The architecture and implementation of the Vega Information Grid

Xiaolin Li*
Institute of Computing Technology
Chinese Academy of Sciences
6, Kexueyuan South Road
Zhongguancun, Haidian District
Beijing, 100080, China
E-mail: lx1@ict.ac.cn
E-mail: char@ict.ac.cn
*Corresponding author

Wenlong Huang
Graduate School of the Chinese Academy of Sciences
Beijing, 100080, China
E-mail: huangwenlong@software.ict.ac.cn

Li Zha
Institute of Computing Technology
Chinese Academy of Sciences
Beijing, 100080, China
E-mail: char@ict.ac.cn

Abstract: There are many common challenges to integrating and sharing information in distributed environment; for example, heterogeneous structure of data sources, dynamic changing of source location and schema, sharing information across multiple autonomous administrative domains, and personalised requirements. Although it is possible to build a specialised solution to meet any scenario requirements, the solution, which is very suitable to the specialised scenario, may not be available to other scenarios. The Vega Information Grid (VIG) abstracts the commonality of requirements and addresses these issues by information virtualisation. It is a toolkit, which provides functionalities suitable for different scenarios, enables users to acquire and deploy information cost-effectively in a virtual grid platform, and to benefit greatly from the decrease in the full life cycle time. This paper introduces its information virtualisation model, decoupled architecture, and virtual database engine to integrate decentralised data sources.

Keywords: information grid; information virtualisation; decoupled architecture; virtual database engine; information sharing.

Reference to this paper should be made as follows: Li, X., Huang, W. and Zha, L. (2007) ‘The architecture and implementation of the Vega Information Grid’, Int. J. Web and Grid Services, Vol. 3, No. 4, pp.462–479.
The architecture and implementation of Vega Information Grid

Biographical notes: Xiaolin Li is an Assistant Professor in the Institute of Computing Technology, Chinese Academy of Sciences. He earned his PhD in 2005. His research areas of interest include grid computing and data integration in decentralised and autonomous environment.

Wenlong Huang is a doctoral candidate of the Graduate School of the Chinese Academy of Sciences. His research focuses on data integration in the grid environment.

Li Zha is an Associate Professor in the Institute of Computing Technology, Chinese Academy of Sciences. He obtained his PhD in 2003. His research is focused on large scale distributed resource management, which includes naming, organisation and policy mechanisms. His interest also includes other classic issues in distributed computing and grid computing field.

1 Introduction

Nowadays, an enterprise’s IT group is being forced to find new and cost-effective means to integrate and share information from multiple information sources in a wide-open network environment in order to react to changes in competition, market dynamics and regulatory mandates (BEA, 2005; The Lowell Database, 2005). There are many critical problems in this endeavour, such as the following:

- Heterogeneity
  In most enterprises, there are a vast number of different information sources, including databases, text files, images, videos, directories, and so on. Unfortunately, the number of formats in which data is stored is as varied as the types of information stores themselves. In order for an application to deal with the variety of information sources as well as their associated means of exposing information, an application developer is forced to code specifically for each source.

- Dynamical changes
  As each information source evolves and new information sources become available, the application must be built again or modified to adapt specifically to the changing of source location, data format, authorisation and exposure mechanism of the information source. Although SQL provides a more general purpose mechanism through which to obtain data stored in databases, its adoption as the means to obtain information from other information sources, such as file and service, never came about.

- Sharing across multiple domains
  Sharing information is often required dynamically between business co-partners; user identity authentication and user authorisation are necessary to keep information secure. Enterprises now consume information in many different formats and from many different data sources in a pattern of autonomous control. As a result, it is difficult to store in advance all forms of information in a single form of information repository, as data warehouse did.
Personalisation

With their requirements changing, users would not be bound to a consolidated information source and application. They need to share and control the full life cycle of their own experiences, including deploying resource, obtaining information, managing resource and authorisation and personalised profile configuration.

