Efficient Query Processing in Integrated Multiple Object Databases With Maybe Result Certification*

Jia-Ling Koh
Department of Computer and Information Education
National Taiwan Normal University
Taipei, Taiwan, R.O.C.
E-mail: jlkoh@ice.ntnu.edu.tw

Arbee L.P. Chen
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 300, R.O.C.
E-mail: alpchen@cs.nthu.edu.tw

Abstract

Within integrated multiple object databases, missing data occurs due to the missing attribute conflict as well as the existence of null values. A set of algorithms is provided in this paper to process the predicates of global queries with missing data. For providing more informative answers to users, the maybe results due to missing data are presented in addition to the certain results. The local maybe results may become certain results via the concept of *object isomerism*. One algorithm is designed based on the *centralized approach* in which data are forwarded to the same site for integration and processing. Furthermore, for reducing response time, the *localized approaches* evaluate the predicates within distinct component databases in parallel. The object signature is also applied in the design to further reduce the data transfer. These algorithms are compared and discussed according to the simulation results of both the total execution and response times. Alternately, the global schema may contain multi-valued attributes with values derived from attribute values in different component databases. Hence, the proposed approaches are also extended to process the global queries involving this kind of multi-valued attribute.

Keywords: object database, schema integration, missing attribute, query processing, isomeric objects, maybe result certification.

---

* This work was partially supported by the Republic of China National Science Council under Contract No. NSC89-2213-E-003-006 and NSC89-2213-E-007-044.

1 Introduction

Recently, due to the rapid development of computer networks, data sharing among multiple databases is essential. Within a distributed heterogeneous system, which consists of object databases, a global object schema created by integrating schemas of the component databases provides a uniform interface as well as a high level location transparency for users to retrieve data. A variety of approaches have been proposed for schema integration [2], [3], [9], [12], [14], [24]. Batini *et al*. discussed twelve methodologies for database or view integration [2]. Czejdo *et al*. employed a language with graphical user interface to perform schema integration in federated database systems [9]. The issues on implementing schema integration tools were reported in [15] and [26]. Furthermore, to automate much of the integration process, the tools employed to detect similarities between structures in two schemas were embedded within the view integration process [14]. A formal semantic model for specifying the correspondences among schemas was proposed in [28]. Additionally, the operational mapping strategy was provided in [3] to integrate heterogeneous data management systems. Alternate approaches defined a set of operators to build a virtual integration of multiple databases or to customize virtual classes [23], [25].

In our previous work, an integration mechanism was proposed to achieve a global object schema for multiple object databases [20]. The *corresponding assertions* were designed to enable the database administrator (DBA) to specify semantic correspondences among component schemas. Based on these assertions, *integration rules*, which employed a set of primitive operators to perform the integration, were designed. Moreover, an equation form was defined to denote the mapping between the global and component schemas [21]. As well, against the constructed global schema, one mechanism of query processing was introduced in [6] and [21].

In the process of schema integration, the classes of the component schemas with the same semantics are integrated into a class in the global schema. In the following context, the classes in the global schema are called *global classes*. Furthermore, the classes, which are integrated into a global class, $\mathbf{C_g}$, are named the *constituent classes* of class $\mathbf{C_g}$. A global class consists of the attributes belonging to its constituent classes. The conflict of *missing attribute* occurs when semantically different attributes exist in the constituent classes [19]. Let $\mathbf{C_c}$ denote a constituent class of the global class $\mathbf{C_g}$. The attributes appearing in class $\mathbf{C_g}$

but not defined in class $\mathbf{C_c}$ are named *missing attributes* of $\mathbf{C_c}$ (we also say $\mathbf{C_c}$ *holds* the missing attributes). The data for the missing attributes in $\mathbf{C_c}$ are called *missing data* and considered to be *null* [11]. Suppose a missing attribute, *A*, of $\mathbf{C_c}$ is involved in predicates of a global query. The objects, which belong to class $\mathbf{C_c}$ and satisfy the predicates except the ones on *A*, are evaluated as *maybe results* [10]. Namely, *maybe results* are produced due to the existence of missing data. In addition to missing attributes, null values that existed originally in component databases are also one kind of missing data.

A real world entity is represented as an object in an object-oriented database. However, within a distributed heterogeneous object database system, an entity may exist in numerous object databases with incompatible local object identifiers (**LOids**). We proposed a strategy in [7] to discover the objects, named *isomeric objects*, which are stored in distinct databases but represent the same real world entity. Data of these isomeric objects must be integrated to provide complete information of an entity represented in the system, Therefore, after integrating with its isomeric objects in other component databases, an object evaluated as a maybe result within a component database may become a *certain result.*

Numerous distributed optimization algorithms have been proposed. A set of rules was provided in [4] to transform the operations in a global query into the ones with less cost. A global query may also be decomposed into subqueries which can be executed within local sites in parallel [5]. Semijoin operations were applied in [31] to reduce the cost of data transmission. In addition, determining the join sequence to decrease the transmission cost was investigated in [8]. Query processing strategies of homogeneous distributed object database systems were considered in [16]. None of the previous studies considered the missing data due to conflict of missing attribute. Furthermore, schema and domain incompatibilities in multidatabase systems were considered in [17] and [27]. Partial values and probabilistic partial values were proposed in [13] and [29], respectively, to solve the schema integration problems including missing data. However, the query processing algorithms were not discussed in these two papers.

In this paper, based on the object data model, the query processing strategy is proposed in particular for the situation when missing data are involved in predicates of global queries. In addition to certain results, the maybe results due to missing data are also provided in the query answer. The concept of object isomerism is applied in query processing such that the

local maybe results may become certain results. Therefore, users can obtain more informative answers from the results. A set of algorithms for global query processing is provided in this paper. An algorithm is designed based on the centralized approach in which the data are dispatched to the same site for both integration and processing. Moreover, for reducing the response time, the *localized approaches* [5] are employed to evaluate the query predicates in distinct component databases in parallel. The object signature is also applied to extend the localized approaches to reduce the data transfer. Finally, the proposed algorithms are compared and discussed through the simulation results.

Furthermore, within a heterogeneous object database system, the semantically equivalent attributes of the constituent classes are integrated into an attribute, A_m, of the global class after schema integration is performed. Assume it is permitted that various values exist in the semantically equivalent attributes of the isomeric objects. Thus, attribute A_m in the global class is a multi-valued attribute. For each constituent class $\mathbf{C_c}$, the objects therein contain only partial data of attribute A_m for representing a real world entity. Thus, A_m is named a *partial missing attribute* of the constituent classes. Suppose the partial missing attribute A_m of constituent class $\mathbf{C_c}$ is involved in predicates of a global query. The objects, which belong to class $\mathbf{C_c}$ and satisfy the predicates except the ones on A_m, are evaluated as maybe results. Notably, this is similar to missing attributes involved in global queries. Therefore, in this paper, the proposed algorithms employed to process the global queries with missing attributes are also extended to process the global queries involving partial missing attributes.

This paper is organized as follows. The next section clarifies the varieties of missing data considered in this paper. As well, centralized and localized approaches for processing global queries that involve missing data are also introduced through examples. Section 3 displays three basic global query processing algorithms as well as two algorithms with an auxiliary structure. Section 4 presents the performance study for the algorithms. To process the global queries that involve partial missing attributes, Section 5 presents the extended approaches. Finally, Section 6 concludes this paper with a discussion of future works.

2 Missing Data and Global Query Processing

2.1 Missing Data and Missing Attributes

In this subsection, an example to illustrate the varieties of missing data in a heterogeneous object database system is provided. Figure 1 displays the schemas of three databases: **DB1**, **DB2** and **DB3**, which are located in distinct sites and employed to store personal information of a school. Figure 2 displays the global schema that results from integrating these three component schemas.

Within constituent classes, missing attributes yields missing data. According to their types, missing attributes of a class are dubbed either *primitive* or *complex missing attributes*. For example, *age* of class **Student@DB2** as well as *specialty* of both **Teacher@DB1** and **Teacher@DB3** are primitive missing attributes. The attributes *address* and *department* are complex missing attributes of class **Student@DB1** and **Teacher@DB2**, respectively. A complex missing attribute, A_c, of class $\mathbf{C_c}$ implies that the data of all the nested attributes rooted at the domain class of A_c are also missing for class $\mathbf{C_c}$.

Null values that existed originally in objects are another source of missing data. If an attribute value of an object is null, it is a missing attribute of the object. For example, suppose the attribute value of an object **s5** in **Student@DB1** is (824308, Jack, -, **t1**, male), where **t1** is an object identifier in **Teacher@DB1** and "-" denotes a null value. Then *age* is a primitive missing attribute of **s5** but not one of class **Student@DB1**. A null value may also occur in a complex attribute of an object, which results in a complex missing attribute of the object.

The global queries considered herein are as follows:

Definition 1. Given a global query, ***Q***, denoted as <***S***, ***R***, ***P***>, where ***S***, ***R*** and ***P*** denote the target, range, and predicate clauses of the query, respectively.

The range clause, ***R***, includes a global class, **C**, which is the sole *range class* of the query.

The target clause, ***S***, consists of $<S_1>$, $<S_2>$, ..., and $<S_m>$, where each $<S_i>$ (i = 1, …, m) is a path expression that denotes a *target attribute* as the following form:

$<S_i>$:= $\mathbf{C}.A^i_1.A^i_2. \ldots .A^i_{u_i}$ ($u_i \geq 1$, and $A^i_1, \ldots, A^i_{u_i}$ are nested attributes of **C**).

In addition, the predicate clause, ***P***, consists of predicates $<P_1>$, $<P_2>$, ..., and $<P_n>$, where each $<P_j>$ (j = 1, …, n) is a predicate as the following form:

<Pj>:= $\mathbf{C}.A^j_1.A^j_2. \ldots .A^j_{v_j}$ <comparator> <constant-value>

($v_j \geq 1$, and $A^j_1, \ldots, A^j_{v_j}$ are nested attributes of **C**).

The query $\boldsymbol{Q}$ = <$\boldsymbol{S}$, $\boldsymbol{R}$, $\boldsymbol{P}$> is formulated in SQL/X [30] as:

SELECT <S_1>, <S_2>, ... , <S_m>

FROM **C**

WHERE <P_1> **and** <P_2> ... **and** <P_n>.

Definition 2. Given a global query $\boldsymbol{Q}$ = <$\boldsymbol{S}$, $\boldsymbol{R}$, $\boldsymbol{P}$>.

Let $\mathbf{S_{bc}}$ = { $\mathbf{C_d}$ | <Pj> in $\boldsymbol{P}$ where <Pj> := $\mathbf{C}.A^j_1.\ A^j_2.\ ...\ .A^j_{vj}$ <comparator> <constant-value>,
and ∃ k (v_j>k≥1 and $\mathbf{C_d}$ is the domain class of $\mathbf{C}.A^j_1.\ A^j_2.\ ...\ .A^j_k$)}.

C specified in $\boldsymbol{R}$ is named the *root class* of query $\boldsymbol{Q}$. The classes in $\mathbf{S_{bc}}$ are named *branch classes* of the query. In addition, the constituent classes of a root class and a branch class are called *local root classes* and *local branch classes*, respectively.

Within the predicates of a global query, the missing attributes of the constituent classes are involved in four distinct situations:

(1) a primitive missing attribute of a local root class is involved,

(2) a complex missing attribute of a local root class is involved,

(3) a primitive missing attribute of a local branch class is involved,

(4) a complex missing attribute of a local branch class is involved.

