International Journal of Software and Informatics, ISSN 1673-7288
http://www.ijsi.org, ijsi@iscas.ac.cn, +86-10-62661048
IJSI, 2021, 11(1): 69−90, doi: 10.21655/ijsi.1673-7288.00241
©2021 by Institute of Software, Chinese Academy of Sciences. All rights reserved.

Research Article

PandaDB: Intelligent Management System for Heterogeneous Data

Zhihong Shen (沈志宏) [1], Zihao Zhao (赵子豪) [1], Huajin Wang (王华进) [1], Zhongxin Liu (刘忠新) [1], Chuan Hu (胡川) [1], Yuanchun Zhou (周园春) [1]

[1] (Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China)
[2] (University of Chinese Academy of Sciences, Beijing 100049, China)
Corresponding author: Zhihong Shen, bluejoe@cnic.cn

Abstract With the development of big data application, the demand of large-scale structured/unstructured data fusion management and analysis is becoming increasingly prominent. However, the differences in management, process, retrieval of structured/unstructured data bring challenges for fusion management and analysis. This study proposes an extended property graph model for heterogeneous data fusion management and semantic computing, and defines related property operators and query syntax. Based on the intelligent property graph model, this study implements PandaDB, an intelligent fusion management system for heterogeneous data. This study depicts the architecture, storage mechanism, query mechanism, property co-storage, AI algorithm scheduling, and distributed architecture of PandaDB. Test experiments and cases show that the co-storage mechanism and distributed architecture of PandaDB have good performance acceleration effects, and can be applied in some scenarios of fusion data intelligent management such as entity disambiguation of academic knowledge graph.

Keywords data management system; heterogeneous data fusion; graph data model; ad-hoc query; AI

Citation Shen ZH, Zhao ZH, Wang HJ, Liu ZX, Hu C, Zhou CY. PandaDB: Intelligent management system for heterogeneous data. *International Journal of Software and Informatics*, 2021, 11(1): 69−90. http://www.ijsi.org/1673-7288/00241.htm.

Nowadays, the number and variety of data are increasing rapidly with the arrival of the era of big data. Education data, which includes media coverage of educational events and related policies enacted by government at all levels, has aroused wide attention of decision makers and researchers. In the field of education, a disruptive change in information technology is quietly happening. Meanwhile, the public opinions of internet users on educational policy and educational events are

This is the English version of the article “PandaDB: 一种异构数据智能融合管理系统. 软件学报, 2021, 32(3): 763−780. doi: 10.13328/j.cnki.jos.006180”.

Funding items: Strategic Priority Research Program of Chinese Academy of Sciences (XDB38030300); National Natural Science Foundation of China (61836013); Special Project of Ministry of Science and Technology for Innovation Methods (2019IM020100); Informatization Plan of Chinese Academy of Sciences (XXH13503)
Received 2020-07-20; Revised 2020-09-03; Accepted 2020-11-06; IJSI published online 2021-03-31

reflected in the network via a variety of forms, such as micro-blog, forums, news reports, etc. It is very meaningful to integrate, process and analyze the multi-source educational data utilizing big data and visualization techniques. It is difficult for users to analyze so various and massive data effectively. An effective method is to use the knowledge graph technique that is widely used at present. Knowledge graph[1], whose essence is a semantic network to reveal the relationship between knowledge, is a new way to represent knowledge. It can express information effectively and infer new knowledge based on what we have accordingly. Knowledge graphs based on web information such as DBpedia[2], YAGO[3], ReVerb[4], are well studied, most of which are constructed by entities and their relationships that are extracted from web, such as Wikipedia, Baidubaike, etc. Besides, another type of knowledge graph focuses on specific domains, such an academic citation relationship and life sciences. They can clearly demonstrate knowledge and directly solve problems in this domain. It is an important research topic how to clearly visualize entities and relationships between them. A commonly used way of displaying knowledge graph is to describe the entity as a node and the relationship between two entities as a line, which can display characteristics of knowledge graph structure. However, the layout of the relationship between entities will produce clutter using traditional graph layout directly to show knowledge graph. In the era of big data, with the promotion and use of various applications, data generation is getting faster, resulting in a larger data volume. On the one hand, the rapid development of data collection technology has made the data more diverse in structure and richer in types. Data is characterized by diversity and heterogeneity, of which unstructured data occupies a large proportion. Studies have shown that unstructured data such as videos, audios, and images accounts for up to 90%[1]. On the other hand, data management and analysis technologies such as data central platform and domain knowledge graph have been widely used in recent years. Data central platform requires structured/unstructured data to be well governed in a unified environment to support multiple applications. The domain knowledge graph, especially multi-modal knowledge graph[2], requires to perform fusion and correlation analysis on the underlying structured/unstructured data and support interactive queries. These technologies all put forward the need for fusion management and analysis of structured/unstructured data.

Structured data usually has a standardized and unified form. At present, for the management and analysis of structured data, there are mature data models, query languages and management systems. Compared with structured data, unstructured data has many differences in the management method, which brings many challenges to efficient fusion management and analysis of structured/unstructured data.

(1) The separated storage management challenges the unified management of structured/unstructured data. Compared with structured data, unstructured data occupies more space. Considering reading and writing efficiency, unstructured data is often stored alone in the file system or object storage system, which makes it more difficult to maintain the consistency of structured/unstructured data.

(2) Different ways of information acquisition pose challenges to the unified analysis of structured/unstructured data. Compared with structured data, unstructured data has complicated content. To achieve efficient retrieval and analysis, we should introduce pattern recognition, deep learning and other methods in advance to achieve information extraction and data mining, so as to obtain the inherent information contained in unstructured data.

(3) Inconsistent retrieval challenges the consistent ad-hoc query of structured/unstructured data. Unlike structured data with relatively mature SQL and SQL-like query languages, the information retrieval of unstructured data often lacks a unified operation mode and query syntax, and the current personalized solutions are mostly available on a case-by-case basis.

To achieve the fusion management and analysis of structured/unstructured data, we need to design a unified representation and query method from the model level. Traditional relational models and property graph models cannot effectively reveal and represent the inherent information of unstructured data. Some scholars have proposed to represent data and schema as edge-labeled graphs to replace the lack of underlying type constraints of unstructured data[3]. However, this method only adds schema to unstructured data and cannot achieve free retrieval of information in unstructured data. Li et al. proposed to define unstructured data from four perspectives: primitive properties, semantic features, underlying features, and original data[4]. However, this method relies on pre-definition and is not suitable for interactive queries of unstructured data. In recent years, some scholars have proposed a method for extracting RDF triples on unstructured data streams [5]. Despite the extraction of triples, this method cannot support interactive query of the internal information of unstructured data and does not have the basic capabilities of data management systems.

Another method of fusion management is to store unstructured data as a Bnary Large OBject (BLOB) in the database. When an application acquires the data, it returns a binary array or data stream, which is not satisfactory in terms of performance and functionality[6]. To solve this problem, researchers have proposed a series of unstructured data management systems[7–9]. Considering the large volume and complex structure of unstructured data, these systems design a suitable storage model to solve the storage and management problems of unstructured data to a certain extent. However, the query service is only based on the file object itself and metadata, and cannot provide the ability to query the internal information of unstructured data.

Therefore, this paper proposes an extended property graph model and its query method. Based on the traditional property graph, the extended property graph model improves the ability to represent the inherent information of unstructured data and the interoperability between structured and unstructured data. On this basis, this paper then proposes PandaDB, an intelligent fusion management system for heterogeneous data based on the intelligent property graph model.

In this paper, Section 1 shows an extended property graph model and related concepts, including cascading property graph, intelligent property graph, and sub-properties, and proposes property operators and query syntax. Section 2 presents the system design and specific implementation of PandaDB. Section 3 verifies the efficiency and feasibility of the system through experiments and cases. Section 4 introduces the work related to the research of this paper. Finally, the possible challenges in future research are prospected.

1 Conceptual Design

Traditional property graph models cannot effectively represent unstructured properties, so in this section, an extended property graph model is proposed to solve the problem of effective representation of unstructured properties. Then, the semantic operation and query syntax design for the extended property graph model is introduced to support the new query features brought by the introduction of unstructured properties and their internal information.

1.1 Extended property graph model

The traditional property graph model can be formally represented as $G=(V, E, P)$. Here, G is the entire data; V is the entity collection in the data; E is the relationship collection between the entities, and P is the property collection of the entities in the data set.

For unstructured properties such as images, voices and texts, the property graph model cannot effectively reveal their inherent information (for example, the property “photo” of a certain vertex contains “license plate number” information). The internal information is usually offline and has a structural delay.