The problems above contain some of commonalities of requirements of information systems. In order to meet these commonalities, the Vega Information Grid (VIG) team focuses research on these issues and has developed a toolkit for building an information-sharing application scenario.

This paper is organised as follows: Section 2 introduces some related works. Section 3 describes an information virtualisation model, which mainly aims to resolve the heterogeneity and dynamically changing data source. Section 4 presents the key concepts from the perspective of utilisation pattern and the decoupled architecture of the VIG platform, which enables all those components to run as an integrated system. Section 5 discusses the functionalities and query optimisation mechanisms of virtual database engine to integrate decentralised data sources. An implementation and evaluation is presented in Section 6. Section 7 presents the conclusion.

2 Related work

In order to solve the integration challenges faced within many enterprises, commercial and research efforts have provided some comprehensive solutions spanning web services (Mayerl et al., 2005; Halley and Bashioum, 2005) and information integration technologies (BEA and IBM Corporation, 2005).

OGSA-DAI (Jackson et al., 2004) focuses on the issues surrounding the design of an efficient, generic, service-based interface to databases. Most previous work on interfaces has been language-dependent and not service-based (e.g., JDBC). Higher-level grid data management systems, such as SRB (Rajasekar et al., 2002) or Chimera (Foster et al., 2002), could use OGSA-DAI either to access structured data resources, or to provide access to their own metadata.

The European Data Grid has developed Spitfire (Bell et al., 2002), a web service interface to relational databases for metadata management, which allows a client to query a relational database over GSI-enabled HTTP(S). An XML-based protocol is used to represent the query, and its result. IBM Corporation provides solutions to enable distributed enterprise integration with WebSphere and the back-end DB2 Information Integrator (II). SkyQuery (Malik et al., 2003) applies the classical wrapper-mediator architecture in a service-based setting, supports querying autonomous astronomy archives over collections of web services. It deploys WSs at each database store for handling metadata, performing queries, and cross-matching partial results.

In a database community, there are many research works, which are called data integration, to construct virtual data view from multiple data sources. In theory, data integration can be divided into two approaches: Local as View (LAV) and Global as View (GAV) (Lenzerini, 2002). There are many projects on data integration that can be
classified into the two categories. The examples of LAV system include Information Manifold (Kirk et al., 1995) and the system presented in Manolescu et al. (2001). The examples of GAV system include TSIMMIS (Garcia-Molina et al., 1997), Garlic (Carey et al., 1995), COIN (Goh et al., 1999) and MOMIS (Beneventano et al., 2000).

These solutions and technologies focus on the efficiency (Goel et al., 2005), availability (Belalem and Slimani, 2007), and fault tolerance of data source integration from the perspective of business requirements, while changing agilely for the personalisation and autonomous control from the perspective of utilisation pattern are often in second place.

From the point of view of data integration, the VIG provides the functions of constructing virtual data view and data query execution by the GAV integration approach. Because many data views are needed by different user requirements, we employ GAV but not LAV as the integration way.

3 Information virtualisation

For such issues in information source as location transparency in accessing, heterogeneous structure and schema, integrating multiple sources into a virtual view, different data formats and uniform access interface – the layered virtualisation technique and approach is often adopted to figure out these issues.

From the point of view of user utilisation, SQL provides a friendly interactive query interface although its adoption as the means cannot obtain information from sources other than databases. What is the most important is that relational databases are the most popular storage of information sources, and majority of information system users are familiar with the utilisation of SQL query (Codd, 1970; Pal et al., 2005).

So, the VIG virtualises source by means of a three-layered architecture model (Figure 1) (Wei and Zhiwei, 2003). The three layers, namely physical relation, virtual relation and effective relation, are represented with relation schema and mapping to the lower-layer relation schema. Physical relation stands for the concrete physical sources in the network, which have been described with a relation-based schema, such as databases, files and directories, and services. Virtual relation is the virtualised information view, which is built based on a physical relation or other virtual relations according to the business requirements of enterprise or user community (Xiaolin et al., 2003); while effective relation denotes the personalised user requirements, which are built based on a virtualised relation by the user self.