Figure 2 presents the constructed global schema. From which the query **Q1** "Retrieve the names of students who are female and older than 25 years old." is formulated in SQL/X and demonstrated in Fig. 3(a). Notably, in **DB2**, *age* is a primitive missing attribute of the local root class **Student**.

Figure 3(b) depicts another query, **Q2**, "Retrieve the name of students living in Taipei together with the advisor's name provided they are teaching in department of computer science and whose specialty is database." Both the missing attributes of local root and local branch classes are included in the predicates of **Q2**. In **DB1**, *address* is a complex missing attribute of the local root class **Student** and *specialty* is a primitive missing attribute of the local branch class **Teacher**. Additionally, in **DB2**, the local branch class **Teacher** holds a complex missing attribute *department*.

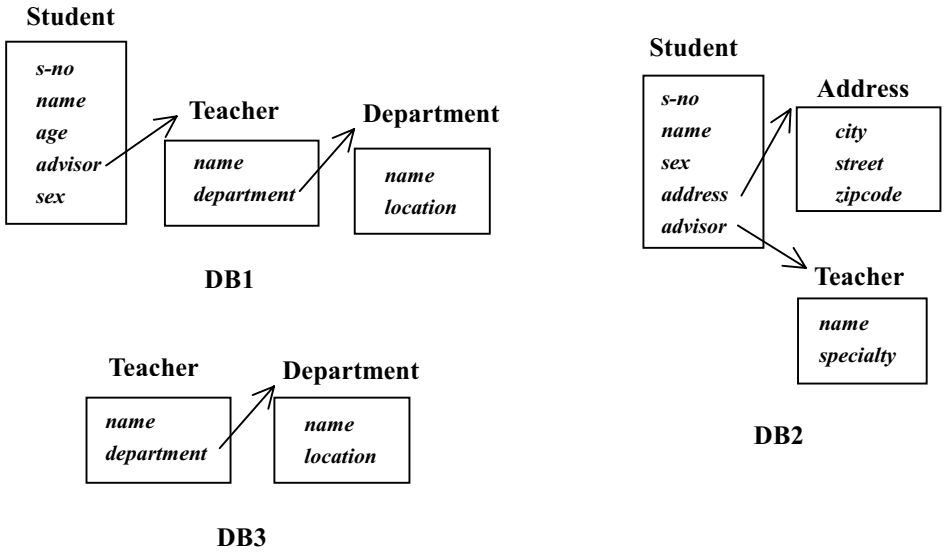

Figure 1: The component schemas

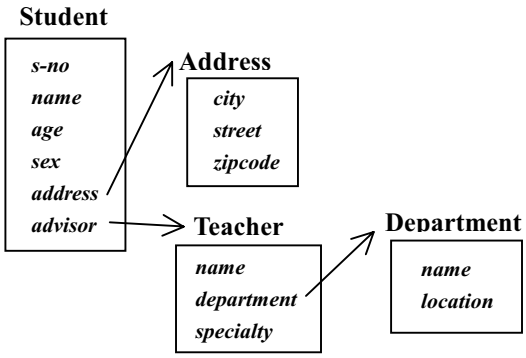

Figure 2: The constructed global schema

Q1: *Select* **X**.*name*
From **Student X**
Where **X**.*age*>25 **and X**.*sex*=Female

(a)

Q2: *Select* **X**.*name*, **X**.*advisor.name*
From **Student X**
Where **X**.*address.city*=Taipei
and X.*advisor.specialty*=database
and X.*advisor.department.name*=CS

(b)

Figure 3: Example queries

Student

LOid	*s-no*	*name*	*age*	*advisor*	*sex*
s1	804301	John	31	**t1**	male
s2	798302	Tony	28	**t3**	male
s3	808301	Mary	24	**t2**	female
s4	794305	Amy	29	**t1**	-

Teacher

LOid	*name*	*department*
t1	Jeffery	**d1**
t2	Abel	**-**
t3	Haley	**d1**
t4	Byron	**d1**

Department

LOid	*name*	*location*
d1	CS	building A
d2	EE	building E
d3	FL	building B
d4	MA	building C

(a) The object instances in **DB1**

Student

LOid	*s-no*	*name*	*sex*	*address*	*advisor*
s1'	762315	Hedy	female	**a1'**	**t1'**
s2'	804301	John	male	**a2'**	**t3'**
s3'	794305	Amy	female	**a1'**	**t3'**
s4'	828307	Fanny	female	**a3'**	**t2'**

Teacher

LOid	*name*	*specialty*
t1'	Kelly	database
t2'	Byron	database
t3'	Jeffery	network

Address

LOid	*city*	*street*	*zipcode*
a1'	Taipei	Part	100
a2'	Kaohsiung	Horber	800
a3'	Taipei	University	100

(b) The object instances in **DB2**

Teacher

LOid	*name*	*department*
t1"	Haley	**d1"**
t2"	Abel	**d1"**
t3"	Kelly	**d2"**

Department

LOid	*name*	*location*
d1"	EE	building E
d2"	CS	-
d3"	PH	building D

(c) The object instances in **DB3**

Figure 4: The object instances in component databases

Student

GOid	gs1	gs2	gs3	gs4	gs5	gs6
LOid@DB1	s1	s2	s3	s4		
LOid@DB2	s2'			s3'	s1'	s4'

Address

GOid	ga1	ga2	ga3
LOid@DB2	a1'	a2'	a3'

Teacher

GOid	gt1	gt2	gt3	gt4	gt5	gt6
LOid@DB1	t1	t2	t3	t4		
LOid@DB2	t3'			t2'	t1'	
LOid@DB3		t2''			t3''	t1''

Department

GOid	gd1	gd2	gd3	gd4	gd5
LOid@DB1	d1'	d2'	d3'	d4'	
LOid@DB3	d2''	d1''			d3''

Figure 5: The **GOid** mapping tables

Student

GOid	*s-no*	*name*	*age*	*advisor*	*sex*	*address*
gs1	804301	John	31	**gt1**	male	**ga2**
gs2	798302	Tony	28	**gt3**	male	**-**
gs3	808301	Mary	24	**gt2**	female	**-**
gs4	794305	Amy	29	**gt1**	female	**ga1**
gs5	762315	Hedy	-	**gt5**	female	**ga1**
gs6	828307	Fanny	-	**gt4**	female	**ga3**

(a) The object instances for global class **Student** when materialized

Teacher

GOid	*name*	*department*	*specialty*
gt1	Jeffery	**gd1**	network
gt2	Abel	**gd2**	-
gt3	Haley	**gd1**	-
gt4	Byron	**gd1**	database
gt5	Kelly	**gd1**	database
gt6	Haley	**gd2**	-

Department

GOid	*name*	*location*
gd1	CS	building A
gd2	EE	building E
gd3	FL	building B
gd4	MA	building C
gd5	PH	building D

Address

GOid	*city*	*street*	*zipcode*
ga1	Taipei	Park	100
ga2	Kaohsiung	Horber	800
ga3	Taipei	University	100

(b) The object instances for other global classes when materialized

Figure 6: The object instances for global classes when materialized

2.2 A Centralized Scheme for Processing Missing Data

The basic schemes for processing missing data include a centralized and a localized approach, which are discussed in this and the next subsections, respectively. The major concepts of these approaches are explained through the example queries presented in Subsection 2.1. The attribute values of the object instances in the three component databases are illustrated as Figs. 4(a), (b) and (c), respectively. Notably, the value denoted in bold is the **LOid** of an object.

As proposed in the Pegasus heterogeneous multidatabase system [1], each object within the distributed system is assigned a global object identifier (**GOid**) herein. Furthermore, the isomeric objects have been identified [7] and the **GOids** for the isomeric objects are the same. Figure 5 demonstrates that the mappings among **LOids** and **GOids** are stored in the *GOid mapping tables*. Identification is executed periodically to recognize the isomeric objects of the newly added objects. Then the GOid mapping tables are updated for each new identification.

First, by considering the processing for the example query **Q1**, where only missing attributes of the local root class are included in the predicates, the centralized approach is introduced. As the name of the centralized approach implies, all the objects within the local root classes are forwarded to the same site for predicate evaluation. Thus, in this example, the objects in **Student@DB1** and **Student@DB2** are sent to the global processing site. Consequently, the objects in these two classes are integrated by an outerjoin over the **GOid**. Moreover, **LOids** that represent complex attribute values are transformed to their respective **GOids**. Figure 6(a) presents the integrated result. Possibly, the isomeric objects of an object provide the value of its missing data. For example, from its isomeric object **s1**, object **s2'** obtains *31* for its missing attribute *age*. Then, the global query is processed against these integrated objects. Finally, the certain result, *Amy*, is obtained, and due to the null *age* values, *Hedy* and *Fanny* are the maybe results.

Detailed processing is required when missing attributes of the local branch classes are also involved in the predicates. When processing **Q2**, the objects in the local root classes **Student@DB1** and **Student@DB2** as well as the local branch classes, including **Teacher@DB1**, **Teacher@DB2**, **Teacher@DB3**, **Department@DB1**, **Department@DB3**, and **Address@DB2** are forwarded to the global processing site. For each global class

involved in the predicates, the objects in the constituent classes are integrated via outerjoin operations over **GOid**. The integrated result is displayed in Fig. 6. Finally, the query result of **Q2** includes the certain results (*Hedy, Kelly*) and (*Fanny, Byron*), as well as the maybe result (*Tony, Haley*).

2.3 A Localized Scheme for Processing Missing Data

The basic concept of the localized approach is to decompose a global query into local queries against the component databases, such that the predicate evaluation is localized to achieve inter-site parallelism.

Definition 3. Given a global query $\boldsymbol{Q} = <\boldsymbol{S}, \boldsymbol{R}, \boldsymbol{P}>$.

The *unsolved predicates* of $\boldsymbol{Q}$ on component database $\boldsymbol{D_i}$ are defined to be:

$\boldsymbol{P_u(Q, D_i)}$:={<Pj>|<Pj> in $\boldsymbol{P}$ where <Pj>:= $\mathbf{C}.A^j_1.A^j_2. \ldots .A^j_{vj}$ <comparator> <constant-value>, and $\exists$ k ($v_j \geq k \geq 1$ and A^j_k specified in<Pj> is a missing attribute of a constituent class within $\boldsymbol{D_i}$)}

In addition, the *local predicates* of $\boldsymbol{Q}$ on component database $\boldsymbol{D_i}$ are defined to be:

$\boldsymbol{P_l(Q, D_i)}$:= {<Pj> | <Pj> is specified in $\boldsymbol{P}$ but not in $\boldsymbol{P_u}$ $\boldsymbol{(Q, D_i)}$ }.

The *local targets* of $\boldsymbol{Q}$ on component database $\boldsymbol{D_i}$ are defined to be:

$\boldsymbol{S_l(Q, D_i)}$:= {<Si> | The target attribute <Si> is specified in $\boldsymbol{S}$ and defined in $\boldsymbol{D_i}$ }.

For each component database $\boldsymbol{D_i}$, the local root class in component database $\boldsymbol{D_i}$ is denoted as $\mathbf{C}_i$. Then a local query $\boldsymbol{Q_i} = < \boldsymbol{S_l(Q, D_i)} \cup \mathbf{LOid}, \mathbf{C}_i, \boldsymbol{P_l(Q, D_i)}>$ is constructed, which can be evaluated in component database $\boldsymbol{D_i}$. In order to integrate the local results later, **LOids** of the qualified objects are also projected.

