unstructured data objects should be analyzed. Figure 9 shows an instance of analysis modeling for Image. Here we give a unique definition ID, and specify which plugin to use and what UType to analyze. We set *image* as the input feature and the analyzed results will be written back to feature *cedd*.

```
<AnalysisDefinition>
    <DefiniitonID>Analysis_CEDD_Definition</DefinitionID>
    <PluginName>GenericImageExtractor</PluginName>
    <UTypeName>Image</UTypeName>
    <InputFeatureNameMap>
        <InputFeature>image</InputFeature>
    </InputFeatureNameMap>
    <OutputFeatureNameMap>
        <OutputFeature>cedd</OutputFeature>
    </OutputFeautreNameMap>
</AnalysisDefinition>
```

Fig. 9 Analysis modeling for Image

3) Index modeling

Index modeling is fairly similar with analysis modeling, while it is used to index data for the purpose of improving search performance. The only difference of analysis definition is that index definition does not have OutputFeatureNameMap tag. Instead, it needs to specify the index server's URI and what index method to use in another PluginProperties tag.

4) Search modeling

Search modeling relies on analysis modeling and index modeling, and defines the evaluation process of data search and display. A search model definition contains three parts:

- Search strategy: depict the strategy used to search, e.g., preference to different conditions, and search order.
- Ranking of search result: define how to order the search results, e.g., when ordering the results with conditions of full-text and properties, which is difficult in unstructured data search.
- Relevance feedback: model the users' evaluation of the search results.

5) Environment modeling

Environment modeling describes the execution environment and execution process of the above four models. The execution environment mainly focuses on data processing of analysis and index tasks, such as the processing mode, like incremental or batch, and degrees of parallelism. For batch processing, the partial collection of UObjects can be skipped by specifying the start point in time. The execution process indicates the processing phases on storage, analysis and index. For example, raw data may be abandoned and only analyzed or cleaned data is stored in D-Ocean. The index process may be executed immediately after the analysis process or done later in a batch mode.

5 Experimental evaluation

In this section, we study the performance of D-Ocean in three parts: storage, analysis and search. The first part is to evaluate the scalability and performance of the unified storage framework in terms of data loading, the methodology of which is similar to the methodology in Ref. [30]. The second part measures the performance of our self-developed incremental data processing component, showing the throughput in terms of different number of workers when analyzing three typical types of unstructured data. Following that, we compared the processing time of the incremental mode with the integrate batch mode. The last part gives the experimental results of different queries on a image data set, including simple queries and compound queries. The data set has been analyzed and indexed using SH and inverted index on corresponding features. More design details are given later in corresponding subsections. For each experiment in this section, we performed 3–10 runs.

• Cluster setup We evaluate the D-Ocean on our two separated clusters: 1) Cluster-I, which consists of 24 nodes. Each node is equipped with quad-core Intel Xeon 3.10GHz CPU, 7GB memory, 250GB SATA disks. The storage experiments were executed in Cluster-I. 2) Cluster-II, which has ten nodes. Each node is Intel Xeon CPU (2.80GHz, 8 cores) with 48 GB memory and 2TB SATA disks. We conducted analysis tasks and queries in Cluster-II. Each of cluster nodes is connected with 1Gbps Ethernet network and running Ubuntu Linux 12.04. We use Hadoop version 1.2.1 and HBase version 0.94.13 running on Java 1.6.0, and FastDFS version 4.05. Regionservers, Datanodes, StorageTrackers, TaskTrackers and D-Ocean servers were co-deployed on the same machines, while masters were run on one node in the cluster. To ensure data availability, we set the replication factor of three for HDFS and configure three storage trackers in each group for FastDFS. Hadoop was set to run two map slots and two reduce slots per node in Cluster-I, while 6 map slots and 2 reduce slots in Cluster-II. However, degrees of parallelism are configured by the number of workers in incremental mode or that of partitions in batch mode. Other configurations are used default settings.