The distinguishing feature of the information virtualisation model is that each layer would not impose heterogeneity of metadata and interface on a higher layer, since each is represented with relation schema. Therefore, the application code developed based on any layer can run smoothly on other layers where the requested relation schema is available. The following subsections discuss the functionalities and features of each layer relation schema.
3.1 Physical relation

Based on the Service-Oriented Architecture, one or multiple data sources are encapsulated as a service, which serves as the container of sources. The data sources of the service are exposed to the grid user by means of accessing the service. As a result, the community user can browse and access data. The single data source of the service, such as database table, view, file or directory, is virtualised and represented as a physical relation of the VIG by describing with a relation schema. A data service can serve as the access interface for a pool of physical relations.

The key features of virtualisation in physical relation layer include:

- Data source providers can deploy or destroy autonomously any data source to the pool of a data service without any modification to the service.
- Providers can decide which table, data sets or view of data source to be shared with other grid users, since relation schema is very convenient to describe the requirement.
- Heterogeneity of data source and location transparency can be addressed transparently to a higher layer by developing and deploying wrapper for the data source in the layer.

However, for the higher layer, the physical relation schema is exposed instead of the data service, as in the case of OGSA-DAI. So the data service information, which should be implied in the description of physical relation, is necessary to access data source. The mechanism makes it easier for users to find the desired data source.
As a whole, the metadata schema of a physical relation includes two parts: the relation data schema and its data service.

Undeniably, there is an important issue that need to be resolved, i.e., how to virtualise other kinds of data source as relations, such as services or software components.

3.2 Virtual relation

Virtual relation is built dynamically according to the community business requirements and schema, it is an integrated relation based on a physical relation or other virtual relations. In contrast, the definition of physical relation is from the point of view of the resource provider.

There are two kinds of virtual relation: basic virtual relation and composed virtual relation. Correspondingly, two kinds of schema mapping operations exist, namely, reference to virtual relations and mapping to physical relation. Basic virtual relation can only map to no more than one physical relation. Composed virtual relation is built by referencing other shared virtual relation sets, including other composed virtual relations, whenever loop reference among relations should be escaped. Consequently, data sources can be integrated and fused by constructing composed virtual relations based on basic virtual relation sets, each element of which in turn maps to a shared physical relation in the network.

Distinguishing the two map operations is valuable for the user who is building a business model. Mapping concerns only schema matching without any awareness of business requirements. But reference definition should be aware of the business model, in which the sharing relationship and data flow of computing among virtual relations based on the schema mapping are interpreted. For example, the sum() functionality of SQL denotes computing of data flow of source based on the schema definition.

The key features of virtualisation in the virtual relation layer include:

- Users can dynamically build business relation data model based on the relation schema. Business data schema or data format can be adapted to business requirements by changing reference to schema without any modification to the application code.
- Integration of data sources can be achieved based on a homogenous virtual relation schema. Integrator concerns only business data model without any awareness of the heterogeneous characteristics of data source.
- The builder of virtual relation can decide on the sharing of virtual relation among grid virtual communities. Therefore, specifications of the business data model can be shared resulting in a decreasing deployment of time and cost of the application scenario.

Generally speaking, the metadata schema of describing virtual relation includes its relation schema, mapping or reference, sharing and authorisation.

3.3 Effective relation

Since virtual relation is the abstraction of the data model of business requirements, all community users can access and acquire a consistent information view by sharing virtual relations. However, considering personal requirements, individual users may need
a personalised information view even when accessing the same data model. Effective relation is the personalised relation built by individual users based on a virtual relation. And it can be shared by other users, too.

For example, virtual relation $V$ provides an integrated information view of all department employees of an enterprise; however, the manager of research and development department may only care for staff employed in the department. An effective relation built based on $V$ can meet the manager’s requirement.

The key features of virtualisation in effective relation layer include:

- Effective relation can be deployed only once and can be accessed many times.
- Effective relation enables an individual user to consider only personalised requirements without needing to construct the data model.
- Effective relations which were created by an individual user consist of the user’s personal workspace, but it is not needed to allocate storage for the individual user’s personal data. Security of personal data can be guaranteed by access control autonomously.