Definition 4. Given a global query $\boldsymbol{Q} = <\boldsymbol{S}, \boldsymbol{R}, \boldsymbol{P}>$.

The *unsolved predicates* of $\boldsymbol{Q}$ on an object $\boldsymbol{o}$ in component database $\boldsymbol{D_i}$ are defined to be:

$\boldsymbol{P_u(Q, o)}$:= { <Pj> | <Pj> in $\boldsymbol{P}$ and <Pj> involving a missing attribute of $\boldsymbol{o}$}

The unsolved predicates on an object, $\boldsymbol{o}$, can be evaluated if one of its isomeric objects contains the missed value of the missing data. Such an isomeric object is called an *assistant object* of $\boldsymbol{o}$, which can be discovered by examining the **GOid** mapping tables and component schemas. To denote all the assistant objects of $\boldsymbol{o}$, $\boldsymbol{O_a(o)}$ is employed.

Q1': *Select* **X**.*Oid*, **X**.*name*
From **Student@DB1 X**
Where **X**.*age*>25 and **X**.*sex*=Female

Q1'': *Select* **X**.*Oid*, **X**.*name*
From **Student@DB2 X**
Where **X**.sex=Female

(a)

Q2': *Select* **X**.*Oid*, **X**.*name*, **X**.*advisor.name*
From **Student@DB1 X**
Where **X**.*advisor.department.name*=CS

Q2'': *Select* **X**.*Oid*, **X**.name, **X**.*advisor.name*
From **Student@DB2 X**
Where **X**.*address.city*=Taipei
and X.*advisor.speciality*=database

(b)

Figure 7: The associated local queries for the example queries **Q1** and **Q2**

Let object ***o*** denote a maybe result in the local results. There must exist at least one unsolved predicate on a missing attribute of ***o***, and ***o*** is defined as *unsolved*. Notably, if ***o*** is solved, the local maybe result becomes a certain result. An unsolved object can become *solved* according the following rule.

[Certification Rule 1]
An unsolved object ***o*** is judged to become a solved object if
$\forall\, p \in P_u(Q, o)$ ($\exists\, o' \in O_a(o)$ and o' satisfying p).
Object ***o*** is eliminated from the query result if
$\exists\, p \in P_u(Q, o)$ ($\exists\, o' \in O_a(o)$ and o' not satisfying p).

Herein, applying this rule on an unsolved object is called to *certify* the unsolved object. Furthermore, to *check* an assistant object means to evaluate an unsolved predicate on the assistant object.

Hence, **Q1** is processed as follows: **Q1'** and **Q1''** are constructed containing the associated local predicates. These two local queries are forwarded to **DB1** and **DB2** for processing, respectively (Fig. 7(a)). The result of **Q1'** is a maybe result (**s4**, *Amy*) with an unsolved predicate on *sex* due to a null value. Alternately, the results of **Q1''** are (**s1'**, *Hedy*), (**s3'**, *Amy*), and (**s4'**, *Fanny*), which are maybe results with an unsolved predicate on missing attribute *age*. Notably, **Student@DB1** and **Student@DB2** provide the absent values of missing attributes for each other and their respective unsolved predicates are evaluated from the other. Therefore, the maybe results can be certified by discovering the isomeric objects among the local maybe results, which can be achieved by performing the **GOids** intersection.

Finally, the certain result is *Amy* whereas *Hedy* and *Fanny* remain maybe results.

Using the same approach, **Q2** is decomposed into two local queries **Q2'** and **Q2"** (Fig. 7(b)). The maybe results obtained from **Q2'** are (**s1**, John, Jeffery), (**s2**, Tony, Haley), (**s3**, Mary, Abel), and (**s4**, Amy, Jeffery) with unsolved predicates on missing attributes *address* and *advisor.speciality*, as well as an unsolved predicate on *advisor.department* for **s3**. (**s1'**, Hedy, Kelly) and (**s4'**, Fanny, Byron) are maybe results obtained from **Q2"** with an unsolved predicate on missing attribute *advisor.department*. Finally, the maybe results identified by **s2**, **s3**, **s1'** and **s4'** are produced, and **s1** and **s4** eliminated. However, the experimental results are not entirely correct. That is, objects **s1'** and **s4'** should be certain results and **s3** eliminated. This inaccuracy is due to the involved missing attributes within **DB2** are located in the local branch class, **Teacher**. Although objects **s1'** and **s4'** fails to have assistant objects in **Student@DB1**, their complex attribute *advisor* have assistant objects within **Teacher@DB1** and **Teacher@DB3**, which are not checked. Similarly, the incorrect retrieval of **s3** also occurs. Therefore, when a missing attribute of a local branch class is involved in the global predicates, additional processing is required.

For a maybe result o_m, suppose an unsolved predicate exists on a missing attribute of its nested complex attribute A_c. Let o_{nc} denote the attribute A_c of o_m, which is an *unsolved item* of the maybe result, o_m. Notably, if o_{nc} satisfies the certification, it becomes a *solved item*. If o_m and all its unsolved items are solved, the local maybe result o_m becomes a certain result. However, o_m is eliminated from the results if any of its unsolved items are eliminated. Under other conditions, o_m remains a maybe result.

[Certification Rule 2]
An unsolved item o_{nc} is judged to become a solved item if

$\forall\, p \in P_u(Q, o_{nc})$ ($\exists\, o' \in O_a(o_{nc})$ and o' satisfying p).

o_m is eliminated if

$\exists\, p \in P_u(Q, o_{nc})$ ($\exists\, o' \in O_a(o_{nc})$ and o' not satisfying p).

Therefore, the local queries should project **LOid** of every nested complex attribute whose domain class holds the missing attributes involved in the predicates. Then, according to the classes to which they belong, **LOids** of these assistant objects are collected and

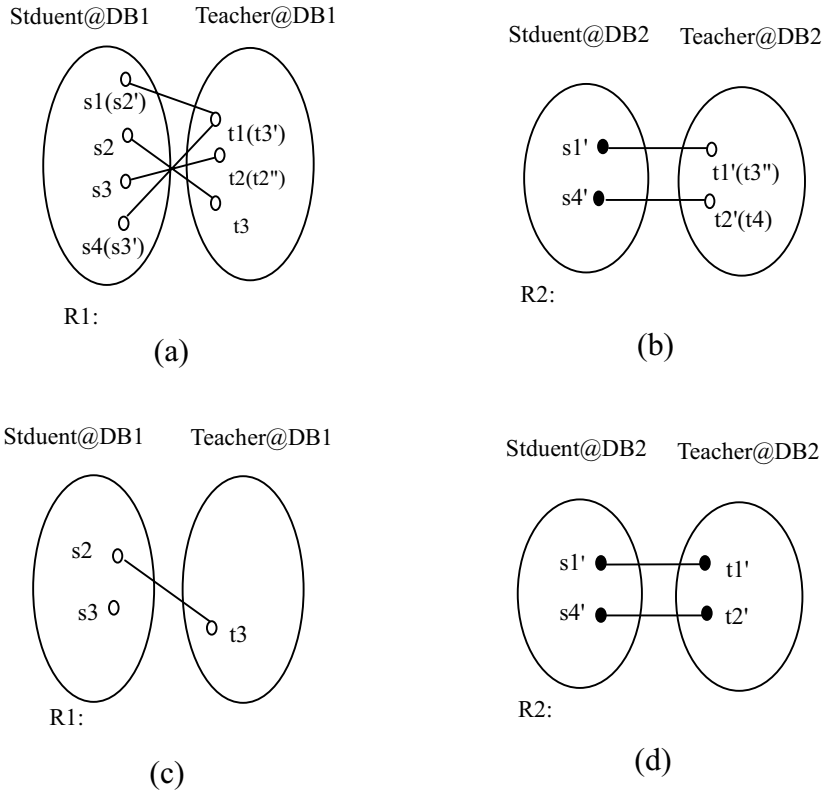

Figure 8: The local results for processing **Q2**

forwarded to the corresponding component databases. Moreover, the associated unsolved predicates are also dispatched with the **LOids** in order to check the assistant objects. The local maybe results and their unsolved items are forwarded to the global processing site, and await checking results of assistant objects to certify the unsolved items. Finally, after collecting the certification results of unsolved items and unsolved maybe results, the final certain results and maybe results are obtained.

Accordingly, local queries, **Q2'** and **Q2"**, are modified to also project the nested complex attribute *advisor*. Herein, the local maybe results **R1** and **R2** are denoted by the object graphs as Figs. 8(a) and (b), respectively. The unsolved maybe results and unsolved items are represented by white dots. Alternately, the black dots denote the solved maybe results and solved items. The **LOids** within the parentheses represent the assistant objects of the unsolved maybe results and unsolved items. For example, for the unsolved items in **Teacher@DB1**, **t3'** and **t2"** are the assistant objects for **t1** and **t2**, respectively. Therefore, **LOid** of the assistant object of **t1**, **t3'**, is dispatched to **DB2** with the predicate *"specialty=database"*. Similarly, **LOid t2"** is forwarded to **DB3** for the unsolved item **t2**

with the predicate on *department*. Notably, for **t2**, the data of missing attribute *specialty* is not provided via any assistant object. **R1**, the local result from **DB1**, is then forwarded to the global processing site and awaits checking results of **t3'** and **t2"**. Since no assistant object satisfies the checking, the unsolved items **t1** and **t2** are eliminated. The other unsolved items remain unsolved because they do not have corresponding assistant objects for checking. Moreover, because their assistant objects are not in the local result of **DB2,** the unsolved maybe results **s1** and **s4** are eliminated. Figure 8(c) presents the certification results. Identified by **s2**, the obtained local result is a maybe result (*Tony, Haley*). The same processing is applied on **R2** to check the assistant objects **t3"** and **t4**. Finally, Fig. 8(d) illustrates that the certain results (*Hedy, Kelly*) and (*Fanny, Byron*) are obtained.

According to the above discussion, the centralized scheme is similar to the query processing strategy applied for heterogeneous relational database systems. However, a different issue is considered in our localized scheme. Although path traversal can be supported by join operations, object-oriented databases support index structures specifically for path expressions. When constructing subqueries, the efficiency of query processing is ensured if path expression is applied. Therefore, the strategy proposed in the localized scheme is different from considering the join orders and join strategies as discussed in heterogeneous relational database systems. Moreover, missing data processing is the major concern of this work.

3 Processing Algorithms

Based on the discussion in Section 2, the following three phases are necessary to process the queries that involve missing data of component databases.

- *phase* **O**: this phase examines the **GOid** mapping tables to determine the assistant objects. As well, for the localized approaches, the assistant objects are also checked in this phase.
- *phase* **I**: this phase integrates the information of an object and its isomeric objects. For the localized approaches, it is the step that certifies the local maybe results as well as the unsolved items.
- *phase* **P**: the predicate evaluation is performed in this phase. Potentially, when the localized approaches are applied, the local predicates are concerned.

Note that phase **I** has to be executed after phase **O**.

In the subsequent subsections, three basic global query processing algorithms are provided via analyzing the combination of these three phases. Moreover, based on the auxiliary structure for storing object signatures, two additional various algorithms are introduced.