• Datasets We adopt three ubiquitous and representative types of unstructured data: 1) Document dataset *SogouP*, which is from SogouPic[10)], and the total number of documents is 130 million (2.13TB). 2) Image dataset *SogouT*, also from SogouPic, contains the million images(625GB). 3) Video dataset crawled form Youtube, is 1 665GB. We constructed partial data sets chosen randomly for storage and analysis experiments according to the experiment design, and use the whole data set to evaluate the search performance.

[10)] http://www.sogou.com/labs/resources.html

5.1 Storage performance

In this experiment, we evaluate the storage performance of D-Ocean by loading different kinds of data sets via the APIs from the clients' local files into D-Ocean. These datasets aim to cover major types of unstructured data, including images, videos, and a mixed data set consisting of the two types. We choose Hadoop's distributed file system as a baseline and compare it with our unified storage framework. Note that there are two ways to load data into Hadoop, the commandline file utility and the internal I/O API. We employ the java API for fairness. In both Hadoop and D-Ocean, each data object is stored with a replication factor of three.

We adopt two approaches for data loading according to the methodology in Ref. [30]. The first one fixes the size of data per node to be the same and only varies the number of nodes from 3, 6, 12 to 24. The second fixes the total dataset size to be the same and evenly divides the data amongst a variable number of nodes. These experiments not only measure how well each system scales as the number of available nodes is increased, but allow us to compare the hybrid multistore system to the single big data store.

The results for loading images and videos are shown in Figs. 10(a) and 10(b), while the size of data per node is fixed to be 500MB and 10GB, respectively. It can be observed that for storage of small files like images, D-Ocean outperforms Hadoop by a factor of 3.2x to 6.7x since the unified storage framework employs suitable data stores. However, as shown in Fig. 10(b), D-Ocean takes more time to load large video files than Hadoop. The overhead is generated by several additional operations in D-Ocean, e.g., UObject construction, storage mapping, and blob location recording. Nevertheless, the overhead is about 13%, which is acceptable. What's more, we load different kinds of data simultaneously in each node since workloads are always mixed. The results are depicted in Fig. 10(c). It shows that D-Ocean also outperforms Hadoop up to 1.9x, which attributes to the appropriate storage decisions of the hybrid storage system.

We also evaluate the scalability of two systems by fixing the total dataset size at 252GB and dividing the data evenly amongst nodes. Figure 10(d) shows the results on the mixed data set. It can be seen that as the number of nodes increases, the loading times of D-Ocean and Hadoop both decrease, which demonstrates the scalability of D-Ocean.

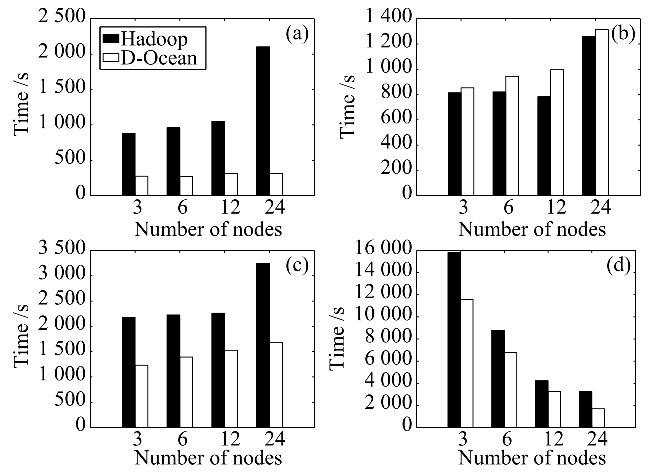


Fig. 10 Load times: (a) Image set (500MB/node); (b) video set (10GB/node); (c) mixed set (500MB images + 10GB videos, per node); and (d) mixed set (252GB/cluster)

5.2 Analysis performance

In this experiment, we investigate the analysis performance of D-Ocean. As mentioned before, D-Ocean provides two kinds of data processing mechanisms: incremental mode and batch mode. We configure each node of cluster-II to be able to execute both incremental tasks and batch tasks. The tasks in our experiments are to analyze several typical types of unstructured data, including Name-Entity extraction on the Document dataset, CEDD feature extraction on the Image dataset and keyframe extraction on the Video dataset. We first measure the throughput of incremental mode in terms of different number of workers, and then compare the two modes under certain degrees of parallelism.