The metadata schema of describing effective relation consists of its relation schema, mapping to virtual relation schema and sharing configuration.

### 3.4 Consistency issue

As the schema space of relational database, virtual schema space in the virtualisation model also supports constraints such as primary key, foreign key, null and unique check, and meets 3-NF constraint, too. For example, if a virtual relation is the union view of multiple physical relations, which in turn map to multiple data sources, the primary key constraint of an attribute should keep each value of the attribute’s union result set from multiple data sources both not null and unique.

An important issue of the virtualisation model deals with how to keep the consistency of changing relation schema, when changing or deleting an attribute of the virtual schema, which does not require an overhaul of the entire schema space in pursuit of low overhead and autonomous control feature. It also deals with when and how to provide user information when exception happens in processing mapping or reference.

If attribute $A_1$ of virtual relation $V_1$ references to attribute $A_2$ of virtual relation $V_2$, and the $A_2$ is deleted, the $A_1$ would remain unchanged. During the process of parsing schema reference, the reference objective of $A_1$ should not be available. If $A_1$ has been set primary key or unique constraints, an exception should be thrown to the user because the situation cannot meet the requirement definition of the schema. In other cases, the default value of $A_1$ or an empty value should be returned for the attribute $A_1$ without any exception.

### 4 Decoupled architecture

We defined and implemented a decoupled architecture of VIG (Figure 2). From the perspective of organisation and utilisation pattern of the full life cycle of the system, the architecture employs three abstractions: community, channel and Micro-Session.
A community is a realisation of the administrative domain. As a persistent construct, it maintains two sets of members, the members of users and the members of effective and virtual relations, which the community members have created. In addition, it maintains context and access control policies shared by the members.

For the end user, the architecture defines channel as the abstraction of user interaction with the Vega Information system. Channel is a persistent construct, too. User’s requests, such as querying data, writing data to the data source, or accessing a web page, are abstracted as one or several of interaction events with a channel handle on runtime. An end user can create and deploy channels in his/her individual virtual space and can share them with the other members within the same community or those of other communities.

Micro-Session is a dynamic construct on runtime, whose original intention is to improve performance and portability by sharing and supporting the context of which developer did have to write codes to take care before. A Micro-Session maintains the life cycle of the context on which the request is processed. The context includes three parts. The subject part corresponds to user identity and certificate, allocated role. The object part maintains the lists of runtime instances of resources (e.g., effective or virtual relation) which have been bound to the Micro-Session for executing the application logic of the request. The status part corresponds to the temporary data structure and status of processing the request of the session.

When catching a request which is represented by a channel handle and corresponding event in a session cycle, the system performs in advance a parsing of channel handle and event, then checks whether the corresponding Micro-Session identified by assembling session id and channel handle has been created in the session container, which serves as the runtime container of Micro-Session. If not, a new Micro-Session should be created to support the life cycle of the processing, and a unique identifier should be allocated to it. Otherwise, the created Micro-Session is scheduled again by the session container to support processing of the current request.
The Micro-Session puts the request to the virtual database engine, which executes the request immediately based on the current context of the Micro-Session. The engine consists of five important components. The query parser parses the request to obtain which effective relation is queried, and checks the validity of the request’s input parameters, then maps effective relation to virtual relation. If the virtual relation has not been bound to the Micro-Session, the query parser would bind it to create an instance of the virtual relation descriptor by accessing virtual relation schema from cache, then do reference parsing to acquire referenced subvirtual relation set and bind them recursively. A virtual relation descriptor is a dynamic combo of schema and context of current Micro-Session (e.g., user identity and active role). It should be destroyed when the Micro-Session dies or expires. Subsequently, based on these instances of descriptor, the integrator decomposes the query to generate subqueries that get executed on physical data sources, and stores these middle results in cache database. Finally, the query integrator executes the composed query based on these results, and returns the final result.