3.1 Basic Algorithms

3.1.1 Centralized Approach (CA)

The centralized approach illustrated in Subsection 2.2 follows the **O** → **I** → **P** order to execute the three essential phases. The procedures executed in the global processing site as well as each component database are as follows:

[global processing site]

Step CA_G1: Send a request to component databases to retrieve objects of the local root and local branch classes, then await the response.

Step CA_G2: To materialize the root and branch classes of the query, outerjoin is performed over join attribute **GOid** to integrate objects of the constituent classes (phase **O** and phase **I**).

Step CA_G3: Evaluate the predicates on the materialized global classes (phase **P**).

[component database]

Step CA_C1: When receiving the request from the global processing site, retrieve and forward all objects in the local root and local branch classes of the query to the global processing site.

Figure 9(a) presents the executing sequence among these steps. Since transferring cost is a concerned issue, the retrieved objects in **step CA_C1** are projected on **LOid** and the attributes involved in the query before being transferred to the global processing site.

3.1.2 Basic Localized (BL) Approach

Subsection 2.3 illustrates the basic localized approach. The execution order for the necessary phases is **P** → **O** → **I**. The tasks of the global processing site as well as the component databases are described as follows:

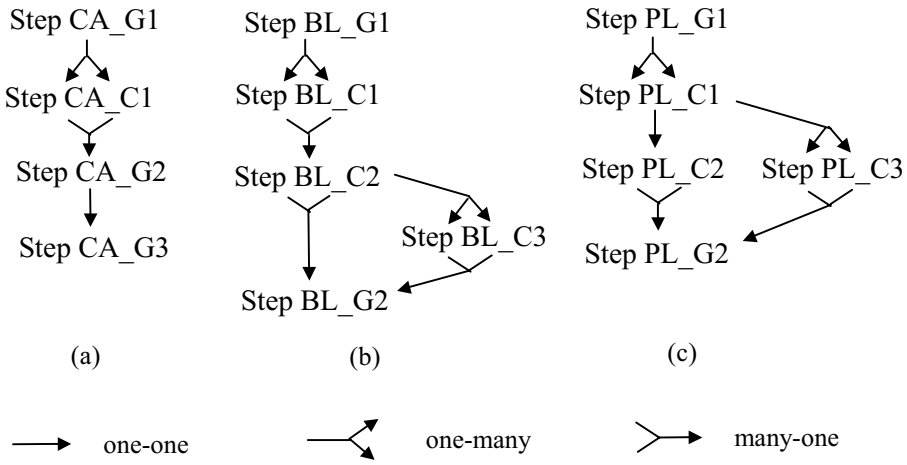

Figure 9: The executing flows of the processing algorithms.

[global processing site]

Step BL_G1: For each component database D_i containing the local root class, a local query Q_i against the local root class is produced. Then the query Q_i as well as unsolved predicates $Pu(Q, D_i)$ are dispatched to component database D_i and the response is awaited.

Step BL_G2: According to the certification rule, the local results are certified via checking the assistant objects (phase **I**). The final certain results and maybe results are thus obtained.

[component database]

Case 1: The local query dispatched from the global processing site is received.

Step BL_C1: The local predicates in the local query are evaluated (phase **P**).

Step BL_C2: Examine the **GOid** mapping tables to discover **LOids** of the assistant objects for the unsolved items of the local maybe results. Notably, these **LOids** and the corresponding unsolved predicates are sent to other associated component databases for further checking (part of phase **O**). The local result is then returned to the global processing site.

Case 2: The request from other component databases to check the assistant objects is received.

Step BL_C3 According to the received **LOids**, the corresponding objects are retrieved and evaluated via the unsolved predicates. The **LOids** of the satisfied objects are returned to the global processing site (phase **O**).

Figure 9(b) demonstrates the entire executing sequence for these tasks.

3.1.3 Parallel Localized (PL) Approach

The execution order for the parallel localized approach is **O** → **P** → **I**. This approach enables checking of the assistant objects and local predicates evaluation to be executed in parallel within different component databases.

[global processing site]

Step PL_G1: This step is identical to step **BL_G1**.

Step PL_G2: This step is identical to step **BL_G2**.

[component database]

Case 1: The local query forwarded from the global processing site is received.

Step PL_C1: For any object, ***o***, in the local root class, **LOids** of its unsolved items are retrieved. There exists at least one unsolved predicate on these retrieved objects. The **GOid** mapping tables are examined to discover the **LOids** of the assistant objects for these objects. Forward these **LOids** as well as the corresponding unsolved predicates to the associated component databases (part of phase **O**).

Step PL_C2: This step is the same as step **BL_C1** (phase **P**). The local results are then forwarded to the global processing site.

Case 2: The request from other component databases to check the assistant objects is received.

Step PL_C3: This step is identical to step **BL_C3** (phase **O**).

The difference between the **PL** approach and the **BL** approach is the execution order for the tasks of Case 1 within the component databases. Moreover, steps **PL_C2** and **PL_C3** are executed in parallel (Fig. 9(c)).

3.2 Algorithms with an Auxiliary Structure

Object signature of an object is constructed by all primitive attribute values of the object. To combine the hashing results of the primitive attribute values, the superimposed coding method is performed. Signature file indexing scheme is applied to accelerate query

processing because query predicates on an object can be evaluated via object signature without retrieving its actual values [32]. That is, only qualified objects require further verification using the actual data. To apply signature file indexing in the algorithms, it is assumed that the following auxiliary structure is supported. For any object, ***o***, with missing data, the object signatures of its assistant objects are stored at the site where ***o*** is located. Thus, by examining the object signatures of these assistant objects, the unsolved predicates can be processed locally. Only when the object signatures satisfy the predicates, does a request to check the assistant objects need to be dispatched to other component databases. The amount of data transfer can thus be reduced.

An additional phase is designed as follows:

- phase **F**: this phase evaluates the unsolved predicates on object signatures of the assistant objects. If an object signature does not satisfy the predicates, the corresponding assistant object need not to be sent out for further checking. Moreover, the associated unsolved item is eliminated.

This phase is applied in the two localized approaches prior to phase **O** to form two additional algorithms.

Basic Localized approach with Signature (BLS)

The basic localized approach with signature is designed by inserting phase **F** into step **BL_C2** during the basic localized approach. The executing order of the four phases is **P** → **F** → **O** → **I**.

Parallel Localized approach with Signature (PLS)

Similar to the previous approach, the parallel localized approach with signature derives from the parallel localized approach with phase **F** inserted into step **PL_C1**. The execution order **F** → **O** → **P** → **I** is followed.

4 Performance Study

In order to evaluate the performance of the proposed algorithms, a simulation experiment is implemented to compute the total execution time and response time of the associated algorithms. The simulation is divided into two parts.

1. The performance of algorithms **CA**, **BL** and **PL** are measured.
2. Demonstrating the effect of the performance when object signatures are applied in the localized approaches.

4.1 Simulation Design

The simulation model consists of a number of component database management systems (DBMSs) connected by a communication network. Each component DBMS contains a processor, a memory, and a hard disk. Moreover, the **GOid** mapping tables are replicated at each site.

The system parameters are displayed in Table 1. Table 2 lists the database and query parameters as well as the default setting for these parameters. To estimate selectivity of a set of predicates, the predicates are assumed to be not totally independent on each other. Therefore, the decreasing rate of the estimated selectivity is set to be an exponential order proportion to the root square of the number of predicates. In this experiment, the setting values of these parameters are adjusted to observe the performance of the proposed algorithms. According to each established setting range, 500 sets of parameter values are generated. For each proposed algorithm, the total execution time and response time for executing the generated samples are computed and averaged to represent the times for this setting.

4.2 Cost Terms

To estimate the performance, the time spent in the algorithms is denoted as *cost terms,* which are divided into three groups (Table 3). Hereinafter, the abbreviation represented as the upper index of a cost term denotes that the term is considered for the associated algorithm. The index i enclosed in the parentheses following a cost term indicates that the cost term represents that spent to process the objects in component database i. Moreover, a cost term followed by index (i,j) denote the cost spent in processing within component database, j, that is resulted from the request of database i. The cost functions that estimate the cost terms are listed in the Appendix.

Table 1: The system parameters

parameters	description	setting
S_a	average size of attributes	32 bytes
S_{GOid}	size of **GOid**	16 bytes
S_{LOid}	size of **LOid**	16 bytes
S_s	size of the object signatures	32 bytes
T_d	average disk access time	15 μs/bytes
T_{net}	average network transfer time	8 μs/bytes
T_c	average cpu processing time	0.5 μs/comparsion
N_{iso}	average number of isomeric objects for the same real world entity	2

Table 2: The database and query profile

parameters	description	default setting
for each global query		
N_{db}	number of component databases involved	3
N_c	number of global classes involved	1~4
for each involved global class k		
N_p^k	number of predicates on the class	0~3
R_{ps}^k	selectivity of the predicates on the class	$0.45^{(N_{pa}^{i,k})^{1/2}}$
for each constituent class of the involved global class k		
R_r^k	ratio of objects to be referenced	0.5~1
R_{isos}^k	ratio of objects having isomeric objects	$1\text{-}0.9^{(N_{db}-1)}$
for each constituent class of the involved global class k in database i		
$N_o^{i,k}$	number of objects	5000~6000
$N_{qa}^{i,k}$	number of attributes involved in the subquery	$\max\{N_{pa}^{i,k}, N_{ta}^{i,k}\}\sim (N_{pa}^{i,k}+N_{ta}^{i,k})$
$N_{pa}^{i,k}$	number of attributes involved in the local predicates	$0\sim N_p^k$
$N_{ta}^{i,k}$	number of attributes which are target attributes in the subquery	0~2
$R_{pps}^{i,k}$	selectivity of the local predicates on the class	$0.45^{(N_p^k)^{1/2}}$
$R_m^{i,k}$	ratio of objects which have missing data	1 if $(N_p^k - N_{pa}^{i,k}) > 0$ 0~0.2 otherwise
$R_{as}^{i,k}$	selectivity of the predicates on the assistant objects	$0.55^{(N_p^k-N_{pa}^{i,k})^{1/2}}$
$R_{ss}^{i,k}$	selectivity of the predicates on the signatures of the assistant objects	$0.6^{(N_p^k-N_{pa}^{i,k})^{1/2}}$

Table 3: The cost terms concerned in the processing

cost term	description
Disk Access Time	
T_{OA}	the access time for objects in the component database
T_{GA}	the access time for GOid mapping tables
T_{SA}	the access time for object signatures
Data Transfer Time	
T_{OT}	the transfer time for objects in the component database
T_{CAT}	the transfer time for LOids of the assistant objects to be checked
T_{SAT}	the transfer time for LOids of the satisfied assistant objects
CPU Processing Time	
T_P	the processing time for the predicate evaluation in phase **P**
T_{OO}	the processing time for the object outerjoin
T_{AP}	the processing time for the predicates on the assistant objects
T_{SP}	the processing time for the predicates on the object signatures
T_{CR}	the processing time for certifying the local maybe results and unsolved items
T_W	the waiting time for the processing

4.2.1 Total Execution Time

The total execution time of the algorithms is represented as the sum of the cost terms listed in the previous subsection.

