

# A Mobile Agent Approach for Global Database Constraint Checking

Praveen Madiraju and Rajshekhar Sunderraman

Department of computer science

Georgia State University

Atlanta GA 30303

{cscpnmx, raj}@cs.gsu.edu

## ABSTRACT

Integrity constraints are valuable tools for enforcing consistency of data in a database. Global integrity constraints ensure integrity and consistency of data spanning multiple databases. In this paper, we propose a general framework of a mobile agent based approach for checking global constraints. An insert/update/delete initiated on single site, say  $S_1$  may cause the violation of a global constraint. The check for such violation involves accessing related data from multiple sites, say  $S_2...S_n$ . *Constraint Checker* on site  $S_1$  generates sub constraint checks on sites  $S_2...S_n$  and sends multiple remote agents,  $rmagent_2...rmagent_n$  to sites  $S_2...S_n$  respectively for checking the sub constraints. These remote agents carry with them data processing code to be executed at remote sites. *Constraint Checker* gathers results from the remote agents and decides if any constraint is violated. The constraint checking mechanism is much faster as the sub constraint checks are executed in parallel.

## Categories and Subject Descriptors H.2.4

[**Database Management**]: Systems – *distributed databases, query Processing*. I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *intelligent agents*.

## Keywords

Multi-Database Systems, Global Constraints, Mobile Agents

## 1. INTRODUCTION

A multidatabase system consists of a collection of autonomous component database systems. Distribution of data across multiple sites is a clear trend in many emerging internet applications.

One major advantage of data distribution is each site can process its own data with some degree of autonomy and user's can be provided with a single global view of the data. A service

that integrates data from multiple sources is called a data integration service. Integration approaches are typically based on multidatabase or federated database systems [3, 10, 17]. Some of the major issues in multidatabase systems are: Global querying and Global constraint checking.

Global querying refers to the problem of information retrieval from heterogeneous and distributed sources. In the conventional approach a user issues a global query. The global query issued from one site is parsed and sub queries are sent to different sites hosting the data. The results from these sub queries are collected and finally summarized answer is returned to the user.

In a multidatabase system, when multiple database systems interoperate, there is a very large likelihood of global constraints to be violated. Global constraints specify and enforce that a particular database state is consistent and ensures integrity of data across multiple databases. A data change on single site might cause a global constraint to be violated. Due to the continuous change in data and frequent global constraint violations, we need an approach that would efficiently and speedily check for constraint violations. This is the basic motivation for us to employ mobile agent based technique for checking global database constraint.

Mobile agents can be considered as an incremental evolution of the earlier idea of "process migration". A mobile agent is an autonomous, active program that can move both data and functionality (code) to multiple places within a distributed system. The state of the running program is saved and transported to the new host, allowing the program to continue execution from where it left off before migration [5]. Mobile agents require two components for their successful execution. The first component is the agent itself. The second component being the place where in an agent can execute. This is often referred to as the software agent framework. It provides services and primitives that help in the use, implementation and execution of systems deploying mobile agents. Some of the popular examples are: IBM's Aglets [8], Mitsubishi Electric ITA's Concordia [13] and Object Space's Voyager [7].

Mobile agents have been recently recognized as an efficient means for distributed information retrieval [4]. Recent research has considered using mobile agents for global querying, but none of the literature so far has looked in to the aspect of using mobile agents for global constraint checking.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC '04, March 14-17, 2004, Nicosia, Cyprus.

Copyright 2004 ACM 1-58113-812-1/03/2004...\$5.00.

Figure 1 shows an outline of our approach. Using the database description of remote database objects, Global Metadatabase is constructed. Global constraints to be enforced are also stored in the Global Metadatabase. We provide the design of the Constraint Checker module, which accepts an insert/update/delete request from a user and considers one constraint at a time from Global Metadatabase and decides if any constraint is violated.