Figure 11 shows the throughput (number of processed UObjects per minute) of incremental tasks in terms of different number of workers from 1 to 28. The throughput of Name-Entity extraction tasks is highest, because documents are very small to obtain fast upload and the computation is simplest. In contrast, the computation of extracting keyframes on videos takes much time, resulting in the lowest throughput shown in Fig. 11(b). For three tasks, the throughput increases as more workers are utilized. However, the throughput increases slowly after a threshold, due to the bottleneck of single processing node. The results provide hints to configure the number of workers.

Table 1 shows the processing time of the two modes. Both incremental tasks and batch tasks employed the same *analyzer* plugin to extract cedd features on a new collection of images. We fixed the number of loading clients in two modes to be 10 and varied the number of workers from 5 to 20. The processing time of incremental tasks is measured as the time is elapsed since the first image was uploaded and the last one was finished. The processing time of batch tasks has two parts: the time of loading data into D-Ocean and the job execution time. Note that batch tasks can be executed repeatedly after the data is loaded. In our experiments, it took 26 minutes to load all images for batch mode. As shown in Table 1, the processing time of the two modes decreases as the number of workers increases. The incremental tasks take less time than batch tasks, which benefits from the overlap of insertion and computation. However, the processing time of incremental mode decreases slightly when the parallelism becomes higher. The reason is that the number of loading clients is fixed, which makes workers under-utilized when the number of workers exceeds a certain value.

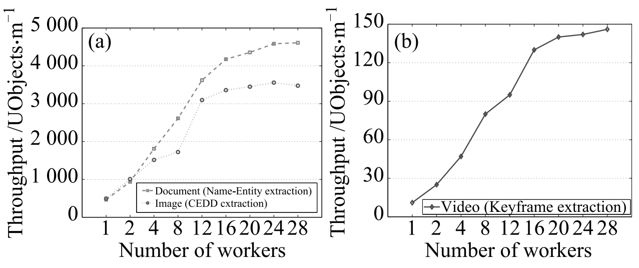


Fig. 11 Throughput of incremental processing tasks. (a) CEDD feature extraction and Name-Entity extraction (document and image analysis); (b) keyframe extraction (video analysis)

Table 1 Processing time of incremental and batch tasks /min

Parallelism (Number of workers)	Incremental	Batch
5	56.9	63.9
10	40.2	49.4
20	38.1	38.2

5.3 Search performance

In this experiment, we evaluate the search performance of D-Ocean by performing a series of queries on the Image data set. We model the data set as a UType *WebImage* with the following features: feature *image* is Blob type and stores web images; feature *description* is String type and stores surrounding texts of web images; feature *size* is Integer type and stores image size of web images; feature *people* is Link type and links to the feature person of another UType, which implies who appears in the surrounding text. We build SH index on feature *image*, inverted index on feature *description*. There is no index on the features *size* and *people*.

Table 2 Average response time of different types of queries

Search Tasks	Sample UQL	Average times/s
Image (with SH index)	Select from WebImage where image like @“path”	0.148
Keywords (with inverted index)	Select from WebImage where description contain “word”	0.056
Image+Keywords	Select from WebImage where image like @“path” and description contain “word”	0.273
Image+Properties	Select from WebImage where image like @“path” and size>100	0.377
Image+Keywords +Properties	Select from WebImage where image like @“path” and description contain “word” size>100	0.175
Link	Select from WebImage where people− >person = “name”	0.839
Image+Keywords +Link	Select from WebImage where image like @“path” and description contain “word” people− >person = “name”	0.232

After the ten million records are stored and indexed in D-Ocean, we test compound queries that consist of two or three individual simple sub-queries. We also test some simple queries for comparison. To demonstrate the effect of the optimization on compound query, we perform all the queries with simple queries under the same environment. Each query task is executed with ten different queries of the same type and disabled query cache. The average response time is shown in Table 2. We find that all the queries are accomplished within one second, which suggests that the efficiency of different types of queries is acceptable.