The total execution time for algorithm **CA** is

$$T_{total}^{CA} = \sum_{i=1}^{N_{db}} (T_{OA}^{CA}(i) + T_{OT}^{CA}(i)) + T_{GA}^{CA} + T_{OO}^{CA} + T_P^{CA} \qquad (1).$$

For algorithm X of the localized approaches (i.e., algorithm **BL**, **PL**, **BLS**, or **PLS**), $T_{A\_total}^{X}(i)$ denotes the total execution time spent in checking and transferring the assistant objects for the unsolved items in database i.

$$T_{A\_total}^{X}(i) = \sum_{j=1, j\neq i}^{N_{db}} (T_{CAT}^{X}(i,j) + T_{AP}^{X}(i,j) + T_{SAT}^{X}(i,j)) \quad (2).$$

Then, the total execution time for algorithm X of the localized approaches without object signatures is:

$$T_{total}^{X} = \sum_{i=1}^{N_{db}} (T_{OA}^{X}(i) + T_P^{X}(i) + T_{GA}^{X}(i) + T_{OT}^{X}(i) + T_{A\_total}^{X}(i)) + T_{CR}^{X} \qquad (3),$$

where X is **BL** or **PL**.

The total execution time for algorithm X of the localized approaches with object signatures is: $T_{total}^{X} = \sum_{i=1}^{N_{db}} (T_{OA}^{X}(i) + T_P^{X}(i) + T_{GA}^{X}(i) + T_{OT}^{X}(i) + T_{SA}^{X}(i) + T_{SP}^{X}(i) + T_{A\_total}^{X}(i)) + T_{CR}^{X}$ (4),

where X is **BLS** or **PLS**.

4.2.2 Response Time

The response time is also estimated in terms of the cost terms.

The response time for algorithm **CA** is:

$$T_{resp}^{CA} = Max_{i\in[1..N_{db}]}\{T_{OA}^{CA}(i)+T_{OT}^{CA}(i)\}+T_{GA}^{CA}+T_{OO}^{CA}+T_{P}^{CA} \quad (5).$$

For algorithm *X* of localized approaches (i.e., algorithm **BL**, **PL**, **BLS**, or **PLS**), $T_{A\_resp}^{X}(i)$ denotes the response time spent in checking and transferring the assistant objects for the unsolved items in database *i*.

$$T_{A\_resp}^{X}(i) = Max_{j\in[1..N_{db}],j\neq i}\{T_{W}^{X}(i,j)+T_{CAT}^{X}(i,j)+T_{AP}^{X}(i,j)+T_{SAT}^{X}(i,j)\} \quad (6).$$

Thus, the response time for algorithm **BL** is:

$$T_{resp}^{BL} = Max_{i\in[1..N_{db}]}\{T_{OA}^{BL}(i)+T_{P}^{BL}(i)+T_{GA}^{BL}(i)+Max\{T_{OT}^{BL}(i),T_{A\_resp}^{BL}(i)\}\}+T_{CR}^{BL} \quad (7).$$

The response time for algorithm **PL** is:

$$T_{resp}^{PL} = Max_{i\in[1..N_{db}]}\{T_{OA}^{PL}(i)+T_{GA}^{PL}(i)+Max\{T_{P}^{PL}(i)+T_{OT}^{PL}(i),T_{A\_resp}^{PL}(i)\}\}+T_{CR}^{PL} \quad (8).$$

The response time for algorithm **BLS** is:

$$T_{resp}^{BLS} = Max_{i\in[1..N_{db}]}\{T_{OA}^{BLS}(i)+T_{P}^{BLS}(i)+T_{GA}^{BLS}(i)+T_{SA}^{BLS}(i)+T_{SP}^{BLS}(i) + Max\{T_{OT}^{BLS}(i),T_{A\_resp}^{BLS}(i)\}\}+T_{CR}^{BLS} \quad (9).$$

The response time for algorithm **PLS** is:

$$T_{resp}^{PLS} = Max_{i\in[1..N_{db}]}\{T_{OA}^{PLS}(i)+T_{GA}^{PLS}(i)+T_{SA}^{PLS}(i)+T_{SP}^{PLS}(i) + Max\{T_{P}^{PLS}(i)+T_{OT}^{PLS}(i),T_{A\_resp}^{PLS}(i)\}\}+T_{CR}^{PLS} \quad (10).$$

4.3 Simulation Results

4.3.1 Comparing Algorithms CA, BL and PL

- The average number of objects within each constituent class is adjusted:

  Figure 10(a) illustrates the total execution time for the three algorithms. In this case, algorithm **CA** requires more time than **BL** and **PL** do. The primary reason is that the local processing eliminates the objects which do not satisfy the local predicates. Therefore, it reduces the time for transferring and integrating data from component databases. Moreover, since both **BL** and **PL** are executed in component databases in parallel, their response time is much shorter than that of algorithm **CA**. Figure 10(b) displays the response time.

Alternately, the figures also present the performance comparison between algorithms **BL** and **PL**. Since **PL** checks the assistant objects before the local predicates are evaluated, more objects on the **GOid** mapping tables require checking. Moreover, more assistant objects are transferred and processed herein, than in **BL**. Although **PL** gains the benefit of parallel processing for phases **O** and **P**, the experimental result demonstrates that the benefit does not overcome its overhead. Therefore, in this case, **BL** performs better than **PL** does.

- The number of component databases is adjusted:

  Figure 10(c) presents the total execution time for the three algorithms. The ratio of objects, which have isomeric objects, increases as the number of component databases increases. For this reason, the number of assistant objects which need to be checked also increases. Moreover, the transfer time gets longer when more component databases transfer data simultaneously. Therefore, for algorithms **BL** and **PL**, the growing rate of the total execution time is higher than that of **CA** when the number of component databases increases. Figure 10(c) indicates the total execution time of **PL** even exceeds that of **CA**. However, the effect of parallel local processing in component databases makes the response time of algorithms **BL** and **PL** still shorter than that of **CA**. Figure 10(d) illustrates their response time.

- The selectivity of one local predicate is adjusted:

  Herein, $N_O^{i,k}$ is increased from 1000 to 2000. Fig. 10(e) and 10(f) demonstrate the total execution and response times, respectively. For algorithm **CA**, the varying selectivity does not influence the total execution and response times. However, these times for **BL** and **PL** increase as the selectivity increases. Also, for these two algorithms, the selectivity determines the number of objects which satisfy the local predicates of component databases. The number of eliminated objects decreases as the selectivity increases. This implies that the times for transferring and integrating data will increase. Furthermore, for algorithm **BL**, the selectivity also influences the number of assistant objects to be checked. Therefore, as the selectivity increases, the growing rate of the times for executing **BL** is higher than that of **PL**.

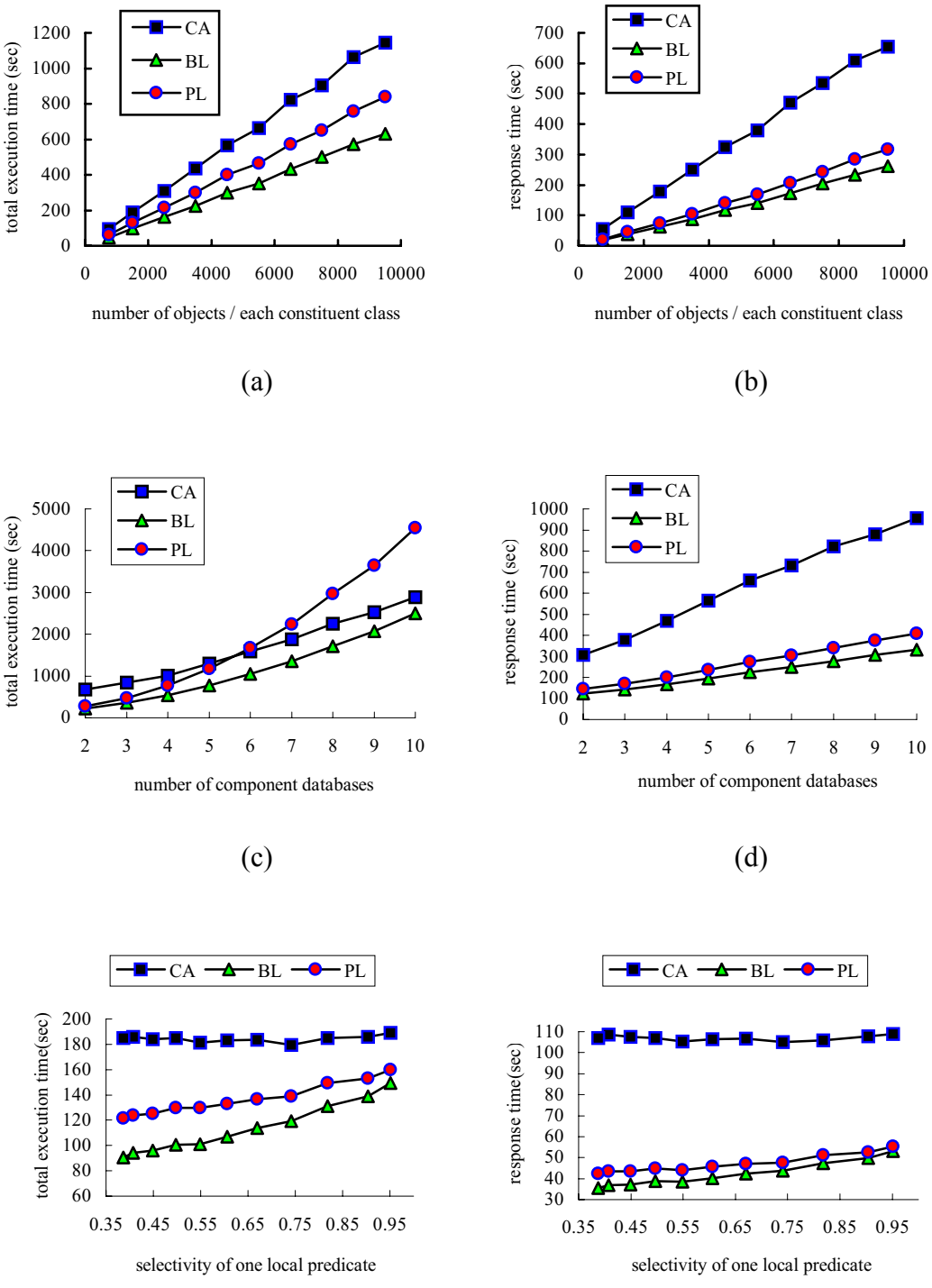

Figure 10: The performance comparisons of algorithms CA, BL, and PL.