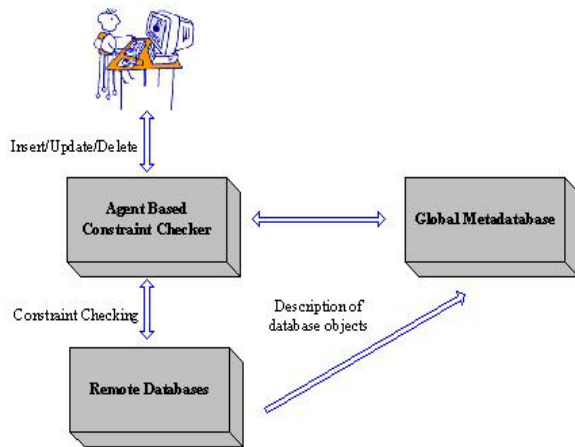


Figure 1: Global Constraint Checking Approach

The rest of this paper is organised as follows: In Section 2, we give an example of a multidatabase system. Through out the paper, we refer to this database and illustrate our constraint checking method on some example constraints. In section 3, we describe the overall system architecture of constraint checking. In section 4, we give the internal architecture of the constraint checker module and illustrate the overall procedure of constraint checking. We then, describe related work in section 5. Finally in section 6, we summarise and offer our conclusions.

## 2. MOTIVATING SCENARIO

Consider a typical health care multidatabase management system. It is a very natural scenario to have patient's information distributed across multiple sites.

**Site  $S_1$ :** Patient information is stored here. A *PATIENT* relation with attributes *name* and type of *healthplan* is recorded in the schema: *PATIENT* (*name*, *healthplan*).

**Site  $S_2$ :** Health insurance companies store patient's claim information here. A *CLAIM* relation with attributes *name* (patient name), *amount* of claim, date of claim and *type* of claim is recorded in the schema:

*CLAIM* (*name*, *amount*, *claimdate*, *type*)

**Site  $S_3$ :** Doctor's office maintains patient's name, doctor treating the patient and disease for which the patient is being diagnosed here. A *DOCTOR* relation with attributes *name* (patient name), *doctorname* and *disease* is recorded in the schema:

*DOCTOR* (*name*, *doctorname*, *disease*).

Figure 2 below shows sample data in the healthcare multidatabase system.

On this multidatabase, we can define global constraints. We limit our constraints only to the class of semantic integrity constraints. We need to use a notation convenient for describing constraints. For this purpose we use the approach of [2]. A constraint is a query whose result is either 0 or 1([2] calls it "panic"). If the query produces 0 on the multidatabase *D*, then *D* is said to satisfy the constraint, or the constraint is violated on *D*. Let us consider a constraint such as: A patient with health plan 'B' diagnosed with 'smallpox' may not claim more than 20,000 dollars. This is can be expressed as follows:

PATIENT : S1		DOCTOR : S3		
name	healthplan	name	doctorname	disease
king	B	king	mike	smallpox
blake	C	king	tony	flu
clark	A	king	mike	skin allergy

CLAIM : S2			
name	amount	claimdate	type
king	10000	6/10/2003	emergency
blake	30000	5/8/2003	prescription
king	5000	6/25/2003	hospital

Figure 2: Sample data for the health care multi-database

$PanicC_1$  :- PATIENT(name, 'B'),  
CLAIM(name, amount, \_, \_),  
DOCTOR(name, \_, 'smallpox'), amount > 20000.

For convenience we will refer  $PanicC_1$  as just  $C_1$ .

## 3. SYSTEM ARCHITECTURE

The system architecture is shown in Figure 3. A constraint checker module resides on each of the data sources. This module is responsible for interfacing with the Global Metadatabase. In figure 3, an update statement;  $U_i$  is issued on site  $S_1$ . It modifies/updates some of the database objects. Constraint Checker on  $S_1$  sends out mobile agent on to the Global Metadatabase.

The Global Metadatabase is a repository of site and domain information. Site information gives description of sites where data sources reside. Domain information gives metadata description of database objects of all data sources and global constraints, say  $C_1, \dots, C_n$ .