- Offline: The information extraction of transforming unstructured properties into structured and semi-structured properties belongs to the preprocessing of data.
- Structural delay: The inherent information of unstructured properties does not have a clearly defined structure, so a specific information extraction method is selected according to the later needs of the application. This structure is unclear and delayed in definition.

To enhance the unified representation ability of structured properties and unstructured properties, this paper extends the property graph model, and proposes a cascading property graph model and an intelligent property graph model.

Definition 1. A property graph with the following features is called a cascading property graph.

1) The cascading property graph G^c can be expressed as $G^c=(V, E, PP, PN)$, where PP is a set of primitive properties and PN is a set of nested properties.

2) The values of the primitive properties are texts, numerical values, binary arrays and other basic data types.

3) The values of the nested properties are another property graph.

Definition 2. A property graph with the following features is named as an intelligent property graph.

1) The intelligent property graph G^I can be expressed as $G^I=(V,E,PP,O)$, where O is the set of semantic operations.

2) There is $PI\subseteq PP$, where PI is a collection of intelligent properties.

3) $O=(S_\varphi, S_\xi)$, where S_φ is the set of extension operations for intelligent properties, and S_ξ is the set of semantic computation operations for intelligent properties.

4) For the intelligent property $p_i\in PI$, there is an extension operation $\varphi\in S_\varphi$ to satisfy $p_n=\varphi(p_i)$ and $p_n\in PN$, namely that the intelligent property p_i is expanded into a nested property p_n.

5) For the intelligent properties $p_1\in PI$ and $p_2\in PI$, there is a semantic calculation operation $\xi\in S_\xi$, which can satisfy $\sigma=\xi(p_1, p_2)$. The two intelligent properties p_1 and p_2 are semantically calculated, and the result σ is obtained.

Definition 3. Nested properties have sub-properties.

1) For the intelligent property graph $G^I=(V, E, PP, O)$, if $V_i\in V$ exists and V_i has nested properties p_i^N, according to Definition 1, there is $p_i^N=G_i^N$, where G_i^N is the traditional property graph (V_i^N, E_i^N, P_i^P).

2) If $|V_i^N|=1$, then P_{ij}^S, any element of P_i^P, is called the sub-property of V_i.

As an example, Figure 1 shows an intelligent property graph. “car1” is a vertex of “Car” type in the graph, which has an intelligent property “photo”. After the extension operation, “photo” has two sub-properties: photo$\rightarrow$plateNumber and photo$\rightarrow$model.

1.2 Property operators

Relying on Definition 2, this paper defines sub-property extraction operators and semantic calculation operators for intelligent properties.

- Sub-property extraction operator: The sub-property extraction operator “$\rightarrow$” extracts the sub-properties of intelligent properties. For example, executing “photo$\rightarrow$plateNumber” for the property “photo” can get the license plate number in “photo”.
- Semantic calculation operator: The predicate in the traditional property graph query language only supports the comparison between properties, such as =, >, <, and regular

matching. This paper adds ~:, ::, :> and other extension predicates for intelligent properties to express the similarity relationship, similarity, inclusion relationship and other logics between unstructured properties (Table 1). The similarity relationship means whether two objects are similar to each other(the similar degree reach a threshold). The similarity means the similar degree, it measures how much two objects are similar to each other.

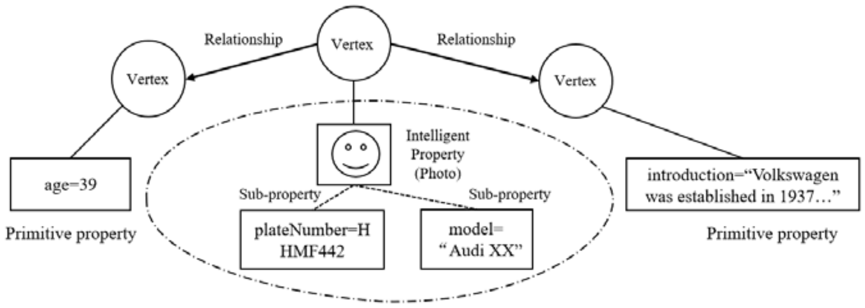


Figure1 Intelligent property graph model

Table 1 Semantic computation operators

Operation name	Symbol	Implication	Example
SemanticCompare	::	Calculate the similarity between *x* and *y*	*x*::*y*=0.7
SemanticLike	~:	Whether *x* and *y* are similar?	*x*~:*y*=true
SemanticUnlike	!:	Whether *x* and *y* are not similar?	*x*!:*y*=false
SemanticIn	<:	Whether *x* is included in *y*	*x*<:*y*=true
SemanticContain	>:	Whether *x* includes *y*	*y*>:*x*=true

In the property graph model, due to the invisibility of internal information, the calculation operation support between unstructured properties is limited. In combination with the description of unstructured properties in Definitions 1–3 and the features of property operators, the intelligent property graph model can online acquire the internal information of unstructured properties, where the structure is predefined.

- Online: The acquisition of structured and semi-structured information (sub-properties) from unstructured properties is on-demand, without special preprocessing of unstructured properties.
- Predefined structure: The inherent information of unstructured data depends on the definition of the schema level, rather than the implementation of information extraction tools. The underlying query mechanism supports direct operations on unstructured properties.

1.3 Query syntax

Aiming at the intelligent property graph model, this paper extends the standardized Cypher query language[10] to form the CypherPlus language. CypherPlus defines the property type 'BLOB' to represent the value of unstructured properties, and introduces new features such as BLOB literals, sub-property extraction operators, and semantic computation operators to support the representation and semantic operations of unstructured properties.

(1) BlobLiteral

BlobLiteral represents the literals of unstructured properties, with the format as <schema:// path>, where schema can be FILE, HTTP(S), FTP(S) and BASE64, as shown in Figure 2.

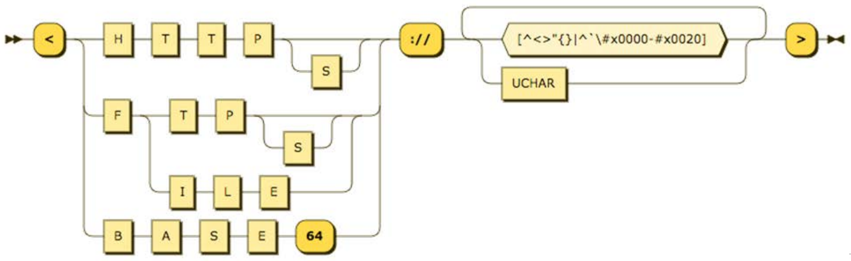


Figure 2 Syntax definition of BlobLiteral

(2) SubPropertyExtractor

SubPropertyExtractor refers to the extraction operation of the sub-properties. As shown in Figure 3, PropertyKeyName is the name of a sub-property.

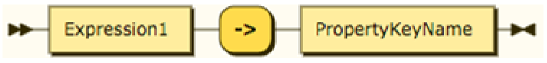


Figure 3 Syntax definition of SubPropertyExtractor

(3) SemanticComparison

Semantic computation operators include SemanticCompare, SemanticLike, SemanticUnlike, SemanticIn, SemanticContain and other operators. For example, SemanticLike indicates whether the values of two properties are similar, whose syntax definition is shown in Figure 4. AlgorithmName is the name of the algorithm for the specified calculation, Threshold is the threshold, AlgorithmName and Threshold are optional. In this case, the execution engine uses the default comparator and threshold.

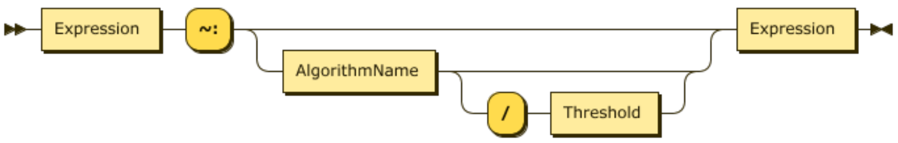


Figure 4 Syntax definition of SemanticLike

For example, for the data model in Figure 1, cars that are similar to the car with the license plate number HHMF442 can be found. The query statement is as follows:

- Q1: match (c1:CAR), (c2:CAR) where c1.photo~:c2.photo and c1→plateNumber= ‘HHMF442’ return c2;
- Q2: return ‘Zhihong SHEN’ ::jaro ‘SHEN Zhihong’.

The query statement Q2 is used to calculate the similarity of two texts.