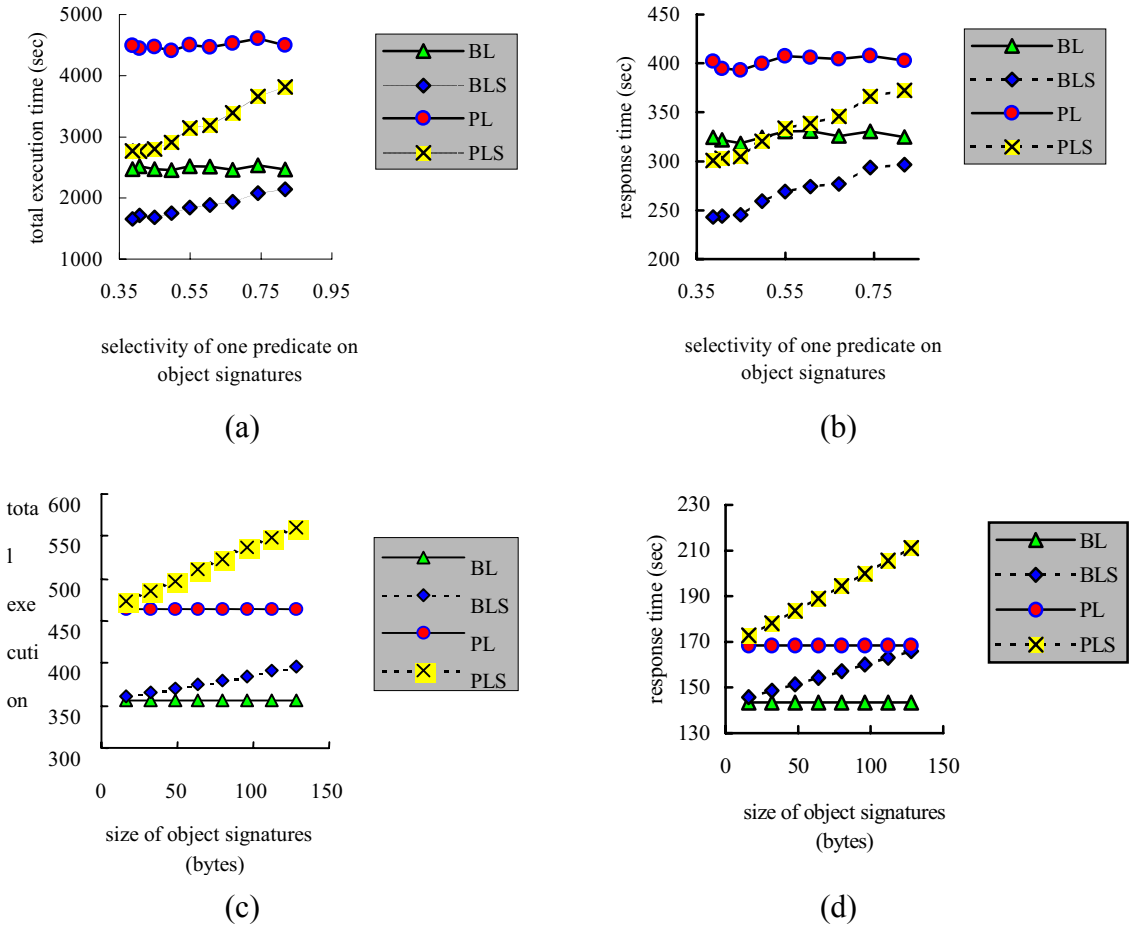

Figure 11: The performance comparisons of algorithms BL, BLS, PL, and PLS.

4.3.2 Comparing Approaches BL, BLS, PL and PLS

- The selectivity of one predicate on the object signatures is adjusted:

  Herein, N_{db} is set at 10. Figures 11(a) and 11(b) present the total execution and response times for these four algorithms, respectively. For both algorithms **BL** and **PL**, the varying of the selectivity does not influence the total execution and response times. However, algorithms **BLS** and **PLS** have better performance when the selectivity is lower. When the selectivity is less than 0.55, the response time of **PLS** is even shorter than that of **BL**. The times for **BLS** and **PLS** increase as the selectivity increases. However, in this situation, both the total execution and response times for **BLS** remain less than those of **BL**. The reason is that evaluating unsolved predicates on object signatures reduces the number of assistant objects to be checked in other component databases. Similarly, the times for **PLS** are less than that of **PL**.

- The size of object signature is adjusted:

  Herein, N_{db} is fixed at 3. Fig. 11(c) and 11(d) demonstrate the total execution time and response times, respectively. In contrast to the previous experiment, both the total execution and response times for **BLS** are longer than that of **BL**. As well, the times for **PLS** are longer than that of **PL**. Although applying object signatures reduces the number of assistant objects to be checked, additional processing overhead exists when evaluating the unsolved predicates on object signatures. The result indicates that the benefit is overcome by the overhead when the number of component databases is limited. Alternately, the figures demonstrate that the varying of the size does not influence the total execution and response times for both **BL** and **PL**. However, the times for **BLS** and **PLS** increase as the size of object signatures increases.

5 Partial Missing Attributes

5.1 Partial Missing Attributes

After schema integration is performed, for each global class, the semantically equivalent attributes in its constituent classes are integrated into an attribute A_m. Assume it is permitted that different values exist in the semantically equivalent attributes of the isomeric objects. Thus, A_m is a multi-valued attribute in the global class, and it is termed a *partial missing attribute* of the constituent classes.

In the example schemas illustrated in Fig. 1, suppose it is allowed that the location values of a department entity stored in **DB1** and **DB3** are different. Since the location values of the isomeric objects jointly represent the attribute *location* of a real world entity, this renders *location* a multi-valued attribute within the constructed global class **Department**. Moreover, *location* is a partial missing attribute of **Department@DB1** and **Department@DB3**.

The concept of partial missing attributes is similar to missing attributes. Suppose A_m is a missing attribute and A_{pm} is a partial missing attribute of a constituent class **C**. An object in **C** can not provide any information of A_m for its represented real world entity. However, the object has either partial or complete data of A_{pm} for the entity. In the next subsection, the proposed algorithms for processing global queries involving missing attributes are extended to process the queries that involve partial missing attributes.

5.2 Modified Approach for Partial Missing Attributes

When processing a global query, the centralized approach integrates data of the local root and local branch classes first. Then, the predicates involving partial missing attributes can be evaluated. Therefore, the centralized approach can be applied directly to process the global queries involving partial missing attributes. However, the localized approaches needs further modifications in certifying objects within the maybe results.

5.2.1 Basic Definitions

Definition 5. Given a global query $\boldsymbol{Q} = <\boldsymbol{S}, \boldsymbol{R}, \boldsymbol{P}>$.

The *partially unsolved predicates* of $\boldsymbol{Q}$ on component database $\boldsymbol{D_i}$ are defined to be:

$\boldsymbol{P_{pu}(Q,D_i)}$:={<Pj>|<Pj> in $\boldsymbol{P}$ where <Pj>:= $\mathbf{C}.A^j_1.A^j_2. \ldots .A^j_{v_j}$ <comparator> <constant-value>,

and $\exists$ k ($v_j \geq k \geq 1$ and A^j_k specified in<Pj> is a partial missing attribute of a constituent class in $\boldsymbol{D_i}$)}.

Notably, the path expression in a partially unsolved predicate is assumed to contain only one partial missing attribute.

Definition 6. Given a global query $\boldsymbol{Q} = <\boldsymbol{S}, \boldsymbol{R}, \boldsymbol{P}>$.

The *partially unsolved predicates* of $\boldsymbol{Q}$ on an object $\boldsymbol{o}$ in component database $\boldsymbol{D_i}$ are defined to be:

$\boldsymbol{P_{pu}(Q, o)}$:= { <Pj> | <Pj> in $\boldsymbol{P}$ and $\exists$ class $\mathbf{C}$ in database $\boldsymbol{D_i}$,

($\boldsymbol{o}$ in class $\mathbf{C}$ and <Pj> involving a partial missing attribute of $\mathbf{C}$)}.

The quantifiers and comparators which operate on the multi-valued attributes include the following [18]:

- quantifier: **each**, **exist**
- set-to-scalar comparator: **has-element**
- scalar-to-set comparator: **is-in**
- set-comparator: **has-subset**, **is-subset**, **is-equal**, **is-overlap**, **is-disjoint**

Each partially unsolved predicate involves one of these quantifiers or comparators.

Within the localized approaches, a global query $\boldsymbol{Q}$ is decomposed into local queries against their associated local root classes. In addition, the partially unsolved predicates are removed from the corresponding local query. Therefore, the maybe results are obtained after the local query is evaluated.

For each object $\boldsymbol{o_m}$ in the maybe results, it is *partially unsolved* if there exists at least one partially unsolved predicate on this object. Let object $\boldsymbol{o_{nc}}$ denote a nested complex attribute value of $\boldsymbol{o_m}$ that its partially missing attribute is involved in a partially unsolved predicate. The object $\boldsymbol{o_{nc}}$ is named a *partially unsolved item* of maybe result $\boldsymbol{o_m}$.

The isomeric objects of a partially unsolved object $\boldsymbol{o_{pu}}$, which have the same partial missing attribute A_{pm} with $\boldsymbol{o_{pu}}$, are called *partial assistant objects* of $\boldsymbol{o_{pu}}$ and denoted as $\boldsymbol{O_{pa}(o_{pu})}$. Object $\boldsymbol{o_{pu}}$ and its partial assistant objects all contain part of the data for attribute A_{pm} of a real world entity. Notably, an object *confirms* a partially unsolved predicate if the object and its partial assistant objects jointly satisfy the partially unsolved predicate. Finally, a partially unsolved object $\boldsymbol{o_{pu}}$ becomes *solved* if it confirms all its partially unsolved predicates. A maybe result $\boldsymbol{o_m}$ becomes a certain result if $\boldsymbol{o_m}$ and all its partially unsolved items are solved.

5.2.2 Sub-Predicate Construction

To confirm a partially unsolved predicate P_{pu} in a localized manner, its sub-predicate P'_{pu}, which can be evaluated in the component database, is constructed. The evaluation results of the sub-predicate on an object and its partial assistant objects are used to determine if the object confirms P_{pu}. For each component database $\boldsymbol{D_i}$, the method for constructing the associated sub-predicate $P'_{pu}(\boldsymbol{D_i})$ depends on whether the partial missing attribute A_{pm} in $\boldsymbol{D_i}$ is single-valued or multi-valued. According to the various quantifiers and comparators specified in P_{pu}, the $P'_{pu}(\boldsymbol{D_i})$ is constructed as follows:

<1> P_{pu} contains a quantifier.

- **each**

  P_{pu} is formulated as: **each** $P(A_{pm})$, where $P(A_{pm})$ denotes a predicate on A_{pm}.
  If A_{pm} is single-valued, $P'_{pu}(\boldsymbol{D_i}) := P(A_{pm})$.
  If A_{pm} is multi-valued, $P'_{pu}(\boldsymbol{D_i})$:= **each** $P(A_{pm})$.

- **exist**

  P_{pu} is formulated as: **exist** $P(A_{pm})$, where $P(A_{pm})$ denotes a predicate on A_{pm}.
  If A_{pm} is single-valued, $P'_{pu}(\boldsymbol{D_i}) := P(A_{pm})$.
  If A_{pm} is multi-valued, $P'_{pu}(\boldsymbol{D_i})$:= **exist** $P(A_{pm})$.

<2> P_{pu} contains a comparator.

Herein, a denotes an attribute value and A' represents a set of attribute values.

- **has-element**

P_{pu} is formulated as: A_{pm} **has-element** a.
If A_{pm} is single-valued, $P'_{pu}(\boldsymbol{D_i})$:= $A_{pm} = a$.
If A_{pm} is multi-valued, $P'_{pu}(\boldsymbol{D_i})$:= A_{pm} **has-element** a.

- **is-in**

P_{pu} is formulated as: a **is-in** A_{pm}.
If A_{pm} is single-valued, $P'_{pu}(\boldsymbol{D_i})$:= $A_{pm} = a$.
If A_{pm} is multi-valued, $P'_{pu}(\boldsymbol{D_i})$:= a **is-in** A_{pm}.

- **is-subset**

P_{pu} is formulated as: A_{pm} **is-subset** A'.
If A_{pm} is single-valued, $P'_{pu}(\boldsymbol{D_i})$:= A_{pm} **is-in** A'
If A_{pm} is multi-valued, $P'_{pu}(\boldsymbol{D_i})$:= A_{pm} **is-subset** A'.