The mobile agent at the Global Metadatabase is equipped with the knowledge of database objects being modified and also data processing code. The mobile agent computes the list of global constraints being affected by  $U_i$ , say  $C_1 \dots C_m$ . The mobile agent returns this list to the constraint checker.

Constraint checker takes as input one global constraint at a time,  $C_1$ . For each global constraint, sub constraints corresponding to remote sites are generated. Mobile agents  $rmagent_2, rmagent_3, rmagent_4$  are spawned to individual sites  $S_2, S_3, S_4$ . Constraint Checker gathers results from these mobile agents and makes a decision if a global constraint is violated. This process is repeated for all remaining constraints  $C_2 \dots C_m$ .

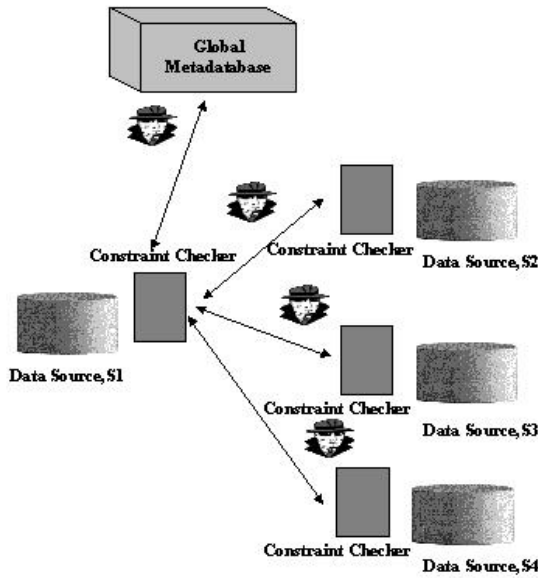


Figure 3: Constraint checking architecture

#### 4. CONSTRAINT CHECKING PROCEDURE

The internal architecture of the proposed constraint checking methodology is shown in Figure 4.

The constraint checker consists of the following modules:

- **Update Parser:** is responsible for parsing an update statement input by the user and identifying the database objects involved in the update statement.
- **Metadatabase Extractor:** extracts all the global constraints being affected by the update statement.
- **Constraint Planner:** devises an effective plan for generating sub constraints on remote sites.
- **Constraint Optimizer:** optimises the ordering of sub constraints for efficient constraint checking.
- **Constraint Executor:** is responsible for generating, spawning mobile agents and gathering results obtained from the agents.

The overall procedure of constraint checking is explained in the following eight steps. Consider the health care multidatabase management system introduced in Section 2. The initial database state is shown in Figure 5.

Consider two global constraints defined on this multidatabase. Constraint  $C_1$  has been defined in Section 2.

```

C1 :- PATIENT(name, 'B'),
      CLAIM(name, amount, _, _),
      DOCTOR(name, _, 'smallpox'), amount > 20000.
C2 :- PATIENT(name, 'B'),
      CLAIM(name, _, _, 'emergency').
  
```

Constraint  $C_1$  states that a patient with healthplan 'B' diagnosed with 'smallpox' may not claim more than 20000 dollars. Constraint  $C_2$  states that a patient with healthplan 'B' may not file a claim of type 'emergency'.