2 System Implementation

For the fusion management and associated query analysis of structured and unstructured data, this paper uses the intelligent property graph model to design and implement the intelligent fusion management system PandaDB for heterogeneous data based on the Neo4j open source version. Section 2.1 introduces the overall architecture of PandaDB, and Sections 2.2–2.5 respectively introduce the design ideas and implementation details of each module.

2.1 Overall architecture

PandaDB organizes data with an intelligent property graph model. The underlying data is divided into three parts: graph structure data, structured property data, and unstructured property data. Among them, graph structure data refers to data describing the structure of the graph such as the nodes and edges of the graph. Structured property data refers to data such as numerical values, strings, and dates. Unstructured property data generally refers to data except structured data, such as videos, audios, photos and documents. PandaDB stores unstructured data in the form of BLOB objects and represents them as properties of entities (nodes). According to the application characteristics of the above three types of data, PandaDB has designed a distributed multi-storage plan.

- Distributed graph data storage: On the basis of the traditional graph database, the graph structure data and property data are stored and the same data copy is stored on each node.
- Structured property co-storage: With external storage such as ElasticSearch and Solr, we can achieve the storage and index construction of large-scale structured property data.
- BLOB storage: In storage systems such as Hbase and Ceph, the unstructured property data can be stored in a distributed manner.

The overall architecture of PandaDB is illustrated in Figure 5, and the important modules are described below.

- Storage engine: This module maintains local graph structure data, schedules external property storage and provides services for the query engine on demand.
- External storage: This module includes two parts: ElasticSearch-based structured property co-storage and HBase-based BLOB storage.
- Query engine: This module parses and executes CypherPlus queries.
- AIPM: This module is a service framework of AI models, which realizes the flexible deployment of AI models and efficient on-demand running through model and resource management. Besides, it effectively shields the dependence between AI models.

The PandaDB cluster adopts a master-less architecture. In Figure 5, PandaNode is one of the nodes with a query engine and a storage engine. Property data and unstructured data are stored in external distributed storage tools. PandaNode only saves graph structure data and interacts with external storage through the property storage interface.

2.2 Storage mechanism

PandaDB introduces BLOB into the type system of Neo4j and modifies the storage structure of Neo4j at the same time. The storage structure is shown in Figure 6. In addition to Neo4j reserved area, the property fields of BLOB also record the metadata of BLOB, including the unique identifier blobid, length and MIME type.

To invoke the external BLOB storage system, PandaDB designs BlobValueManager interface, and defines getById(·)/store(·)/discard(·) and other operation methods. As an implementation of BlobValueManager, HBaseBlobValueManager accesses BLOB data with the HBase cluster. In this plan, HBase is designed as a wide table containing N columns so as to support large-scale BLOB storage. Blobid/N is taken as the row key of HBase and blobid%N corresponds to a certain column of HBase.

We encapsulate the content reading of BLOB as an InputStream to speed up the reading of BLOB. When users acquire BLOB content or perform semantic computation through the Bolt protocol, this streaming read mechanism improves the performance of the operation.

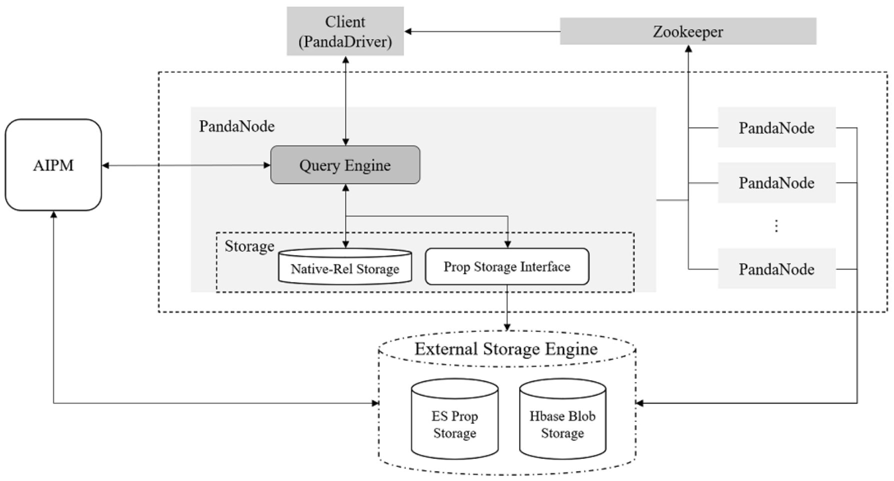


Figure 5 Architecture of PandaDB

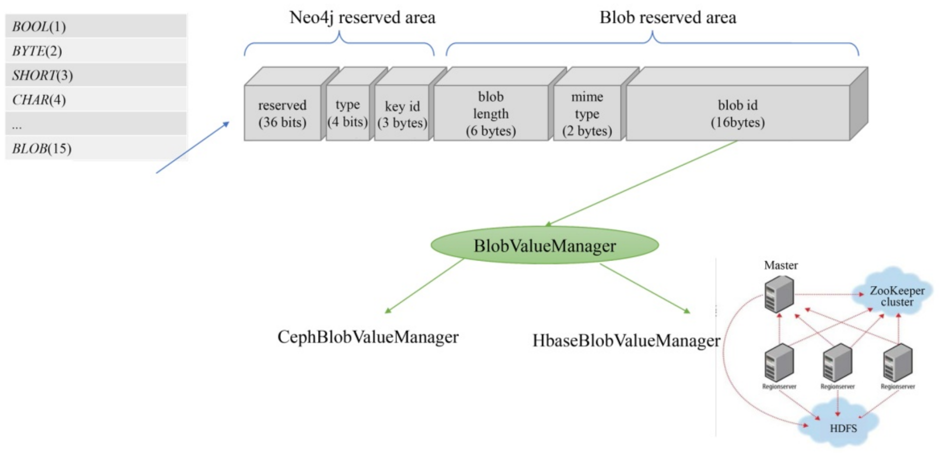


Figure 6 Design of BLOB storage structure

Multiple storage complicates storage transaction guarantees. When the client writes data, it sends a writing operation request to the leader node of PandaDB, which then performs the specific writing operation. The specific operation process of the leader node is as follows.

(1) The leader node starts the transaction, executes Cypher analysis and translates the transaction into specific execution operations.

(2) A request is sent to BLOB storage engine to perform the writing operation of BLOB data. If the execution fails, roll back the writing execution. and the transaction is marked as failed.

(3) If the BLOB data is written successfully, the writing operation of graph structure data and structured property data will be executed. If the execution fails, perform rolling back upwards and the transaction is marked as failed.

(4) The modification of structured property data is synchronized to the co-storage. If the execution fails, carry out rolling back upwards and the transaction is marked as failed.

(5) The transaction is submitted.

(6) The transaction is closed and then the return operation succeeds.

2.3 Query mechanism

PandaDB query engine mainly aims at the analysis of query statements, the generation and optimization of logical plans, and the optimization and execution of physical plans. On the basis of Neo4j, PandaDB query engine primarily improves the following parts.

(1) Parsing stage: This stage enhances the parsing rules of Cypher language, supports BLOB literal constant (BlobLiteral), BLOB sub-property extraction operator (SubPropertyExtractor), and property semantic operator (SemanticComparison).

(2) Grammar inspection stage: This stage performs formal inspections for BlobLiteral, SubPropertyExtractor and SemanticComparison, such as finding illegal BLOB paths, semantic operators and thresholds.

(3) Plan optimization stage: This stage optimizes the operation of BlobLiteral, and uses predicate pushdown and other strategies for large-scale property filtering situations.

(4) Plan execution stage: This stage fully schedules the property co-storage module, AIPM module and BLOB storage module to achieve efficient property priority filtering, BLOB acquisition and semantic computation. Figure 7 shows a typical query process that requires returning all nodes that are similar to the face in photo0 and have an age value greater than 30.