- **is-overlap**

P_{pu} is formulated as: A_{pm} **is-overlap** A'
If A_{pm} is single-valued, $P'_{pu}(\boldsymbol{D_i})$:= A_{pm} **is-in** A'.
If A_{pm} is multi-valued, $P'_{pu}(\boldsymbol{D_i})$:= A_{pm} **is-overlap** A'.

- **is-disjoint**

P_{pu} is formulated as: A_{pm} **is-disjoint** A'.
If A_{pm} is single-valued, $P'_{pu}(\boldsymbol{D_i})$:= A_{pm} **not is-in** A'.
If A_{pm} is multi-valued, $P'_{pu}(\boldsymbol{D_i})$:= A_{pm} **is-disjoint** A'.

- **is-equal**

P_{pu} is formulated as: A_{pm} **is-equal** A'.
If A_{pm} is single-valued, $P'_{pu}(\boldsymbol{D_i})$:= A_{pm} **is-in** A'.
If A_{pm} is multi-valued, $P'_{pu}(\boldsymbol{D_i})$:= A_{pm} **is-subset** A'.

- **has-subset**

P_{pu} is formulated as: A_{pm} **has-subset** A'.
No suitable sub-predicate can be constructed regardless if attribute A_{pm} is single-valued or multi-valued. Further processing is required and described later.

5.2.3 Partial Unsolved Predicate Confirmation

Two rules are provided to judge the confirmation of a partially unsolved object o_{pu}.

[OR-TYPE Confirmation Rule]

A partially unsolved object o_{pu} confirms a partially unsolved predicate P_{pu} if
(o_{pu} in component database D_i and o_{pu} satisfying $P'_{pu}(D_i)$) **or**
($\exists\ o' \in O_{pa}(o)$ (o' in component database D_j and o' satisfying $P'_{pu}(D_j)$).

Otherwise, object o_{pu} is eliminated from the query result.

[AND-TYPE Confirmation Rule]

A partially unsolved object o_{pu} confirms a partially unsolved predicate P_{pu} if
(o_{pu} in component database D_i and o_{pu} satisfying $P'_{pu}(D_i)$) **and**
($\forall\ o' \in O_{pa}(o)$ (o' in component database D_j and o' satisfying $P'_{pu}(D_j)$).

Otherwise, object o_{pu} is eliminated from the query result.

For each partially unsolved predicate P_{pu} on a partially unsolved object o_{pu}, the processing to confirm P_{pu} is divided into three cases according to the quantifier or comparator in P_{pu}.

Case 1: If **exist**, **is-in**, **has-element**, or **is-overlap** is specified in P_{pu}, the **OR-TYPE** confirmation rule is applied to confirm P_{pu}. Initially, the associated sub-predicate is evaluated on o_{pu}. If the sub-predicate is satisfied, o_{pu} confirms P_{pu}. Otherwise, the partial assistant objects of o_{pu} are checked by their associated sub-predicates. If one of the partial assistant objects satisfies the sub-predicate, P_{pu} is confirmed by o_{pu}. Object o_{pu} is eliminated if all of its partial assistant objects violate the sub-predicates.

Case 2: If **each**, **is-subset**, or **is-disjoint** is specified in P_{pu}, the **AND-TYPE** confirmation rule is applied to confirm P_{pu}. First, the associated sub-predicate is evaluated on o_{pu}. If the sub-predicate is not satisfied, o_{pu} is eliminated. Otherwise, the partial assistant objects of o_{pu} are checked by their associated sub-predicates. If all of the partial assistant objects satisfy the sub-predicates, o_{pu} confirms P_{pu}. Again, if any of its partial assistant objects do not satisfy the sub-predicate, o_{pu} is eliminated.

Case 3: The final case is applied when comparator **is-equal** or **has-subset** is specified in P_{pu}.

Case 3-1: P_{pu} is formulated as: A_{pm} **is-equal** A'.

The **AND-TYPE** confirmation rule is applied to confirm P_{pu}, which is similar to the process described in Case 2. However, the satisfaction of the sub-predicates on o_{pu} and its

partial assistant objects is not sufficient to confirm P_{pu}. Rather, the A_{pm} values in $\boldsymbol{o_{pu}}$ and its partial assistant objects have to be collected such that the set comparison, *is-equal*, with A' can be evaluated. To reduce the cost for transferring the A_{pm} values, a bitmap called *partial assistant bitmap* is applied to represent the correspondences between the values of A_{pm} and A' for an object. A partial assistant bitmap is constructed for an object $\boldsymbol{o}$ if $\boldsymbol{o}$ satisfies the sub-predicate. For each element in A', there is an associated bit in the partial assistant bitmap. If the ith element in A' is contained in $\boldsymbol{o}.A_{pm,}$ the ith bit in the bitmap is set to 1. Otherwise, the bit is set to 0. Finally, if $\boldsymbol{o_{pu}}$ and its partial assistant objects all satisfy their associated sub-predicates, their partial assistant bitmaps are then combined via **OR** operator. If all bits in the resultant bitmap are 1s, $\boldsymbol{o_{pu}}$ confirms P_{pu}. Otherwise, $\boldsymbol{o_{pu}}$ is eliminated.

Case 3-2: P_{pu} is formulated as: A_{pm} **has-subset** A'.

No sub-predicate is constructed to be evaluated on $\boldsymbol{o_{pu}}$ and its partial assistant objects. However, to denote the correspondences between the values of A_{pm} and $A',$ their partial assistant bitmaps are constructed. As in Case 3-1, these partial assistant bitmaps are combined via **OR** operation. If all bits in the resultant bitmap are 1s, P_{pu} is confirmed.

When a global class has more than two constituent classes, it is possible that an attribute A in the global class is a missing attribute of some constituent classes and a partial missing attribute of others. Assume that a predicate in a global query involves A. After local predicate evaluation, the unsolved objects are obtained from the constituent classes with missing attribute A. To certify an unsolved object, its assistant objects must be checked by the associated unsolved predicate P_u. Because A is a partial missing attribute of the assistant objects, the checking process is to evaluate if these assistant objects confirm the partially unsolved predicate P_u.

6 Conclusion

Missing data occurs within integrated multiple object databases due to the missing attribute conflict as well as the existence of null values. A set of algorithms is provided to process the predicates of global queries with missing data. The concept of object isomerism is applied to integrate local data in query processing. Therefore, it is possible for the local maybe results, which are produced due to missing data, to become certain results. First, the centralized

approach integrates data forwarded from component databases and then processes the predicates. Alternately, the localized approaches process the local predicates, and the local results are then integrated and certified. Therefore, the localized approaches achieve inter-site parallelism. Furthermore, object signatures are applied in the algorithms to reduce the data transfer.

According to the simulation results, the total execution and response times of the proposed algorithms are compared and discussed. Among the algorithms **CA**, **BL**, and **PL**, the performance of **BL** is superior. The number of component databases is a major factor that influences the total execution time of **PL**. If selectivity of the local predicates increases, the performance of algorithms **BL** and **PL** declines. This effects on **BL** more than it does on **PL**. Alternately, when the number of component databases increases, the performance of the localized approaches improves more via object signatures. However, when the size of the object signatures increases, the performance of algorithms with signature declines. Furthermore, the multi-valued attributes in the global schema, with values derived from attribute values in different component databases, result in the partial missing attributes. The proposed approaches are also extended to process global queries that involve partial missing attributes.

In this paper, the predicates in global queries are assumed to be in conjunctive form. In the future, the proposed algorithms will be modified to process global queries containing predicates in disjunctive form. The explicit join may also appear in a global query. This infers that the global query contains more than on range class. The efficient processing of global queries with explicit joins requires subsequent research. Furthermore, in a predicate of a global query, the path expression may contain more than one partial missing attribute. The strategy for processing such a global query also requires further research.

Reference

[1] R. Ahmed, P. DeSmedt, W. Du, W. Kent, M. Ketabchi, W. Litwin, A. Rafii, M.C. Shan, "Pegasus Heterogeneous Multidatabase System", *IEEE Computer*, Dec. 1991.

[2] C. Batini, M. Lenzerini, and S.B. Navathe, "A Comparative Analysis of Methodologies for Database Schema Integration," *ACM Computing Surveys*, vol. 18, no. 4, pp. 323-364, 1986.

[3] E. Bertino, M. Negri, G. Pelagatti, and L. Spampinato, "Applications of Object-Oriented Technology to the Integration of Heterogeneous Database Systems," *Distributed and Parallel Databases*, vol. 2, no. 4, pp. 343-370, 1994.

[4] A.L.P. Chen, "Outerjoin Optimization in Multidatabase Systems," *Proc. IEEE Int'l Symposium on Databases in Parallel and Distributed Systems*, 1990.

[5] A.L.P Chen, "A Localized Approach to Distributed Query Processing," *Proc. Int'l Conf. on Extending Data Base Technology*, 1990.

[6] A.L.P. Chen, J.L. Koh, T.C.T. Kuo, and C.C. Liu, "Schema Integration and Query Processing for Multiple Object Databases," *Integrated Computer-Aided Eng.: Special Issue on Multidatabase and Interoperable Systems*, vol. 2, no. 1, *Wiley and Sons, 1995*.

[7] A.L.P. Chen, P.S.M. Tsai, and J.L. Koh, "Identifying Object Isomerism in Multiple Databases," *Distributed and Parallel Databases*, vol. 4, no. 2, Kluwer Academic Publishers, 1996.

[8] M.-S. Chen and P.S. Yu, "A Graph Theoretical Approach to Determine a Join Reducer Sequence in Distributed Query Processing," *IEEE Trans. Knowledge and Data Eng.*, vol. 6, no. 1, 1994.

[9] B. Czejdo, M. Rusinkiewicz and D.W. Embley, "An Approach to Schema Integration and Query Formulation in Federated Database Systems," *Proc. 3rd Int'l Conf. on Data Eng.*, pp.477-484, 1987.

[10] E.F. Codd, "Extending the Database Relational Model to Capture More Meaning," *ACM Trans. Database Systems*, vol. 4, pp. 397-434, 1979.

[11] E.F. Codd, "Missing Information (Applicable and Inapplicable) in Relational Databases," *ACM SIGMOD RECORD*, vol. 15, pp.53-78, 1986.

[12] S.M. Deen, R.R. Amin, and M.C. Taylor, "Data Integration in Distributed Databases," *IEEE Trans. Software Eng.*, vol. SE-13, no. 7, pp.860-864, 1987.

[13] L.G. DeMichiel, "Resolving Database Incompatibility: an Approach to Performing Relational Operations over Mismatched Domains," *IEEE Trans. Knowledge and Data Eng.*, vol. 1, no. 4, pp.485-493, 1989.

[14] W. Gotthard, P.C. Lockemann, and A. Neufeld, "System-Guilded View Integration for Object-Oriented Databases," *IEEE Trans. Knowledge and Data Eng.*, vol. 4, no. 1, pp.1-22, 1992.

[15] S. Hayne and S. Ram, "Multi-User View Integration System (MUVIS) : An Expert System for View Integration," *Proc. 6th Int'l Conf. on Data Eng.*, pp. 402-409, 1990.