Figure 12(a) presents the average response time of a simple image query with SH index, a simple keyword query with Inverted index, and a compound query of them, under different values of query parameter top N. From Fig. 12(a), we find that the compound query of two simple sub-queries with indices uses more time than each simple query takes. This fact owes to the effect of merging strategy. The merging overhead of the compound query is very small when top N is smaller than 1 000. It means that merge strategy might be efficient when all the sub-queries of a compound query have efficient index and top N is small.

Figure 12(b) presents the average response time of a compound query that consists of one index-assisted sub-query and a property filter without any index, and another com-

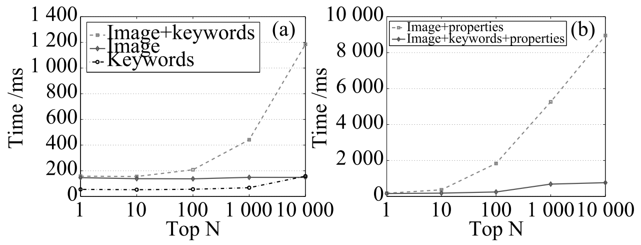


Fig. 12 Average response time of queries. (a) Merging strategy; (b) filtering strategy

pound query that consists of two index-assisted sub-queries and a property filter without any index, under different values of query parameter top N. From Fig. 12(b), we find that with more index-assisted sub-queries executed before property filter, the compound query takes less time than that with less index-assisted sub-queries. This is because property filter without index costs very much. This finding suggests that filter strategy might fit for compound queries that consist of at least one time-consuming property sub-query and some other index-assisted sub-queries. More index-assisted sub-queries a compound query has, the least time it costs, and this trend is more obvious while top N is large.

6 D-Ocean in production

Nowadays, many running systems in several domains, such as Digital Library, Digital Media, e-Government, have deployed D-Ocean to manage their unstructured data.

Taking china academic digital associate library (CADAL)[11] as an example, which is one of the largest digital libraries in China, it is facing several challenges now. It has digitized more than 2.5 million books and other multimedia resources such as Chinese calligraphy, painting, audio, video, etc. In addition, the traditional services such as browsing and reading services in the digital library are not satisfying. More smart services are needed to discover values in these large volumes of data, and more personalized services are needed to meet different people’s needs.

However, data in CADAL is in an unstructured form, and smart services depend on complex analysis and mining, which would be a big obstacle to develop CADAL further. Besides, the lack of an easy-to-use UI and interactive tools makes it difficult to develop different application services.

In order to address these challenges, D-Ocean has been deployed as an infrastructure in CADAL since 2013, where D-Ocean is layered under the Digital Library Engine [31] to support the entire digital library. So various types of data in-

[11] http://www.cadal.zju.edu.cn

cluding small files like images, paintings, calligraphy characters and book pages, large files like video, semi-structured data like metadata and catalog of books, entities extracted from books and the relationships between entities are all modeled with D-Ocean Data Model and stored in D-Ocean. Several smart services were developed as OSGI bundles, where the actual analysis and mining were executed in the generalized data processing framework. Take Chinese calligraphy service, Chinese traditional medicine service and Chinese literature chronicle service as examples.

- In Chinese calligraphy service, calligraphy works, characters, radicals and strokes are extracted sequentially, and then low-level features such as shape feature, stroke feature and high-level features such as writing style feature are extracted and mined for each calligraphic character. Then, low-level features and high-level features are indexed in D-Ocean. After that, several services such as calligraphy retrieval, calligraphic character recognition are developed.
- In Chinese traditional medicine service, entities such as herb, prescription and symptom, and illustrations about herbs are extracted from books. All extraction methods are implemented as plugins in D-Ocean. Then, prescription comparative analysis, compatibility analysis for herbs by mining the frequent set and property-similar herbs by clustering on the complex graph of herbs are provided through the plugins in D-Ocean.
- In Chinese literature chronicle service, entities such as people, works, office, location, etc., are extracted and annotated in the historical literature documents. So the literature document is organized with four dimensionalities of time, location, people and events.