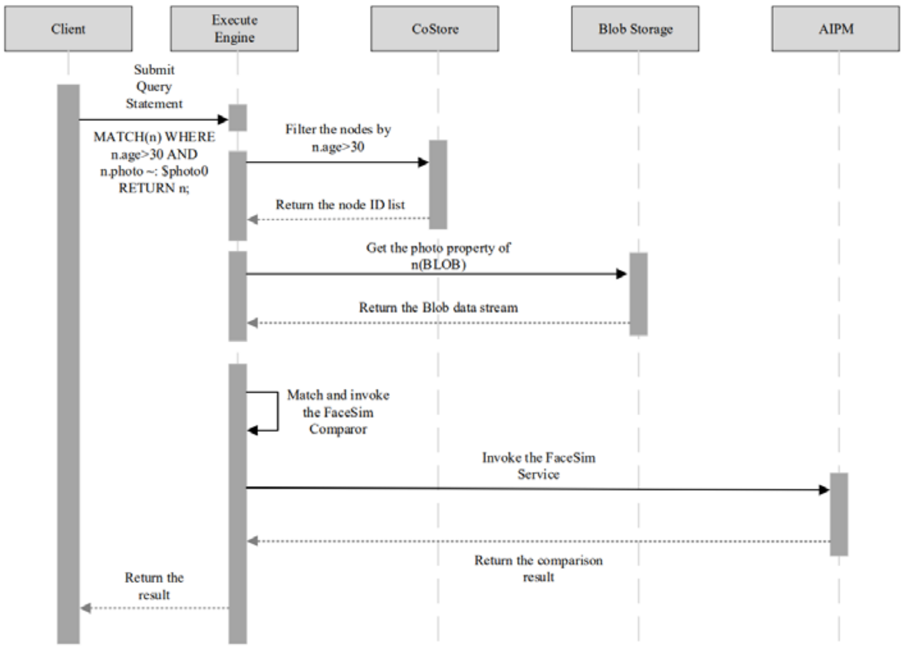

Client
Execute Engine
CoStore
Blob Storage
AIPM
Submit Query Statement
MATCH(n) WHERE n.age>30 AND n.photo ~: $photo0 RETURN n;
Filter the nodes by n.age>30
Return the node ID list
Get the photo property of n(BLOB)
Return the Blob data stream
Match and invoke the FaceSim Comparor
Invoke the FaceSim Service
Return the comparison result
Return the result

Figure 7 CypherPlus query process

Figure 8 shows the design of query mechanism of PandaDB from the three levels: Query syntax, query plan, and execution engine. The parsing engine converts the symbols in the query statement into semantic operators, and the execution engine selects the AI model to process the corresponding data according to the rule set and returns the result.

To speed up the query of unstructured data, PandaDB implements a semantic index function. The information in unstructured data is considered to be a kind of semantic information, such as the face in the image, the license plate number of the car in the image and the text information contained in the recording. The extraction of information from unstructured data by AI models can be regarded as the mapping of data from high-dimensional space to low-dimensional space, and the result of mapping in the low-dimensional space can be used as the semantic index of the data in this scenario.

For example, in the comparison and query of faces, it is necessary to compare the similarity of faces in different images. Face recognition models are usually used to extract facial features and compare the similarity of the two features. In PandaDB, the facial features represented in vector form are regarded as the semantic index of the unstructured data in the current query scenario. When the system processes a query involving face comparison, it first checks whether there is a corresponding semantic index. If the semantic index corresponding to the query exists, a processing request to AIPM is not sent and the semantic index is compared directly to get the result.

Semantic index can reduce the number of requests made by the query engine for AI services; therefore, repeated data transmissions are avoided and system efficiency is improved.

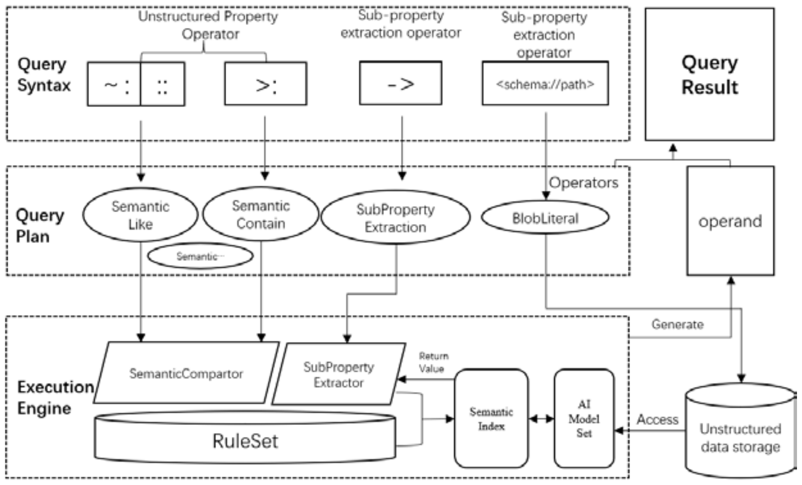


Figure 8 Query mechanism of intelligent property graph

2.4 Property data co-storage

PandaDB introduces a property data co-storage mechanism to implement full-text indexing of structured property data and improve the efficiency of filtering and querying node properties. Currently, PandaDB supports ElasticSearch as a co-storage engine.

Figure 9 shows the writing process of the PandaDB co-storage module. The key design of the co-storage mechanism is as follows.

(1) The storage structure of property data in ElasticSearch: Each Neo4j graph database corresponds to an independent index in the co-storage engine (ElasticSearch). The property data and label data of each node are organized into a document in ElasticSearch. Specifically, the ID, property name and property value of the nodes in the Neo4j database respectively serve as those of the document, and the label data of the nodes is represented as a specially set property label. Numerical values, strings, coordinates, dates, time and other structured property data types are converted to corresponding data types in ElasticSearch.

(2) Property writing and updating: To maintain the consistency of the local data in Neo4j database and the data in ElasticSearch, PandaDB extends the node update part in the transaction operation execution module (operations) in Neo4j and designs ExternalPropertyStore to store all operations performed in the Neo4j transaction. When Neo4j database performs operations such as

inserting nodes, adding tags, setting properties and deleting nodes, it also caches the corresponding operation data in the ExternalPropertyStore. When Neo4j database performs the transaction submission operation, the cached operation data is synchronized to ElasticSearch.

(3) Property filtering: To achieve node property filtering based on co-storage, PandaDB modifies Cypher query execution plan of Neo4j and pushes down the node property filtering predicates to the co-storage management module. According to the predicate filtering conditions, the ElasticSearch search request is generated. Finally, the list of hit documents (nodes) is returned to the query engine for further filtering. To avoid the network transmission delay due to a large number of query results, PandaDB uses the results delivered by asynchronous batching.

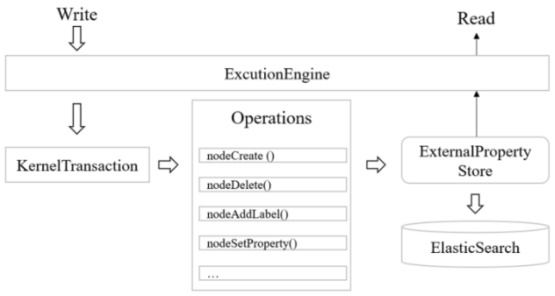


Figure 9 Writing process of property co-storage module in PandaDB

2.5 AI algorithm integration and scheduling

AI algorithm integration and scheduling mainly include local algorithm-driven management and service framework of AI algorithms.

(1) Local algorithm-driven management

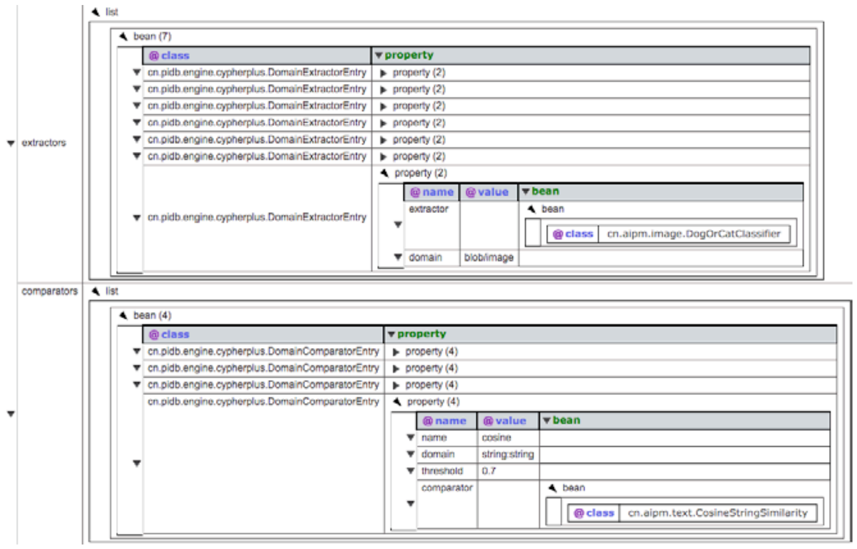
PandaDB develops a drive management rule base to uniformly manage different extractors (SubPropertyExtractor) and semantic comparators (SemanticCompartor). Figure 10 shows part of the content in the rule base, where DogOrCatClassifier extracts the type of pets and is suitable for the property of the blob/image type. CosineStringSimilarity calculates the cosine similarity of two text strings and is only suitable for the properties of two string types.