[16] B.P. Jeng, D. Woelk, W. Kim, and W.L. Lee, "Query Processing in Distributed ORION," *MCC Technique Report Number: ACA-ST-035-89*, (1989) pp. 1-26.

[17] W. Kent, "Solving Domain Mismatch and Schema Mismatch Problems with an Object-Oriented Database Programming Language," *Proc. 16th Very Large Data Bases Conf.*, pp.147-160, 1991.

[18] W. Kim, "Introduction to Object-Oriented Databases," *The MIT Press*.

[19] W. Kim, I. Choi, S. Gala, and M. Scheevel, "On Resolving Schematic Heterogeneity in Multidatabase Systems," *Distributed and Parallel Databases*, vol. 1, no. 3, pp.251-279, 1993.

[20] J.L. Koh and A.L.P. Chen, "Integration of Heterogeneous Object Schemas," *Lecture Notes in Computer Sciences : Entity-Relationship Approach-ER '93*, vol. 823, Springer-Verlang: Berlin, pp. 297-314, 1993.

[21] J.L. Koh and A.L.P. Chen, "A Mapping Strategy for Querying Multiple Object Databases with a Global Object Schema," *Proc. 5th Int'l Workshop on Research Issues in Data Eng.*, pp.50-57, 1995.

[22] J.L. Koh, "Integration and Query Processing for Object and Multimedia Databases, " Ph.D. Dissertation of Department of Computer Science, National Tsing Hua University, 1998.

[23] A. Motro, "Superviews : Virtual integration of multiple databases," *IEEE Trans. on Software Eng.*, vol. 13, no. 7, pp.785-798, 1987.

[24] M.P. Reddy, B.E. Prasad, P.G. Reddy, and A. Gupta, "A Methodology for Integration of Heterogeneous Databases," *IEEE Trans. on Knowledge and Data Eng.*, vol. 6, no. 6, pp.920-933, 1994.

[25] E.A. Rundensteiner, "Multiview : A Methodology for Supporting Multiple Views in Object-Oriented Databases," *Proc. 8th Very Large Data Bases Conf.*, pp.187-198, 1992.

[26] A. Sheth, J. Larson, A. Cornelio, and S. Navathe, "A Tool for Integrating Conceptual Schemas and User Views," *Proc. 4th Int'l Conf. on Data Eng.*, pp.176-183, 1988.

[27] A. Sheth and V. Kashyap, "So Far (Schematically) yet So Near (Semantically)," *Proc. the IFIP Conf. on Semantics of Interoperable Database Systems*, Lorne, Australia, Nov.

1993.

[28] S. Spaccapietra and C. Parent, "View Integration: A Step Forward in Solving Structural Conflicts," *IEEE Trans. on Knowledge and Data Eng.*, vol. 6, no. 2, pp.258-274, 1994.

[29] F.S.C. Tseng, A.L.P. Chen, and W.P. Yang, "Answering Heterogeneous Database Queries with Degrees of Uncertainty," *Distributed and Parallel Databases*, vol. 1, no. 3, pp. 281-302, 1993.

[30] UniSQL, Inc., "UniSQL/X Database System User's Manual," Release 22.0, Austin, Texas, 1993.

[31] C. Wang, A.L.P Chen, and S.-C. Shyu, "A Parallel Execution Method for Minimizing Distributed Query Response Time," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, no. 3, pp.325-333, 1992.

[32] H.S. Yong, S. Lee, and H.-J. Kim, "Applying Signatures for Forward Traversal Query Processing in Object-Oriented Database Systems," *Proc. 10th Int'l Conf. on Data Eng.*, pp.518-525, 1994.

Appendix

- T_{OA}

$$T_{OA}(i) = \sum_{k=1}^{N_c} [\, T_d \times N1 \times (N_{qa}^{i,k} \times S_a + S_{LOid})] \qquad 1 \le i \le N_{db}$$

where $N1 = N_o^{i,k}$ for algorithm **CA**,

$= N_o^{i,k} \times R_r^k$ for algorithms **BL**, **PL**, **BLS**, and **PLS**.

- T_{OT}

Let $N_{ro}^{i,k} = N_o^{i,k} \times R_r^k$,

$R_{sf}^{i,k} = R_m^{i,k} \times R_{iso}^k \times (1 - R_{ss}^{i,k})$

$F_d = \sqrt{N_{db}}$

$$T_{OT}^{CA}(i) = \sum_{k=1}^{N_c} [\, T_{net} \times N_o^{i,k} \times (N_{qa}^{i,k} \times S_a + S_{LOid}) \times F_d] \quad 1 \le i \le N_{db}$$

For algorithms **BL**, **PL**, **BLS**, and **PLS**:

$$T_{OT}(i) = \sum_{k=1}^{N_c} [\, T_{net} \times N2 \times (N_{qa}^{i,k} \times S_a + S_{GOid}) \times F_d] \quad 1 \le i \le N_{db}$$

where $N2 = N_{ro}^{i,k} \times \prod_{j=1}^{N_c} R_{pps}^{i,j}$ for algorithms **BL** and **PL**,

$= N_{ro}^{i,k} \times \prod_{j=1}^{N_c} R_{pps}^{i,j} \times (1 - R_{sf}^{i,k})$ for algorithms **BLS** and **PLS**.

- T_{GA}

$$T_{GA}^{CA} = \sum_{i=1}^{N_{db}} \sum_{k=1}^{N_c} (T_d \times N_o^{i,k} \times S_{GOid})$$

For algorithms **BL**, **PL**, **BLS**, and **PLS**:

Let $N_{so}^{i,k} = N_{ro}^{i,k} \times \prod_{j=1}^{N_c} R_{pps}^{i,j}$

$$T_{GA}(i) = \sum_{k=1}^{N_c} (T_d \times N3 \times S_{GOid}) \qquad 1 \le i \le N_{db}$$

where $N3 = N_{so}^{i,k} \times R_m^{i,k}$ for algorithms **BL** and **BLS**,

$= N_{ro}^{i,k} \times R_m^{i,k}$ for algorithms **PL** and **PLS**.

- T_P

$$T_P^{CA} = \sum_{i=1}^{N_{db}} \sum_{k=1}^{N_c} [\, T_c \times N_{ro}^{i,k} \times (1 - R_{iso}^k / N_{iso}) \times \prod_{j=1}^{k} R_{ps}^j \times N_p^k]$$

For algorithms **BL**, **PL**, **BLS**, and **PLS**:

$$T_P(i) = \sum_{k=1}^{N_c} (T_c \times N4 \times N_{pa}^{i,k}) \qquad 1 \le i \le N_{db}$$

$N4 = N_{ro}^{i,k} \times \prod_{j=1}^{k} R_{pps}^{i,j}$ for algorithms **BL, PL** and **BLS**,

$= N_{ro}^{i,k} \times \prod_{j=1}^{k} R_{pps}^{i,j} \times (1 - R_{sf}^{i,k})$ for algorithm **PLS**.

- T_{CAT}, T_{AP}, T_{SAT}

For algorithms **BL**, **PL**, **BLS**, and **PLS**:

Let $R_{ao}^{i,k} = R_{m}^{i,k} \times R_{iso}^{k}$

$$T_{CAT}(i,j) = \sum_{k=1}^{N_c} [T_{net} \times N5 \times (S_{LOid} + S_{GOid}) \times F_d] / (N_{db} - 1)$$

$$T_{AP}(i,j) = \sum_{k=1}^{N_c} [(T_d \times S_a + T_c) \times (N_p^k - N_{pa}^{i,k}) \times N5]$$

$$T_{SAT}(i,j) = \sum_{k=1}^{N_c} [T_{net} \times N5 \times R_{as}^{i,k} \times S_{GOid} \times F_d] / (N_{db} - 1)$$

$$1 \le i \le N_{db}, \quad 1 \le j \le N_{db}, \quad i \ne j$$

where $N5 = N_{so}^{i,k} \times R_{ao}^{i,k}$ for algorithms **BL**,

$= N_{ro}^{i,k} \times R_{ao}^{i,k}$ for algorithms **PL**,

$= N_{so}^{i,k} \times R_{ao}^{i,k} \times R_{ss}^{i,k}$ for algorithms **BLS**,

$= N_{ro}^{i,k} \times R_{ao}^{i,k} \times R_{ss}^{i,k}$ for algorithms **PLS**.

- T_{OO}

$$T_{OO}^{CA} = \sum_{k=1}^{N_c} \sum_{i=1}^{N_{db}} \sum_{j=i+1}^{N_{db}} (T_c \times N_o^{i,k} \times N_o^{j,k})$$

- T_{CR}

For algorithms **BL**, **PL**, **BLS**, and **PLS**:

$$T_{CR} = \sum_{k=1}^{N_c} \sum_{i=1}^{N_{db}} (T_c \times N_{so}^{i,k} \times N6 \times R_{ao}^{i,k} \times R_{as}^{i,k})$$

$N6 = N_{so}^{i,k}$ for algorithms **BL**,

$= N_{ro}^{i,k}$ for algorithms **PL**,

$= (1 - R_{sf}^{i,k}) \times N_{so}^{i,k} \times R_{ss}^{i,k}$ for algorithms **BLS**,

$= (1 - R_{sf}^{i,k}) \times N_{ro}^{i,k} \times R_{ss}^{i,k}$ for algorithms **PLS**.

- T_{SA} and T_{SP}

For algorithms **BLS**, and **PLS**:

$$T_{SA}(i) = \sum_{k=1}^{N_c} (T_d \times N7 \times S_s)$$

$$T_{SP}(i) = \sum_{k=1}^{N_c} [T_c \times N7 \times (N_p^k - N_{pa}^{i,k})] \qquad 1 \le i \le N_{db}$$

$N7 = N_{so}^{i,k} \times R_{ao}^{i,k}$ for algorithms **BLS**,

$= N_{ro}^{i,k} \times R_{ao}^{i,k}$ for algorithms **PLS**.

- T_W

For algorithms **BL**, **PL**, **BLS**, and **PLS**:

$$T_W(i,j) = (\sum_{k=1}^{N_{db}} T_{AP}(j,k) + T_P(j)) \times 1/2$$

Biography of authors:

Jia-Ling Koh received the B.S. degree in computer science from National Chiao-Tung University, Taiwan, R.O.C. in 1991, the M.S. and the Ph.D. degrees in computer science from the National Tsing Hua University, Taiwan in 1993 and 1997, respectively. She is currently an assistant professor in the Department of Computer Education, National Taiwan Normal University. Her current research interests include multimedia information retrieval and data mining.

Arbee L.P. Chen received the B.S. degree in computer science from National Chiao-Tung University, Taiwan, R.O.C. in 1977, and the Ph.D. degree in computer engineering from the University of Southern California in 1984.

He joined National Tsing Hua University, Taiwan, as a National Science Council (NSC) sponsored Visiting Specialist in August 1990, and became a Professor in the Department of Computer Science in 1991. He was a Member of Technical Staff at Bell Communications Research, New Jersey, from 1987 to 1990, an Adjunct Associate Professor in the Department of Electrical Engineering and Computer Science, Polytechnic University, New York, and a Research Scientist at Unisys, California, from 1985 to 1986. His current research interests include multimedia databases, data mining and mobile computing.

Dr. Chen has organized (and served as a Program Co-Chair) 1995 IEEE Data Engineering Conference and 1999 International Conference on Database Systems for Advanced Applications (DASFAA) in Taiwan. He is a recipient of the NSC Distinguished Research Award.