For querying and writing data to data source, access control and constraints check are necessary. Access control can provide fine-grained control on relation cell of data source based on the RBAC model. A constraints check can keep data consistent and avoid redundancy of data in a virtual multiple data source environment.

An execution of request may access one or multiple physical relations. The data service proxy is responsible for binding and invoking corresponding data service deployed in a Tomcat/Axis host environment. The identifier or address of data service is available in the physical relation schema.

5 Virtual database engine

The virtual database engine is the most important implementation of the virtualisation model. Its main functionality is to integrate decentralised and autonomous multiple sources in a dynamic environment by the means of querying on data sources. Its key components are the query parser and query integrator. In traditional distributed environment, the physical data sources usually belong to an autonomous domain. Thus, the integrator has enough privileges to know the metadata about data sources. The integrator can gain the knowledge about the capacity of different data sources, network connection speed between data sources and the loads on the physical data source servers. In a grid environment, it is common to share data among autonomous domains. For security reasons, the integrator does not have enough privilege to know much about the data sources. So, the policy of data query execution and optimisation in a grid environment is different from that in traditional distributed environment. In VIG, the virtual database engine emphasises on the query optimisation mechanism based on autonomous control on data sources in a decentralised environment.

From the point of view of functionalities, the virtual database engine is responsible for multiple query parsing, query optimisation and query execution, parallel query, query dataset caching, batch query, virtual relation lock control and management.
5.1 Query parsing

A request of user and application is a query on a virtual relation. The query needs to be parsed to queries on other virtual relations or physical relations. The parsing process is recursive until queries can be executed on physical data sources. Query parsing is based on the reference and mapping of virtual relation schema and user request’s parameters. The bind operation loads virtual relation schema to memory, with user request’s parameters, to construct a resource descriptor instance. Consequently, query parsing bases on the resource descriptor object without the need to load metadata from disk storages. The result of query parsing is the runtime query graph, which denotes the topology of query object instances. Each query of the graph can be executed on physical data sources or middle results in cache database to construct query datasets.

5.2 Query optimisation

Query optimisation is necessary to improve the performance of multiple queries on distributed data sources. In a grid environment, the conditions where users access virtual data view concurrently and distributed queries run across many physical data sources may cause the VIG to process huge amounts of data. So the performance requirement is more important than the requirement on individual data resources requested by individual users. Four important optimisation mechanisms are adopted: query reference, parallel query, query caching and query paging.

Firstly, executing query graph can be divided into one or multiple iterative query processes, each of them doing subtasks as follows:

• bind physical data sources addresses to the query
• pose the query to relative physical data sources and get the result
• compose the results returned and send the results back to upper query or user.

When optimising the query execution plan, it is necessary to optimise each of the processes or minimise the number of processes. The optimisation is based on some assumptions:

• The provider of data source has no knowledge about other data sources.
• Many users are interested only in a fraction of the data set. In other words, massive user queries embody local principle.
• The virtual database engine usually does not have enough privileges to obtain enough metadata about physical data sources.
• The cost of time to transfer data across network is greater than the cost of time to process data.

To improve the performance of query execution, the engine takes several measures:

• Query reference can map upper query execution to subqueries or physical data sources by parsing the schema reference of virtual relation. So, query reference can push down data aggregation operation. It includes two aspects, pushing down user query condition, which can reduce the amounts of data transferred across the
network, and pushing down query processing to subvirtual relations or physical data sources, which can minimise the number of processes of the query graph. The engine can optimise the operation according to operation property. For example, when a simple select operation is applied to the virtual data table formed by three physical tables, two of which are from the same physical data source. The engine can first carry out the select operation on union table formed at the physical data source. Next, the engine will transfer the result and the table data from the other physical data source, then executes select operation on them.

- Employ multithread programming model to serve the query concurrently. It can improve the throughput of VIG and the utilisation rate of CPU. Based on the query graph, the subquery graph among which there are no query-dependent relationship of data accessing should be parallel launched. The engine can execute independent subqueries simultaneously and keep their synchronisation. On the other hand, the virtual database engine also supports multithread to parallel the query on single network data sources.