(2) Service framework of AI algorithms

AI services enable the information extraction in PandaDB. AIPM provides AI services for PandaDB, shields the dependency conflicts between different AI models, reduces the difficulty of deployment and maintenance of artificial intelligence models and facilitates PandaDB to extend AI operators on demand. Figure 11 shows the interaction logic between AIPM and the system. The system sends a query request to AIPM in the form of an HTTP request. The request path has a corresponding relationship with the AI algorithm. Then AIPM receives the query request, invokes the corresponding AI algorithm to process the data and returns the result in the form of a JSON string. To enhance the extensibility of AI operators, AIPM designs a unified integrated interface and requires operators to support these interfaces. Figure 12 shows the management framework of AI models.

3 System Effect Evaluation

To verify the effectiveness of the model design and implementation of PandaDB, this paper tests property co-storage, distributed scheme and unstructured data information query. At the same time, application cases verify the fusion management ability of PandaDB for structured and unstructured data.



```
extractors   list
               bean (7)
                 @class                                          property
                 cn.pidb.engine.cypherplus.DomainExtractorEntry  property (2)
                 cn.pidb.engine.cypherplus.DomainExtractorEntry  property (2)
                 cn.pidb.engine.cypherplus.DomainExtractorEntry  property (2)
                 cn.pidb.engine.cypherplus.DomainExtractorEntry  property (2)
                 cn.pidb.engine.cypherplus.DomainExtractorEntry  property (2)
                 cn.pidb.engine.cypherplus.DomainExtractorEntry  property (2)
                 cn.pidb.engine.cypherplus.DomainExtractorEntry  property (2)
                                                                   @name      @value      bean
                                                                   extractor              bean
                                                                                            @class  cn.aipm.image.DogOrCatClassifier
                                                                   domain     blob/image
comparators  list
               bean (4)
                 @class                                           property
                 cn.pidb.engine.cypherplus.DomainComparatorEntry  property (4)
                 cn.pidb.engine.cypherplus.DomainComparatorEntry  property (4)
                 cn.pidb.engine.cypherplus.DomainComparatorEntry  property (4)
                 cn.pidb.engine.cypherplus.DomainComparatorEntry  property (4)
                                                                    @name       @value         bean
                                                                    name        cosine
                                                                    domain      string:string
                                                                    threshold   0.7
                                                                    comparator                 bean
                                                                                                 @class  cn.aipm.text.CosineStringSimilarity
```

Figure 10 Matching rule sets of extractors and semantic comparators

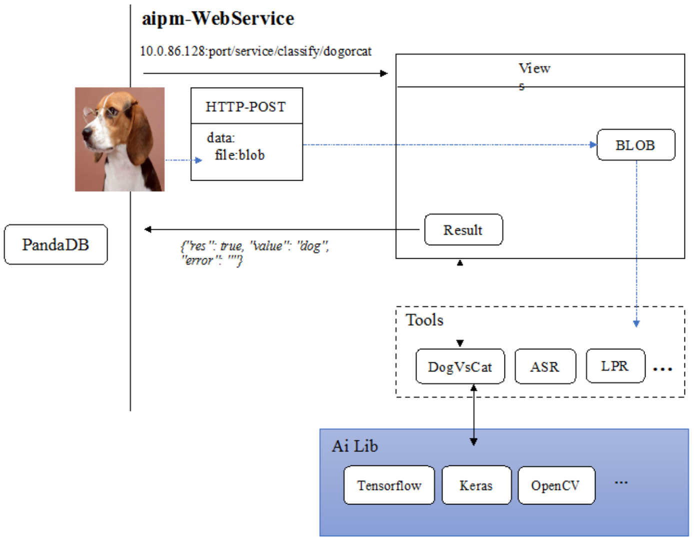


Figure 11 Interaction of PandaDB and AIPM

3.1 Test of property co-storage performance

This test verifies the performance based on the property co-storage scheme of ElasticSearch, and compares the query performance of Neo4j and PandaDB after the co-storage scheme is introduced. The test environment is shown in Table 2.

The Cypher query statement used in the experiment is shown in Table 3. In the experiment, each query statement is tested multiple times and the average execution time is taken (Table 4 and Figure 13). To avoid the performance impact caused by the cold start, we warm up the system before the test.

The test results demonstrate that due to the use of ElasticSearch as the co-storage and index of node properties, PandaDB has a clear advantage in the test of the above query statements. Especially in multi-property filtering query and fuzzy matching query of nodes, the performance is improved by 2 to 6 times on average.

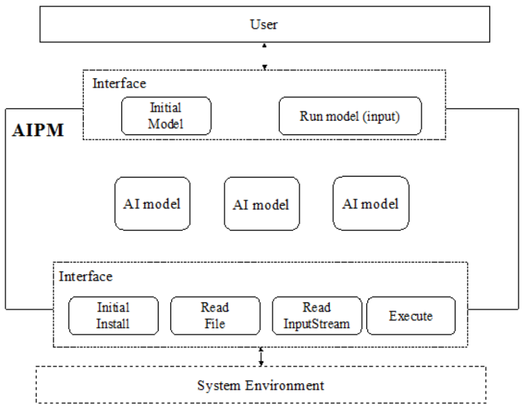


Figure 12 Management framework of AI models

Table 2 Information about test environment

Test environment	Description
Server software and hardware environment	Five physical servers of the same configuration: CPU: 32 Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz RAM: 128 GB HD: 6TB 7200 RPM SAS HDD Network: 1000Mbps interconnection OS: CentOS Linux release 7.7.1908 (Core)
Testing software version	Neo4j: 3.5.6 (Community Edition) PandaDB: v0.0.2(Developer Edition) ElasticSearch: 6.5.0
Environment deployment	ElasticSearch is deployed on all the five servers. PandaDB is deployed on four servers. Neo4j is deployed on one server. (Note: This Neo4j version only supports single-node deployment)
Test data	Graph data set: 100 million nodes (including 6 types of labels), 1.7 billion relationships (including 10 types) (Note: This data set is a real data set based on scientific research talents, institutions and papers, awards, patents, standards, and monographs.)

3.2 Distributed performance test

To verify the improvement of query response capabilities brought about by the distributed architecture, this paper deploys PandaDB and Neo4j stand-alone versions on a physical machine cluster (Table 2 for physical machine configuration) to evaluate the throughput rate of concurrent query requests. Because the Neo4j community edition only supports the stand-alone mode, PandaDB is deployed on three physical machines and Neo4j on one physical machine.

In this paper, common queries in graph computing analysis are selected as test cases. For example, to calculate the out-degree and in-degree of nodes, we set the timeout threshold to 300 s. On the premise that 90% of the queries do not time out, the throughputs of Neo4j and PandaDB are respectively 15 and 40 queries per second, which means the throughput of PandaDB is close to 3 times that of Neo4j.

Table 3 Query statement in the property co-storage test

No.	Query statement	Description of test type
Q-1	match (*n*:*paper*) where *n.country*="Malta" return *count*(*n*);	Node query (single- property precise filtering)
Q-2	match (*n*:*person*) where *n.org* STARTS WITH "Shanghai" return *count*(*n*);	Node query(single-property fuzzy atching)
Q-3	match (*n*:*paper*) where *n.country*="United States" and *n.citation*=10 return *count*(*n*);	Node query (two-property precise filtering)
Q-4	match (*n*:*person*) where *n.org* STARTS WITH "Shanghai" and *n.citations*=1 return *count*(*n*);	Node query (two-property filtering combining fuzzy matching and precise matching)
Q-5	match (*n*:*person*) where *n.citations*=100 and *n.citations*5=200 return *count*(*n*);	Node query (two-property precise filtering)
Q-6	match (*n*:*person*) where *n.citations*=192 and *n.citations*5=204 and *n.nationality*="France" return *count*(*n*);	Node query (three-property precise iltering)
Q-7	match (*n*:*person*) where *n.citations*=192 and *n.citations*5=204 and *n.nationality*="France" and *n.publications*5=5 return *count*(*n*);	Node query (four-property precise filtering)
Q-8	match (*startNode*:*paper*) [*r*]→(*other*) where *startNode.country*="Japan" and *startNode.citation*=5 return *count*(*other*);	Relation query(return end node information)
Q-9	match (*n*:*person*) [*r*:*work_for*]→(*other*) where *n.org* starts with "Beijing" and *n.citations*=1 return *count*(*r*)	Relation query(return relation information)
Q-10	match (*startNode*:*person* {personId:'32'}) optional match (*startNode*) [*r*:*write_person*] (*other*) return *count*(*other*)	Relation query(optional relationship matching)