Figure 13 shows the data model for calligraphy word, content, author and dynasty, where some links exist between entities for entity relationships modeling. A character Analyzer is implemented for feature extraction. Through the GUI Admin tool of D-Ocean, a task is defined to analyze characters in *picture* field and store the extracted feature to *histogram* field, as shown in Fig. 14. The extraction task could be executed in a batch mode or triggered by inserting new characters.

Once the feature is extracted, it could be indexed by adding index definition, which can be executed in the same way as the analysis task. After that, a UQL "*select from calligraphy:calligraphyword where picture like @'d:\TEMP\1.jpg' on histogram*" is used to search similar characters with the query image on histogram feature, as shown in Fig. 15.

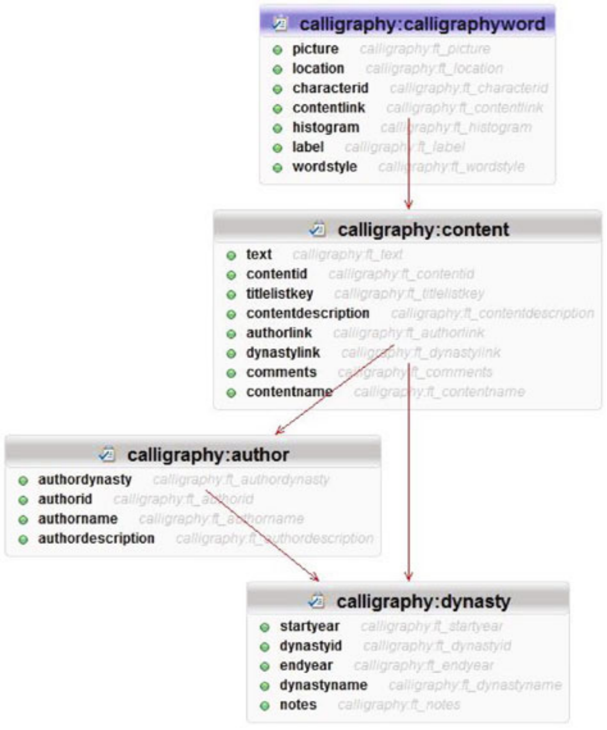
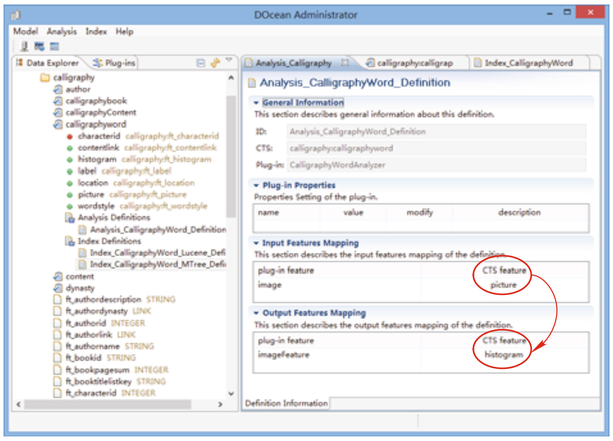

calligraphy:calligraphyword
picture calligraphy:ft_picture
location calligraphy:ft_location
characterid calligraphy:ft_characterid
contentlink calligraphy:ft_contentlink
histogram calligraphy:ft_histogram
label calligraphy:ft_label
wordstyle calligraphy:ft_wordstyle
calligraphy:content
text calligraphy:ft_text
contentid calligraphy:ft_contentid
titlelistkey calligraphy:ft_titlelistkey
contentdescription calligraphy:ft_contentdescription
authorlink calligraphy:ft_authorlink
dynastylink calligraphy:ft_dynastylink
comments calligraphy:ft_comments
contentname calligraphy:ft_contentname
calligraphy:author
authordynasty calligraphy:ft_authordynasty
authorid calligraphy:ft_authorid
authorname calligraphy:ft_authorname
authordescription calligraphy:ft_authordescription
calligraphy:dynasty
startyear calligraphy:ft_startyear
dynastyid calligraphy:ft_dynastyid
endyear calligraphy:ft_endyear
dynastyname calligraphy:ft_dynastyname
notes calligraphy:ft_notes