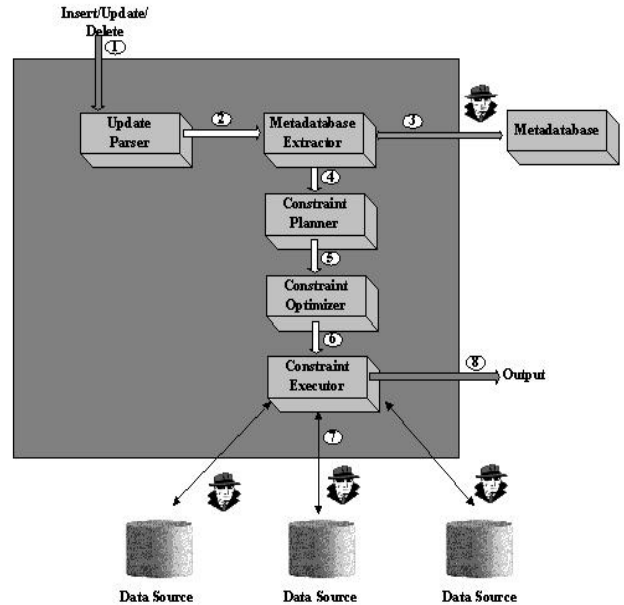


Figure 4: Constraint checker internal architecture

#### STEP 1

The user issues an insert/update/delete statement on to his local data source. Figure 5 shows the initial multi-database state. For example, user issues an update statement on Site  $S_2$ :

```

U1 = insert into CLAIM values ('john', 25000,
                                '06/10/2003', 'emergency');
  
```

Figure 6 shows the modified database state after the insert statement is issued. Our goal is to check if this updated database state still satisfies all the integrity constraints.

#### STEP 2 (Update Parser)

The Update Parser parses the given update statement and identifies the database objects being modified. The output from this step is a *database object list*

```

database object list = {CLAIM (name='john',
                              amount=25000, claimdate='06/10/2003',
                              type='emergency')}
  
```

PATIENT : S1		DOCTOR : S3		
name	healthplan	name	doctorname	disease
john	B	john	mike	smallpox

CLAIM : S2			
name	amount	claimdate	type

Figure 5: Initial multi-database state

CLAIM : S2			
name	amount	claimdate	type
john	25000	6/10/2003	emergency

PATIENT : S1		DOCTOR : S3		
name	healthplan	name	doctorname	disease
john	B	john	mike	smallpox

Figure 6: Multi-database state after update

### STEP 3 (Meta-database Extractor)

The Metadatabase Extractor takes as input *database object list* and this is fed to a mobile agent. The mobile agent is sent to the Global Metadatabase. The mobile agent returns with it the Constraint Data Source Table (CDST). For the running example, the CDST is shown in Figure 7.

$$CDST(C_i) = \langle C_i, list(S_j) \rangle$$

-  $C_i$  is the constraint identifier

-  $list(S_j)$  is the list of data sources being affected by  $C_i$

CDST	
$C_i$	$list(S_j)$
$C_1$	$(S_1, S_2, S_3)$
$C_2$	$(S_1, S_2)$

Figure 7: The constraint data source table

### STEP 4

The Metadatabase Extractor sends the CDST to the Constraint Planner

### STEP 5 (Constraint Planner)

The Constraint Planner generates a set of sub constraints for each  $C_i$ . The Constraint Planning Table (CPT) is generated.

$$CPT(C_i) =$$

$$\langle C_i, list(C_{ij}, S_j), and-list(C_i) \rangle$$

$C_i$  is the constraint identifier

$C_{ij}$  is the sub constraint corresponding to  $C_i$  and the particular Site  $S_j$

$and-list(C_i)$  is the list of sub constraints forming a conjunction of sub constraints

CPT		
$C_i$	$list(C_{ij}, S_j)$	$and-list(C_i)$
$C_1$	$(C_{11}, S_1) (C_{12}, S_2) (C_{13}, S_3)$	$(C_{11}, C_{12}, C_{13})$
$C_2$	$(C_{21}, S_1) (C_{22}, S_2)$	$(C_{21}, C_{22})$

Figure 8. The constraint planning table

$C_{11} =$  select 1 from dual where exists ( select \* from PATIENT where name = 'john' and healthplan = 'B');

$C_{12} =$  select 1 from dual where exists ( select \* from CLAIM where name = 'john' and amount > 20000);

$C_{13} =$  select 1 from dual where exists ( select \* from DOCTOR where name = 'john' and disease = 'smallpox');

$C_{21} =$  select 1 from dual where exists ( select \* from PATIENT where name = 'john' and healthplan = 'B');

$C_{22} =$  select 1 from dual where exists ( select \* from CLAIM

where name = 'john' and type = 'emergency');

The idea is to have a mobile agent spawned to each  $S_j$  corresponding to each  $C_{ij}$ . The use of mobile agents enhances efficiency as the sub constraints are executed in parallel. The CPT is sent to Constraint Optimizer.