Table 4 Test result for property co-storage

Test statement number	Average execution time (ms)		Ratio of execution time
	Neo4j	PandaDB	
Q-1	184	189	0.97
Q-2	96.5	97.5	0.98
Q-3	1 001.5	55	18.20
Q-4	426	42	10.14
Q-5	358	309	1.15
Q-6	304.5	51.5	5.91
Q-7	94	52.5	1.79
Q-8	2 658	2 210	1.20
Q-9	453	415	1.09
Q-10	53	62	0.85

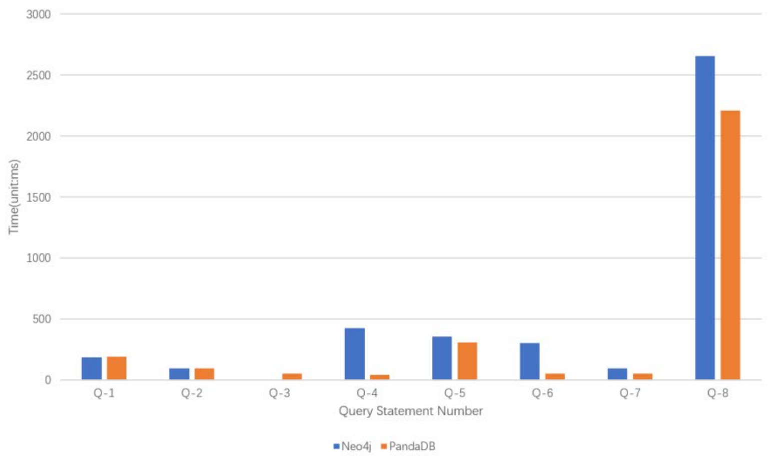


Figure 13 Comparison of query time in co-storage

3.3 Accelerated testing of unstructured data query

To accelerate the query of unstructured data, PandaDB adopts a method of constructing a semantic index to reduce the number of data transmissions and invoke AI services. In face detection scenarios, this section uses the LFW data set [11] to construct a comparative experiment and verifies the acceleration effect of the semantic index scheme by comparing the query time under various schemes.

The goal of this experiment is to find the face image with the highest similarity to the target face image from the sample set. This experiment is divided into four groups for comparison, and the details of the experimental content and conditions in each group are as follows.

- NOOP: No optimization. PandaDB sends a comparison request of image similarity to AIPM and sends the images one by one in the form of a BLOB stream to AIPM. AIPM extracts features after receiving the request and returns the comparison result to PandaDB.
- DLOC: Data localization scheme. PandaDB and AIPM are deployed on the same server to reduce the network overhead of BLOB transmission.
- AIIDX: AI service caching scheme. PandaDB requests data from AIPM, which responds to the request with local caching feature data. The implementation of data pipeline tools is imitated.
- SEMIDX: Semantic index scheme. The semantic index (face feature vector) of unstructured data is constructed in PandaDB. When the query is executed, the local index data is directly invoked for comparison.

Table 5 shows the time each scheme takes to execute the same query with different sample numbers. Compared to other schemes, SEMIDX scheme reduces data transmission and repeated data extraction, so in theory, this scheme is the fastest among the four schemes.

Table 5 shows that SEMEDX scheme has the shortest query time at the same number of samples. In the comparison of retrieval performance with more than 2 000 samples, the speedup ratio is more than 10 000.

Table 5 Comparison of query time in different methods

Number of samples	NOOP	DLOC		AIIDX		SEMIDX	
		Time consumption (ms)	Speedup ratio	Time consumption (ms)	Speedup ratio	Time consumption (ms)	Speedup ratio
1	993	999	0.99	124	8.04	5	199.80
5	1 636	1 641	1.00	672	2.44	10	164.10
10	3 708	3 482	1.06	1 299	2.68	6	580.33
20	6 854	7 036	0.97	2 448	2.87	7	1 005.14
50	16 336	15 665	1.04	6 169	2.54	8	1 958.13
100	32 138	30 714	1.05	12 291	2.50	10	3 071.40
500	163 154	156 025	1.05	61 505	2.54	22	7 092.05
1 000	320 760	307 251	1.04	122 525	2.51	33	9 310.64
2 000	643 462	615 047	1.05	244 661	2.51	60	10 250.78
5 000	1 616 986	1 542 788	1.05	615 192	2.51	144	10 713.81

3.4 Case: Entity disambiguation and visualization of academic knowledge graph

Academic knowledge graph generally refers to the domain knowledge graph with academic content as the main body. The typical applications are AMiner[12] and AceKG[13]. In this paper, a small academic knowledge graph is built based on the participant information[14] of KDD2020, which contains three types of nodes: papers, authors, and institutions. According to the paper creation relationship and the affiliation relationship between scholars and institutions, unstructured data such as authors' photos and the PDF full texts of papers are directly stored in PandaDB as properties.

Figure 14 shows the visualization of KDD2020 data (based on the open source project InteractiveGraph: https://github.com/grapheco/InteractiveGraph). Profile photo properties are stored in PandaDB, based on which, the function of "Search by Image" is provided, namely that fast matching based on profile photos is realized.

The same name of scholars and the same abbreviations of different institutions usually appear in the original data of an academic knowledge graph. Therefore, entity disambiguation is usually faced in creating the graph. At present, most entity disambiguation methods are based on clustering. Refs. [15–17] adopted surface feature values to calculate the similarity between entities, which cannot make full use of context features because the methods based on surface features face insufficient information. Some scholars tried to introduce external information such as the information in WikiPedia and used these knowledge resources as extended features of entities to assist in improving the accuracy of clustering, with Refs. [18,19] as representatives. Refs. [20,21] applied graph calculation to making full use of the structural information of network data, which did not have good universality. In recent years, with the tremendous progress of deep learning technology, some scholars have applied embedding technology[22] and neural network technology[23]. These methods have better effects than traditional methods but the accuracy still cannot meet people's expectations. This case proposes a disambiguation method that combines structured properties and unstructured properties based on PandaDB. As shown in Figure 15, there are two character nodes named Park Bill and Tom Green and a paper node named Data Vis in an academic knowledge graph. It is necessary to determine whether T. Green and Tom Green are the same entity, so as to determine whether there is a cooperative relationship between Park Bill and Tom Green. Due to insufficient property data in papers, disambiguation is difficult. The extraction of unstructured information and semantic comparison in PandaDB enable the full use of Tom Green's photos and the photo list of authors of the papers, thus realizing disambiguation. The lower part of Figure 15 shows the CypherPlus statement to complete the operation, where the '<:' operator represents containment relationship. Only the corresponding node needs to be found to judge whether *n.photo*<:*p.screenshot* is established or not, and then whether there is a cooperative relationship between the two can be calculated.

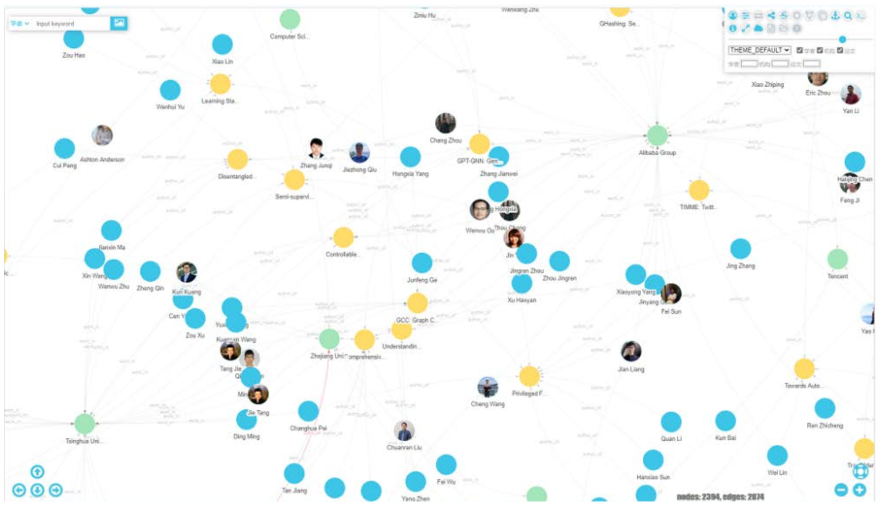
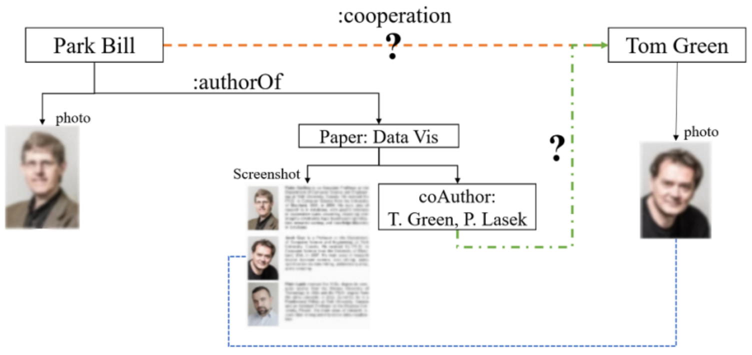