Fig. 13 Data model for calligraphy service



calligraphy
author
calligraphybook
calligraphyContent
calligraphyword
characterid calligraphy:ft_characterid
contentlink calligraphy:ft_contentlink
histogram calligraphy:ft_histogram
label calligraphy:ft_label
location calligraphy:ft_location
picture calligraphy:ft_picture
wordstyle calligraphy:ft_wordstyle
Analysis Definitions
Analysis_CalligraphyWord_Definition
Index Definitions
Index_CalligraphyWord_Lucene_Defi
Index_CalligraphyWord_MTree_Defi
content
dynasty
ft_authordescription STRING
ft_authordynasty LINK
ft_authorid INTEGER
ft_authorlink LINK
ft_authorname STRING
ft_bookid STRING
ft_bookpagesum INTEGER
ft_booktitlelistkey STRING
ft_characterid INTEGER

Analysis_CalligraphyWord_Definition

General Information
This section describes general information about this definition.
ID: Analysis_CalligraphyWord_Definition
CTS: calligraphy:calligraphyword
Plug-in: CalligraphyWordAnalyzer

Plug-in Properties
Properties Setting of the plug-in.

name	value	modify	description

Input Features Mapping
This section describes the input features mapping of the definition.

plug-in feature	CTS feature
image	picture

Output Features Mapping
This section describes the output features mapping of the definition.

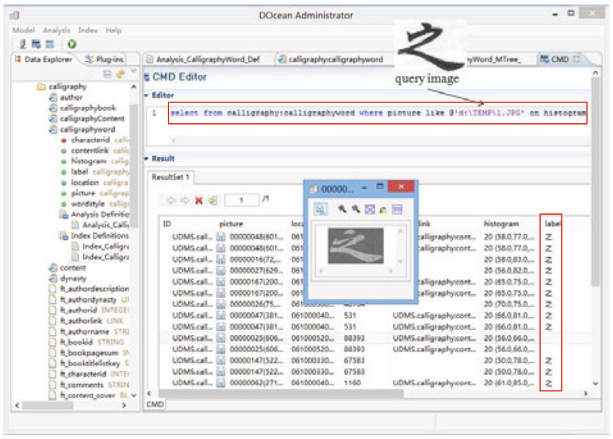
plug-in feature	CTS feature
imageFeature	histogram

Fig. 14 Data analysis definition for calligraphic character

With the help of Link Type, the following UQLs are used to search more complex results. For examples,

- search someone's calligraphy works: *select from calligraphy:content where authorlink->authorname="颜真卿"*;
- search all calligraphy characters of someone and they are also similar with the uploaded image: *select from calligraphy:calligraphyword where contentlink->authorlink->authorname="颜真卿" and picture like @'d:\TEMP\1.jpg'*.

Besides, lots of users' behavior logs are generated in



```
1   select from calligraphy:calligraphyword where picture like $'d:\TEMP\1.JPG' on histogram
```

Fig. 15   Data search for calligraphic character

CADAL every day, which are used for recommendation service by exploiting incremental processing and batch processing in D-Ocean. Specifically, logs are gathered and then preprocessed on incremental processing framework. Finally, all logs are mined through collaborative filtering recommendation algorithms on batch processing framework.

The D-Ocean in digital library application shows: 1) Built-in data model and plugins make data management and service development more convenient. 2) The unified storage and processing framework ensures the high performance of CADAL digital library. 3) The RAISE model, which provides storage, analysis, index, search and environment modeling, can simplify application development.

7 Conclusion

This paper presents D-Ocean, an unstructured data management system for data ocean environment. D-Ocean has an open and scalable architecture to support extensible data types, storage infrastructures, and process elements. It integrates batch and incremental processing modes to process unstructured data, and provides a combined search engine to conduct compound queries. It provides the RAISE process modeling method to define the whole process of Repository, Analysis, Index, Search, and Environment, which can greatly simplify application development. The experimental results demonstrate that D-Ocean can perform unstructured data storage, analysis, and search efficiently. Use cases in a digital library application show the ease of use of D-Ocean. The design and implementation of D-Ocean give insights to an easy way to construct Data Ocean which will boost big data applications and smart services.