- The VIG provides memory caches to metadata and mediator database to store data. This can reduce the latency to fetch the metadata and data. Aggregation operations can cause a large amount of data transferred across network to be processed. Temporary results can be cached to decrease the overhead of queries and improve performance by sharing the results of queries executed before. Meanwhile, the data validity in cache can be assured by checking the timestamp of temporary table when binding virtual relation. Under an environment where the metadata of physical data sources and virtual data change infrequently, or multiple users access the same data sources simultaneously, the time of fetching data and metadata can be greatly reduced.

- Provide query paging technique to give user a chance to choose the number of returned results. Some users just want to browse part of total records, so the paging query result technique can greatly reduce the latency in these occasions.

These measures, unlike the query plan optimisation measures of data integration community, which depends on the detail knowledge of physical data sources, depend on the virtual data view relationship from the point of view of business process to improve the query performance effectively.

Other functionalities include batch query and lock control and management. The virtual database engine supports application to read and write a large amount of dataset by way of batching to save system resources. In addition, because multiple queries can reuse query result by sharing the same temporary table, it is necessary for the engine to lock all virtual relations when it begins to serve a request and releases them when finishing the execution of query graph in order to avoid other threads to drop the temporary table or delete data from it at the same time.

With these functions, the virtual database engine can provide high performance for querying data belonging to the autonomous domain and residing at different physical data sources.
6 Implementation and evaluation

The virtualisation model and architecture presented above have been implemented and deployed respectively in a test-bed and production mode platform in the Institute of Computing Technology, CAS (ICT, CAS) (Xiaolin et al., 2003). Some of the departments, project teams, or virtual laboratories in ICT compose the communities of the production platform. Each community is a subinformation grid with multiple users allocated one or multiple role by the community administrator. It needs to integrate and share isolated applications, databases and files to provide users with a single system image and agility of changing requirements of information fusion.

Since information exchanges often occur across multiple communities, interconnecting communities as an integrated information grid is needed, in which accessing transparently to other community resource is allowed if the resource provider has shared the resource to the community.

Some evaluation works have been done on the test-bed server, which is equipped with two 1800 MHz AMD Athlon CPUs, 1000 MB memory, and running Tomcat4.0.6 container with Axis engine1.0, and MySQL l4.0.26 database.

Figure 3 shows the evaluation result of concurrent performance. The primary concern is that the multiple layers of virtualisation may increase the overload of processing a request. The result shows that Micro-Session and instance of virtual relation descriptor can maintain and share context and current status, most of which would be initialised when processing request for the first time during a session. Consequently, this decreases evenly the overload if request for the same data source was sent continually, e.g., browsing a series of information pages, in contrast to the situation without the support of these runtime construct. Correspondingly, Figure 4 shows the relationship of throughput with the number of concurrent requests.

Figure 3 Mean time of response (second) increases evenly with an increase of concurrent user requests.
Figure 4  Relationship of throughput (requests/second) with concurrent user requests

When considering the performance and efficiency of the virtual database engine in the VIG, we again deploy a TCP-H benchmark test-bed to test the performance of the virtual database engine.

We use a subset of TPC-H data for our test and distribute these data across two physical data sources. The two physical data sources are SQL Server 2000 and Oracle9i database Server. The TPC-H tables we used are PART, PARTSUPP, SUPPLIER, NATION, CUSTOMER, ORDERS. Figure 5 shows the architecture of the test-bed. We distribute these tables across the two data sources to simulate the scenario that the data of two databases is integrated and shared.

Figure 5  Topology of TPC-H test-bed

The primary key values of these tables are distributed in round-robin fashion. The largest table, ORDERS, contains about 1500 thousand of rows. The PARTSUPP table contains about 800 thousand of rows, while the CUSTOMER and PART tables contain
The architecture and implementation of Vega Information Grid

150 thousand and 20 thousand of rows, respectively. The smallest tables are SUPPLIER and NATION, with about 10,000 and 25 rows, respectively, which are deployed equally on both database sources.