Figure 14 Visualization of KDD2020



```
MATCH(m), (p) Where n.name='Park Bill' and (m)-[:authorOf]->(p)
WITH m and p
MATCH(n),(p) WHERE n.photo <: p.screenshot
CREATE (n)-[:cooperation]-(m)
```

Figure 15 Entity disambiguation based on PandaDB

4 Related Work

In view of the unified management and analysis of multiple heterogeneous data in data applications, one method is to use ETL tools to process heterogeneous data in a unified form. However, this method is costly and changes in data and analysis requirements will invalidate[24] the original ETL process. Heterogeneous data management can support the storage and unified query of multi-model data. The concept of solution to the federal database was put forward in the 1980s, with Multibase[25] as the representative. The feature of this type of system is as follows: With defined global schema and mapping-based query language, users carry out query based on global schema, and then the system maps the query to the partitioned schema. Another type of system represented by Spark SQL[26] provides a unified API, which allows users to use the data as a relational model; as a result, it can improve the efficiency and simplify queries. BigIntegator[27], Forward[28], D4M[29] and other systems integrate NoSQL. Forward uses a JSON-based data model to organize data. D4M adopts an associative-array data model which is flexible and allows the system to query heterogeneous data.

In the field of data management, property graph model[30] is a commonly used data model for managing graph data. The nodes and relationships in the property graph can be given labels and associated properties in the form of any key-value pairs[31]. Property graph contains node and edge information without changing the overall structure of the graph. At present, the property graph model is widely accepted by the graph database industry[32,33], including the famous graph databases: Neo4j[34] and Titan[35]. Neo4j is a popular open source graph database, which has a rich ecosystem from native graph data storage to visualization plug-ins and then to graph data analysis plug-ins. JanusGraph[36] is a distributed graph database based on property graphs, which is developed on the basis of Titan. JanusGraph is designed to separate storage layer from query engine, in which Cassandra or HBase serves as the storage layer. JanusGraph implements retrieval functions with third-party distributed index libraries: ElasticSearch, Solr and Lucene. Other graph databases include Amazon's Neptune[37]. Microsoft's Azure CosmosDB[38], TigerGraph[39], and OrientDB[40].

Artificial intelligence technology has been widely applied to image recognition, speech recognition, machine translation and other fields. To understand different information such as voices and images, the research on multi-modal machine learning tries to integrate the model's understanding of unstructured data from the perspective of machine learning[41,42]. The crossover study between artificial intelligence and database has always been a hot topic in academic research. The integration of artificial intelligence and database has two directions[43]: AI4DB (AI for database) and DB4AI (database for AI). AI4DB aims to improve the efficiency and capability of DB through AI technology, such as automated database parameter optimization[44,45], cardinality estimation[46], index recommendation[47] and query optimization[48,49]. DB4AI provides data services for the training and learning of AI algorithms based on databases. For example, it provides users with customized functions to assist in building the model with the help of the unified SQL interface of databases; it helps model training through database tensor calculation; the AI model is made persistent for reuse.

With the advent of the era of big data, the stand-alone services have failed to meet the demand for storage and computing of massive amounts of data, and more and more tasks need the support of distributed systems. It is essential to ensure the reliability and consistency of distributed systems. A well-known algorithm to solve this problem is the Paxos algorithm[50,51] proposed by Lamport. Since then, researchers have proposed many new algorithms[52] on the basis of Paxos to adapt to different engineering environments. The famous ones are Multi-Paxos[51,53], VR (viewstamped replication) algorithm[54,55] proposed by Liskov *et al.*, ZAB (Zookeeper’s atomic broadcast) algorithm[56] designed by Yahoo, and Raft algorithm[57] proposed by Ongaro *et al*.

5 Conclusions and Prospects

From the perspective of the fusion management and ad-hoc query requirements of structured/unstructured data, this paper analyzes the current difficulties in unified representation and interactive query of the fusion management of multiple heterogeneous data. On this basis, this paper also proposes an extended property graph model which can achieve the unified representation of heterogeneous data, as well as property operators and query syntax for online query and calculation. In Section 2, this paper proposes PandaDB, a fusion management system of distributed data based on intelligent property graph models. The system realizes efficient storage management of structured/unstructured data and provides a flexible AI operator extension mechanism. As such, it has the ability to query the internal information of multiple heterogeneous data on an ad-hoc basis. Tests and cases prove that PandaDB performs well in large-scale property filtering queries and high-concurrency query responses. In addition, it can be applied to scenarios with the fusion management of multiple heterogeneous data such as the entity disambiguation and visualization of academic knowledge graphs.

Currently, PandaDB still has some disadvantages. On the one hand, the AIPM module deployed independent of the system reduces the coupling of the latter, which is conducive to system extension and maintenance. However, in case of large-scale information query requests of unstructured data, the information transmission between modules costs much. In the future, a more reasonable integration mechanism of AI functions should be studied, and task scheduling methods should be designed in combination with the scenario features of ad-hoc query for multiple heterogeneous data to improve system performance. On the other hand, the operation of extracting nested properties from intelligent properties currently lacks an effective caching and prediction mechanism, which causes a large delay in ad-hoc extraction. PandaDB will further combine applications to improve the performance and stability of the system, thus enhancing the fusion management of multiple heterogeneous data.

References

[1] Gantz J, Reinsel D. Extracting value from chaos. IDC Iview, 2011, 1142 (2011): 1−12.

[2] Liu Y, Li H, Garcia-Duran A, *et al*. MMKG: Multi-modal knowledge graphs. Proc. of the European Semantic Web Conf. 2019. 459−474.

[3] Buneman P, Davidson S, Fernandez M, *et al*. Adding structure to unstructured data. Proc. of the Int'l Conf. on Database Theory. Berlin, Heidelberg: Springer-Verlag, 1997. 336−350.

[4] Li W, Lang B. A tetrahedral data model for unstructured data management. Science China Information Sciences, 2010,40(8):1039–1053 (in Chinese with English abstract).[doi: 10.1007/s11432-010-4030-9]

[5] Gerber D, Hellmann S, Bühmann L, *et al*. Real-Time RDF extraction from unstructured data streams. Proc. of the Int'l Semantic Web Conf. Berlin, Heidelberg: Springer-Verlag, 2013. 135−150.

[6] Sears R, Van Ingen C, Gray J. To Blob or not to blob: Large object storage in a database or a file system? arXiv preprint cs/0701168, 2007.

[7] Zhu Y, Du N, Tian H, *et al*. LaUD-MS: An extensible system for unstructured data management. Proc. of the 2010 12th Int'l Asia-Pacific Web Conf. IEEE. 2010. 435−440.

[8] Zhang X,Du XY, Chen JC, *et al*. Managing a large shared bank of data by using Free-Table. Proc. of the 12th Asia-Pacific Web Conf. (APWeb 2010). Busan, 2010. 441−446.

[9] Zhou NN, Zhang X, Sun XY, *et al*. Design and implementation of adaptive storage management system in MyBUD. Journal of Frontiers of Computer Science & Technology, 2012, 6(8): 673−683.

[10] Francis N, Green A, Guagliardo P, *et al*. Cypher: An evolving query language for property graphs. Proc. of the 2018 Int'l Conf. on Management of Data. 2018. 1433−1445.

[11] Huang GB, Mattar M, Berg T, *et al*. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Workshop on Faces in Real-Life Images: Detection, Alignment, and Recognition. 2008.

[12] Wan H, Zhang Y, Zhang J, *et al*. Aminer: Search and mining of academic social networks. Data Intelligence, 2019,1(1): 58−76.

[13] Wang R, Yan Y, Wang J, *et al*. Acekg: A large-scale knowledge graph for academic data mining. Proc. of the 27th ACM Int'l Conf. on Information and Knowledge Management. 2018. 1487−1490.