### STEP 6 (Constraint Optimizer)

The Constraint Optimizer optimises the constraint checking process. The constraint optimizer generates the Constraint Optimized Table (COT). Any optimisations that would increase the efficiency of the constraint checking process are carried out here. In the above example,  $C_2$  involves accessing two sites  $S_1$  and  $S_2$ , where as  $C_1$  involves accessing sites  $S_1, S_2$  and  $S_3$ . Constraint Optimizer orders the execution of the Constraints and also sub constraints.

COT		
$C_i$	$list(C_{ij}, S_j)$	$and-list(C_i)$
$C_2$	$(C_{21}, S_1) (C_{22}, S_2)$	$(C_{22}, C_{21})$
$C_1$	$(C_{11}, S_1) (C_{12}, S_2) (C_{13}, S_3)$	$(C_{12}, C_{11}, C_{13})$

Figure 9: The constraint optimized table

Observe that if  $C_2$  is violated, we will not bother to check  $C_1$  and since the constraint checking is much faster doing  $C_2$  first and then  $C_1$ , we have gained efficiency. Also, the *and-list* is ordered for each sub constraint. In this example, since  $C_1$  is initiated on  $S_2$ , we have ordered the *and-list*( $C_2$ ) in the order of  $C_{22}$  and  $C_{21}$ . The idea is to first check for local sub constraints (local to  $S_2$ ) and then any remote sub constraints. The reason is if one of  $C_{22}$  or  $C_{21}$  returns "false" or "no rows returned", then constraint  $C_2$  is satisfied. In a similar way COT ( $C_1$ ) is also ordered.

### STEP 7 (Constraint Executor)

The Constraint Executor reads the COT and spawns mobile agent for each  $C_i$ . Constraint Executor gathers the results from each of the mobile agents.

$C_{22} = 1(\text{true})$  and  $C_{21} = 1(\text{true})$ .  $and-list(C_2) = C_{22} \wedge C_{21}$ . Hence  $and-list(C_2) = 1(\text{true})$ . Constraint  $C_2$  is violated.

In this case, Constraint Executor does not have to check for  $C_1$  as we already know  $C_2$  is violated.

### STEP 8

The results are sent to the user.

## 5. RELATED WORK

Much of the research concerning integrity constraint checking has been done in the area of relational database systems. Grefen and Apers ([12]) provide an excellent survey. Widom and Ceri ([11]) give an exhaustive survey of protocols for integrity constraint checking in federated database systems. Gupta and Widom ([1]) give approaches for constraint checking in distributed databases. Some of the approaches for distributed databases and federated databases can be easily applied to multi-databases with some minor changes. Ceri and Widom ([15]) propose inter-database triggers for maintaining equality constraints between heterogeneous databases. Widom and Ceri ([9]) mention research on active databases and constraints. Our

work differs from the above by expressing a global constraint as set of sub constraints and ultimately a query is carried over to the remote databases. To our knowledge, we have not seen any research work on using mobile agents for global constraint checking.

ACQUIRE [16], an agent based complex query and information retrieval engine considers an agent-based approach for information retrieval from distributed data sources. ACQUIRE translates each user query into a set of sub queries by employing a combination of planning and traditional database query optimisation techniques. For each sub query ACQUIRE then sends a corresponding mobile agent which does the computation work and retrieves the result. When all the agents have returned, ACQUIRE filters and merges the retrieved data and the results are displayed to the user. MOMIS [14] gives a framework for information integration that deals with the integration and query of multiple, heterogeneous information sources. MOMIS (Mediator environment for multiple information sources) uses agent-based approach, where in they have multiple agents doing different kinds of tasks. Our idea is also similar to the above, however they are using mobile agents for global querying and we are using mobile agents for global constraint checking.