Acknowledgements   The work in this paper was supported by the National Science and Technology Major Project of China (2010ZX01042-002-003-001).

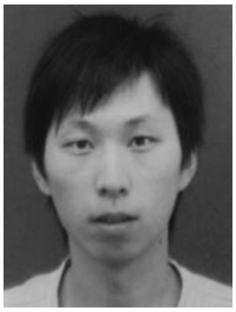
References

1. Cui B, Mei H, Ooi BC. Big data: the driver for innovation in databases. National Science Review, 2014, 1(1): 27–30
2. Laney D. 3D data management: controlling data volume, velocity and variety. META Group Research Note, 2001, 6: 70
3. David F, Adam L. UIMA: an architectural approach to unstructured information processing in the corporate research environment. Natural Language Engineering, 2004, 10(3–4): 327–348
4. Pan Y. Important developments for the digital library: data ocean and smart library. Journal of Zhejiang University-Science C, 2010, 11(11): 835–836
5. Martinez J M, Pereira F. MPEG-7: the generic multimedia content description standard, part 1. MultiMedia, IEEE, 2002, 9(2): 78–87
6. Doller M, Tous R, Gruhne M, Yoon K J, Sano M, Burnett I S. The MPEG query format: unifying access to multimedia retrieval systems. MultiMedia, IEEE, 2008, 15(4): 82–95
7. Melton J, Eisenberg A. SQL multimedia and application packages (SQL/MM). ACM Sigmod Record, 2001, 30(4): 97–102
8. Buneman P, Davidson S, Hillebrand G, Suciu D. A query language and optimization techniques for unstructured data. ACM SIGMOD Record, 1996, 25(2): 505–516
9. Halevy A, Franklin M, Maier D. Principles of dataspace systems. In: Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 2006, 1–9
10. Dittrich J P, Salles M V. iDM: a unified and versatile data model for personal dataspace management. In: Proceedings of the 32nd International Conference on Very Large Data Bases. 2006, 367–378
11. Stonebraker M, Weisberg A. The voltdb main memory dbms. IEEE Data Engineering Bulletin, 2013, 36(2): 21–27
12. LeFevre J, Sankaranarayanan J, Hacigumus H, Tatemura J, Polyzotis N, Carey M J. MISO: souping up big data query processing with a multistore system. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. 2014, 1591–1602
13. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM, 2008, 51(1): 107–113
14. Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCayley M, Franklin M J, Shenker S, Stoica I. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. 2012, 2
15. Oscar B, Sam R, Ian O C, Jimmy L. Summingbird: a framework for integrating batch and online MapReduce computations. Proceedings of the VLDB Endowment, 2014, 7(13): 1441–1451
16. Jiang D, Chen G, Ooi B C, Tan K L, Wu S. epiC: an extensible and scalable system for processing big data. Proceedings of the VLDB Endowment, 2014, 7(7): 541–552
17. Lewis D D, Jones K S. Natural language processing for information retrieval. Communications of the ACM, 1996, 39(1): 92–101
18. Lew M S, Sebe N, Djeraba C, Jain R. Content-based multimedia information retrieval: state of the art and challenges. ACM Transactions