[14] KDD2020. 2020. https://www.aminer.cn/conf/kdd2020/homepage.

[15] Bagga A, Baldwin B. Entity-Based cross-document coreferencing using the vector space model. Proc. of the 36th Annual Meeting of the Association for Computational Linguistics and 17th Int'l Conf. on Computational Linguistics, Vol. 1. 1998. 79−85.

[16] Pedersen T, Purandare A, Kulkarni A. Name discrimination by clustering similar contexts. Proc. of the Int'l Conf. on Intelligent Text Processing and Computational Linguistics. Berlin, Heidelberg: Springer-Verlag, 2005. 226−237.

[17] Chen Y, Martin JH. Towards robust unsupervised personal name disambiguation. Proc. of the 2007 Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL). 2007. 190−198.

[18] Cucerzan S. Large-Scale named entity disambiguation based on Wikipedia data. Proc. of the 2007 Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL). 2007. 708−716.

[19] Han X, Zhao J. Named entity disambiguation by leveraging Wikipedia semantic knowledge. Proc. of the 18th ACM Conf. on Information and Knowledge Management. 2009. 215−224.

[20] Hassell J, Aleman-Meza B, Arpinar IB. Ontology-Driven automatic entity disambiguation in unstructured text. Proc. of the Int'l Semantic Web Conf. Berlin, Heidelberg: Springer-Verlag, 2006. 44−57.

[21] Minkov E, Cohen WW, Ng AY. Contextual search and name disambiguation in email using graphs. Proc. of the 29th Annual Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. 2006. 27−34.

[22] Zhang B, Al Hasan M. Name disambiguation in anonymized graphs using network embedding. Proc. of the 2017 ACM on Conf. on Information and Knowledge Management. 2017. 1239−1248.

[23] Huang H, Heck L, Ji H. Leveraging deep neural networks and knowledge graphs for entity disambiguation. arXiv preprint arXiv:1504.07678, 2015.

[24] Tan R, Chirkova R, Gadepally V, *et al*. Enabling query processing across heterogeneous data models: A survey. Proc. of the 2017 IEEE Int'l Conf. on Big Data (Big Data). IEEE, 2017. 3211−3220.

[25] Smith JM, Bernstein PA, Dayal U, *etal*. Multibase: Integrating heterogeneous distributed database systems. Proc. of the National Computer Conf. 1981. 487−499.

[26] Armbrust M, Xin RS, Lian C, *et al*. Spark SQL: Relational data processing in spark. Proc. of the 2015 ACM SIGMOD Int'l Conf. on Management of Data. 2015. 1383−1394.

[27] Zhu M, Risch T. Querying combined cloud-based and relational databases. Proc. of the 2011 Int'l Conf. on Cloud and Service Computing. IEEE, 2011. 330−335.

[28] Ong KW, Papakonstantinou Y, Vernoux R. The SQL++ unifying semi-structured query language, and an expressiveness benchmark of SQL-on-Hadoop, NoSQL and NewSQL databases. CoRR, abs/1405.3631, 2014.

[29] Kepner J, Arcand W, Bergeron W, *et al*. Dynamic distributed dimensional data model (D4M) database and computation system. Proc. of the 2012 IEEE Int'l Conf. on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2012. 5349−5352.

[30] EhrigH,Prange U, Taentzer G. Fundamental theory for typed attributed graph transformation. Lecture Notes in Computer Science, 2004,3256:161−177.

[31] Robinson I, Webber J, Eifrem E. Graph Databases: New Opportunities for Connected Data. O'Reilly Media, Inc., 2015.

[32] Wang X, Zou L, Wang CK, Peng P, Feng ZY. Research on knowledge graph data management: A survey. Ruan Jian Xue Bao/Journal of Software, 2019, 30(7): 2139−2174 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5841.htm. [doi: 10.13328/j.cnki.jos.005841]

[33] Angles R. A comparison of current graph database models. Proc. of the 2012 IEEE 28th Int’l Conf. on Data Engineering Workshops. IEEE. 2012. 171−177.

[34] The Neo4j Team. The Neo4j manual v3.4. 2018. https://neo4j.com/docs/developer-manual/current/.

[35] Spmallette. Titan—Distributed graph database. 2018. http://titan.thinkaurelius.com/.

[36] JanusGraph Authors. JanusGraph—Distributed graph database. 2018. http://janusgraph.org/.

[37] Amazon Web Services, Inc. Amazon Neptune—Fast, reliable graph database build for cloud. 2018. https://aws.amazon.com/neptune/.

[38] Microsoft Azure. Microsoft Azure Cosmos DB. 2018. https://docs.microsoft.com/en-us/azure/cosmos-db/introduction.

[39] TigerGraph. TigerGraph—The first native parallel graph. 2018. https://www.tigergraph.com/.

[40] Callidus Software Inc. OrientDB-Multi-Model database. 2018. http://orientdb.com/.

[41] Baltrušaitis T, Ahuja C, Morency LP. Multimodal machine learning: A survey and taxonomy. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2018, 41(2): 423−443.

[42] Ramachandram D, Taylor GW. Deep multimodal learning: A survey on recent advances and trends. IEEE Signal Processing Magazine, 2017, 34(6): 96−108.

[43] Li GL, Zhou XH. XuanYuan: An AI-native database systems. Ruan Jian Xue Bao/Journal of Software, 2020,31(3):831−844. [doi: 10.13328/j.cnki.jos.005899]

[44] Hang J, Liu Y, Zhou K, Li G. An end-to-end automatic cloud database tuning system using deep reinforcement learning. Proc. of the SIGMOD. 2019.

[45] Wang W, Zhang M, Chen G, Jagadish HV, Ooi BC, Tan K. Database meets deep learning: Challenges and opportunities. SIGMOD Record, 2016, 45(2): 1722.

[46] Kipf A, Kipf T, Radke B, Leis V, Boncz PA, Kemper A. Learned cardinalities: Estimating correlated joins with deep learning. Proc. of the CIDR. 2019.

[47] Pedrozo WG, Nievola JC, Ribeiro DC. An adaptive approach for index tuning with learning classifier systems on hybrid storage environments. Proc. of the HAIS. 2018. 716−729.

[48] Krishnan S, Yang Z, Goldberg K, Hellerstein JM, Stoica I. Learning to optimize join queries with deep reinforcement learning. CoRR, abs/1808.03196, 2018.

[49] Marcus R, Papaemmanouil O. Deep reinforcement learning for join order enumeration. Proc. of the 1st Int’l Workshop on Exploiting Artificial Intelligence Techniques for Data Management. 2018. 3:13:4.

[50] Lamport L. The part-time parliament. ACM Trans. on Computer Systems, 1998, 16(2): 133−169.

[51] Lamport L. Paxos made simple. ACM SIGACT News, 2001, 32(4): 18−25.

[52] Wang J, Zhang MX, Wu YW, Chen K, Zheng WM. Paxos-Like consensus algorithms: A review. Journal of Computer Research and Development, 2019, 56(4): 692−707 (in Chinese with English abstract).

[53] Chandra TD, Griesemer D, Redstone J. Paxos made live: An engineering perspective. Proc. of the 26th Annual ACM Symp. on Principles of Distributed Computing (PODC 2007). New York: ACM, 2007. 398−407.

[54] Oki BM, Liskov BH. View stamped replication: A new primary copy method to support highly-available distributed systems. Proc. of the7th Annual ACM Symp. on Principles of Distributed Computing. 1988. 8−17.

[55] Liskov, Barbara, Cowling J. View stamped Replication Revisited. 2012.

[56] Medeiros A. ZooKeeper's atomic broadcast protocol: Theory and practice. Technical Report, 2012.

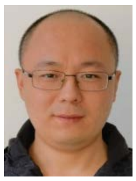
[57] Ongaro D, Ousterhout J. In search of an understandable consensus algorithm. Proc. of the 2014 USENIX Annual Technical Conf. 2014. 305−319.

Zhihong Shen, Ph.D., professor, doctoral supervisor. His research interests include big data, graph data management, and semantic net.

Zhongxin Liu, engineer. His research interests include distributed storage and database.

Zihao Zhao, Ph.D. candidate, CCF student member. His research interests include distributed graph database, and fusion data query.

Chuan Hu, master's degree, research interests: Knowledge graph visualization.



Huajin Wang, Ph.D., assistant professor. His research interests include distributed computing and big data analysis technology.

Yuanchun Zhou, Ph.D., professor, doctoral supervisor, senior member of CCF. His research interests include scientific big data and knowledge graph.