## 6. CONCLUSION

In this paper we have presented a general framework of a mobile agent based approach for constraint checking across multiple databases. Our architecture is completely distributed, where in we have a constraint checker module running on each database. We have presented the design of the constraint checker module. The constraint checker module has the ability to check if a given update statement violates any global constraints. Our main contribution is the design of a faster and speedy constraint checking mechanism. The reason is sub constraint checks on remote databases are executed in parallel by mobile agents.

Works are in progress to develop and improve our constraint checking mechanism. We plan to give an algorithmic description of the constraint planner module and also performance cost model for the constraint optimizer module. We plan to design a first prototype of the constraint checker, which will implement the ideas discussed in the paper. For this purpose, we need a software agent framework, which would support creation, activation and management of agents. We have kaariboga agent system [6] in mind. It is 100% java compatible and open source. Although the system is not as robust and secure as [7,8,13], It is suitable for developing agents that are of experimental value. As a future work, the ideas presented in this paper open new avenues for constraint checking on mobile multidatabases. Mobility of databases introduces new challenges for constraint checking, as the mobile agent executing a constraint at remote database may not return results because of disconnection or network failure. Since mobile agent framework is inherently asynchronous, our approach is very much extendable to mobile multidatabases.

## REFERENCES

[1] A. Gupta and J. Widom. Local Verification of Global Integrity Constraints in Distributed Databases. Proceedings

of the ACM SIGMOD International Conference on Management of Data, pages 49-58, May 1993.

[2] A. Gupta, Y. Sagiv, J.D. Ullman, and J. Widom. Constraint Checking with Partial Information. Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pages 45-55, May 1994.

[3] A. Sheth, J. Larson: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Comp. Surveys* 22:3 (1990), 183-236.

[4] Brewington, B., Gray, R., Moizumi, K., Kotz, D., Cybenko, G., and Rus, D. (1999) "Mobile Agents in Distributed Information Retrieval", *Intelligent Information Agents*, Springer-Verlag, 1999.

[5] C. G. Harrison, D.M. Chessm, A.kershenbaum. *Mobile Agents: Are they a good idea?* Research Report, IBM Research Division, 1994.

[6] Dirk Struve, kaariboga. <http://www.projectory.de/kaariboga/index.html>.

[7] G.Glass. Overview of Voyager: ObjectSpace's Product Family for State-of-the-Art Distributed Computing. Technical report, ObjectSpace, 1999.

[8] G. Karjoth, D. Lange, and M. Oshima A Security Model for Aglets. *IEEE Internet Computing*, Vol. 1, No. 4, 1997.

[9] J. Widom and S. Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, San Francisco, California, 1996.

[10] M. Bright, A. Hurson, S. Pakzad: A Taxonomy and Current Issues in Multidatabase Systems. *IEEE Computer* 25:3, Pages 50-59.

[11] P. Grefen and J. Widom. Protocols for Integrity Constraint Checking in Federated Databases. *International Journal of Distributed and Parallel Databases*, 5(4): 327-355, 1997.

[12] P. Grefen and P. Apers, Integrity Control in Relational Database Systems - An Overview, *Journal of Data and Knowledge Engineering*, 10 (2), 187-223, 1993

[13] R. Koblick. Concordia. *Communications of the ACM*, 42(3): 96-99, March 1999.

[14] S. Bergamaschi, G. Cabri, F. Guerra, L. Leonardi, M. Vincini, F. Zambonelli. Supporting Information Integration with Autonomous Agents. *CIA 2001*: 88-99.

[15] S. Ceri and J. Widom. Managing Semantic Heterogeneity with Production Rules and Persistent Queues. Proceedings of the Nineteenth International Conference on Very Large Data Bases, pages 108-119, Dublin, Ireland, August 1993.

[16] S. K. Das, K. Shuster, C. Wu. ACQUIRE: agent-based complex query and information retrieval engine. *AAMAS 2002*: 631-638.

[17] W. Kim: *Modern Database Systems: The Object Model, Interoperability, and Beyond*, ACM Press, New York, 1995.