on Multimedia Computing, Communications, and Applications, 2006, 2(1): 1–19

19. Wu E, Diao Y, Rizvi S. High-performance complex event processing over streams. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data. 2006, 407–418
20. Lux M, Chatzichristofis S A. Lire: lucene image retrieval: an extensible java CBIR library. In: Proceedings of the 16th ACM International Conference on Multimedia. 2008, 1085–1088
21. Brenna L, Demers A, Gehrke J, Hong M, Ossher J, Panda B, Riedewald M, Thatte M, White W. Cayuga: a high-performance event processing engine. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data. 2007, 1100–1102
22. Alsubaiee S, Altowim Y, Altwaijry H, Behm A, Borkar V, Bu Y, Carey M Cetindil I, Cheelangi M, Faraaz K. AsterixDB: a scalable, open source BDMS. Proceedings of the VLDB Endowment, 2014, 7(14): 1905–1916
23. Wang Y, Lu W, Wei B. Transactional multi-row access guarantee in the key-value store. In: Proceedings of the International Conference on Cluster Computing. 2012, 572–575
24. Yu Q. FastDFS: framework analysis and configuration optimization. In: Proceedings of Database Technology Conference China. 2012
25. Meng X, Wang X, Xie M, Zhang X, Zhou J. OrientX: an integrated, schema based native XML database system. Wuhan University Journal of Natural Sciences, 2006, 11(5): 1192–1196
26. Isard M, Budiu M, Yu Y, Birrell A, Fetterly D. Dryad: distributed data-parallel programs from sequential building blocks. ACM SIGOPS Operating Systems Review, 2007, 41(3): 59–72
27. Jarke M, Koch J. Query optimization in database systems. ACM Computing Surveys, 1984, 16(2): 111–152
28. Fagin R, Lotem A, Naor M. Optimal aggregation algorithms for middleware. Journal of Computer and System Sciences, 2003, 66(4): 614–656
29. Zhuang Y, Liu Y, Wu F, Zhang Y, Shao J. Hypergraph spectral hashing for similarity search of social image. In: Proceedings of the 19th ACM International Conference on Multimedia. 2011, 1457–1460
30. Pavlo A, Paulson E, Rasin A, Abadi D, Dewitt D J, Madden S, Stonebraker M. A comparison of approaches to large-scale data analysis. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data. 2009, 165–178
31. Lu W, Zheng L, Shao J, Wei B, Zhuang Y. Digital library engine: adapting digital library for cloud computing. In: Proceedings of the 6th IEEE International Conference on Cloud Computing. 2013, 934–941

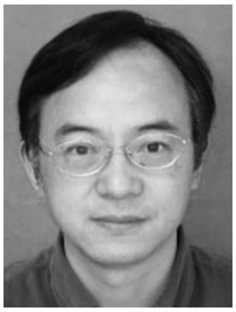
Yueting Zhuang received his BS, MS and PhD in computer science from Zhejiang University (ZJU), China in 1986, 1989 and 1998, respectively. From 1997 to 1998, he was a visiting scholar at Prof. Thomas Huang's group, University of Illinois at Urbana-Champaign, USA. Currently, he is a professor at the College of Computer Science, ZJU. His research interests mainly include artificial intelligence, multimedia retrieval, computer animation, digital library and databases.


Yaoguang Wang received his BS from South China University of Technology, China in 2010. He is currently a PhD student in Zhejiang University, China. His research interests include massive data storage management, parallel data processing, and distributed system.

Jian Shao received his BS in Department of Electronic Science and Engineering from Nanjing University, China in 2003, and his PhD in Institute of Acoustics, Chinese Academy of Science, China in 2008. Currently, he is an associate professor at the College of Computer Science, Zhejiang University, China. His research interests include cross media retrieval, artificial intelligence, and unstructured data management.

Ling Chen received his BS and PhD in computer science from Zhejiang University (ZJU), China in 1999 and 2004, respectively. Currently, he is an associate professor in the College of Computer Science, ZJU. His research interests include ubiquitous computing, HCI, AI, pattern recognition, distributed systems, databases, and data mining.

Weiming Lu received his PhD from Zhejiang University (ZJU), China in 2009. He is currently a lecturer in ZJU. His research interests are multimedia analysis and retrieval, artificial intelligence, digital library and unstructured data management.



Jianling Sun received his PhD in computer science from Zhejiang University (ZJU), China in 1993. Currently, he is a professor in the college of computer science of ZJU. His research interests include databases, data mining, distributed systems, and financial Information Technology.

Baogang Wei received his PhD from Northwestern Polytechnical University, China in 1997. He is currently a professor at Zhejiang University, China. His main research interests include artificial intelligence, pattern recognition, digital library, and information and knowledge management.

Jiangqin Wu received her PhD from Harbin Institute of Technology, China. Currently, she is an associate professor in Zhejiang University, China. Her research interests include multimedia computing, pattern recognition, and digital library.