The Microsoft SQL Server 2000, Oracle database and VIG platform reside in a desktop Windows workstation equipped with an Intel P4 3.4 GHz CPU, 1024 MB memory and Windows XP Operation System. JMeter runs on a PC equipped with an Intel P4 2.4 GHz CPU and 512 MB memory. All systems are connected by a 100 Mbps Ethernet network.

We give four queries, they are:

query 1:  select distinct p_name, p_mfgr, p_type, p_partkey  
   from union_part  
   where p_type like ‘%burnished%’ and p_name like ‘%lavender%’  
   order by p_partkey  fetch first 20 rows only;

query 2:  select ps_partkey, s_name, s_suppkey,  
   min(ps_supplycost) as ps_supplycost  
   from union_partsupp, unionSupplier, union_nation  
   where ps_partkey = 25 and s_nationkey = n_nationkey  
   and ps_suppkey = s_suppkey  
   group by ps_partkey, s_name, s_suppkey;

query 3:  select sum(o_totalprice) as totalordered, count(*) as num_orders, c_custkey,  
   c_name  
   from fed_customer, fed_orders  
   where o_custkey = c_custkey and  
   o_orderdate >= date(‘1997-10-01’) and o_orderdate < date(‘1998-10-01’)  
   group by c_custkey, c_name  
   order by totalordered desc  
   fetch first 10 rows only;

query 4:  select c_custkey, c_name, o_totalprice, n_name  
   from db2_customer, ora_orders, nation  
   where  
   c_nationkey = n_nationkey and  
   c_custkey = o_custkey and  
   o_totalprice > 450000 and  
   n_name in (‘Japan’, ‘China’, ‘Vietnam’, ‘India’).

We run JMeter to simulate concurrent users accessing the web page that returns dynamic results. We send 1, 10, 20, 30, 40, 50 concurrent requests in five seconds to the VIG system to observe the performance of VIG. The four queries return different numbers of results, so they can emulate the query in real world. The response time of queries are shown in Figure 6 (query 1, query 2, and query 4) and Figure 7 (query 3).
Because of the query caching mechanism of the virtual database engine, after having finished the first request, other requests can reuse the cached data and the response time will decrease dramatically. From the response time figure, we can get the conclusion. But we notice that query 3 presents different behaviours from the other three queries. Here we give out the reasons. Because we send out all requests in five seconds, it is easy to understand that the later request cannot immediately reuse the result of the first request because of the long time consumed by query 3. When we increase the time interval to send out all requests, the result of the first request can be utilised efficiently. We can see this in Figure 7. With more interval time to send out all requests, we can get better mean response time for query 3.

Figure 6  Mean response time of query1, query2, query4 at interval of five seconds

Figure 7  Mean response time for query 3 under different intervals of time
In order to confirm the agility of deploying data source, the life cycle of 100 data sources deployed in the test-bed has been monitored. Figure 8 shows that the average time of user interaction is around 1.8 min when creating, registering, or changing a virtual relation schema by a web-based toolkit.

Figure 8  Time cost of deploying data sources in the VIG platform

7 Conclusion

Heterogeneity, agility of changing and personalisation are primary issues in integrating and sharing information sources across multiple administrative domains in a distributed environment. The VIG addresses these issues with information virtualisation-based relation schema, virtual database engine, and a loosely coupled architecture, and provides a toolkit for building an information system to meet commonalities of these requirements. Preliminary evaluations show that this is a feasible approach for taking care of both performance and agility of change. The future works include enhancing the virtual database engine to support adaptive query on data sources when the data sources schema and dataset are dynamically changing.

Acknowledgements

The authors are thankful that the work is supported in part by the National Science Foundation of China (Grant No. 90412010), the China Ministry of Science and Technology 863 Program (Grant No. 2006AA01A106) and the China National 973 Program (Grant No. 2005CB321807).
References

