International Journal of Multimedia and Ubiquitous Engineering
Vol. 10, No. 2 (2015), pp. 371-382
http://dx.doi.org/10.14257/ijmue.2015.10.2.34

City-Wide Smart Healthcare Appointment Systems Based on Cloud Data Virtualization PaaS

Ping He[1], Penghai Wang[2], Jiechun Gao[3] and Bingyong Tang[4,*]

[1]*College of Business Administration, Donghua University, Shanghai 200051; Shanghai Hospital Development Center, China*
[2]*Shanghai IntelRay Information Technology Co., LTD, China*
[3]*Shanghai Hospital Development Center, China*
[4]*College of Business Administration, Donghua University, Shanghai 200051, China*

Abstract

While the cloud computing benefits healthcare IT modernization, it introduces new challenges for data integration, which is crucial for information sharing, data mining and business intelligence applications. Data virtualization brings a new strategy and related technologies to cope with these challenges, particularly suitable for cloud data services and SaaS (Software-as-a-Service) applications. The paper discussed a unique architecture and related technologies to implement the data virtualization for cloud data. The paper also demonstrated how this technology can be used in the real-world applications, and a city-wide healthcare appointment system has been implemented for example.

Keywords: *Cloud Computing, Data Virtualization, Smart Healthcare, Appointment System, PaaS, SaaS*

1. Introduction

Increasing healthcare challenges (such as rising cost, shortage of resources and inefficient services) become a global issue, and these challenges become main driving forces for the healthcare reform and modernization. It is certain that the healthcare IT (Information Technology) modernization plays a key role in the processes. The reduced cost can enable the limited resources to be wisely used, making the citizen access of the medical care and the protective health service easier. All of these goals are strongly dependent upon a modernized healthcare IT infrastructure, and some emerging technologies such as the cloud computing bring new possibilities to the healthcare.

In order to cope with the increasing healthcare challenges, the city of Shanghai has continuously supported some healthcare IT modernization projects in the past decade. In 2010, the project of Shanghai Health Cloud was granted and started to speed up its ambitious modernization process by leveraging cloud computing technologies. The healthcare appointment system is one of main sub-projects, and it is targeting to a city-wide, scalable, high performance system to serve 2400 million citizens. The goal of this project is to effectively and wisely use the limited healthcare resources.

The Health Cloud project planned to build 19 cloud data centers geologically located over 18 districts in Shanghai, China, where there are 600 hospitals, thousands of primary care centers or clinics. More than six thousands medical specialists are providing daily out-patient

* Corresponding Author

ISSN: 1975-0080 IJMUE

services and other cares. Without a city-wide healthcare appointment system, patients will keep blindly rushing to large hospitals. Some of patients even need to spent many hours in the long queue to compete few out-patient visiting opportunities. Available resources in these big hospitals are seriously mismatched to the demand of the services. However, there are a lot of healthcare resources are idle in smaller hospitals and primary care centers. A city-wide healthcare appointment system is expected to balance the increasing demand and limited healthcare resources.

Certainly, building a city-wide appointment system for 30 million citizens is not a trivial work. It is a challenge to synchronize high concurrent appointment requests with the dynamic work schedule of thousands of medical specialists in a transactional manner. It is more complex and problematic to integrate with hundreds of hospital scheduling system. Obviously, the scalability of the system is crucial. Capable to handle very high concurrent requests in the peak time is challenge as well.

New types of data integration challenges are emerging while enterprise applications started to move into cloud [1]. Before organizations realize these new integrations issues they may wind up with multiple SaaS (Software-as-a-Service) application silos, which will be more difficult to be integrated by traditional methodology and integration tools widely used today. Besides, numerous reasons make traditional data integration technologies invalid for many application scenarios. Integration of Big Data is an example [2].

OLAP (Online Analytic Processing) and BI (Business Intelligence) applications are often requiring data integration from discrete data sources or heterogeneous information silos to build a data warehouse. Most popular methodology often called ETL, stands for Extract, Transform and Load. When data becomes too big to ETL, when data in a cloud which not support ETL, when data is in other organizations which not allow to ETL, when the analytic application requires "real-time" data integration but ETL is not capable to meet the agility requirements, Data Virtualization is an appropriate technology for dealing with above issues [3, 4].

Data Virtualization is originally defined as an alternative data integration technology for some specific integration scenarios. However, today, it is not limited on data integration anymore. Data Virtualization is a virtualization technology to hide the complexity of real data, no matter the virtualized data coming from multiple data sources or sub-setting from a big data store. Data Virtualization provides client applications a transparent data access mechanism, which can significantly decouple the applications from the data complexity. As an alternative data integration technology, the data virtualization technology is not new. It has been around for many years. Originally, it was innovated as an alternative approach of data integration or data federation for data warehousing or BI (Business Intelligence) applications. This technology didn't gain broad attentions until new integration demand is emerged, particularly because the cloud computing and Big Data applications, which has become more popular recently.

Typical Data Virtualization was implemented as a middleware [5, 6]. A complete data virtualization platform may include an integrated development environment (IDE), a data virtualization server and a set of management tools. IDE provides user-friendly developing environment for modeling and mapping virtualized data model to discrete data sources. The data virtualization server is responsible for executing run-time queries from client applications. The query engine within the server, which is specifically designed to process federated queries across multiple sources in a wide-area network, optimizes and executes queries across one or more data sources as defined by the view or data service. Cost- and rule-based optimizers automatically calculate the best query plan for each individual query

from a wide variety of supported join techniques. Parallel processing, predicate push-down, scan multiplexing and constraint propagation techniques optimize database and network resources [4]. Data virtualization has been widely used in BI data federation, data warehouse extension, enterprise data virtualization, and other data integration fields [7]. Besides, a cloud data virtualization service, Connection Cloud, has on production recently for data integration of SaaS applications [1].

The organization of this paper is as follows. Section 2 introduced a unique architecture to implement cloud data virtualization as a PaaS (Platform-as-a-Service). Section 3 designed the cloud data virtualization PaaS particularly with some key components, i.e. the virtual SQL engine and the data grid bus, a distributed computing infrastructure for the cloud data virtualization. In the section 4, implementation of a city-wide healthcare appointment system based on the cloud data virtualization PaaS is demonstrated. Section 5 made a conclusion and discussed the future works.

2. Architecture of Cloud Data Virtualization PaaS

As a PaaS, it is crucial to make development support and run-time data services highly transparent, scalable and simple. At the architecture level, the Cloud Data Virtualization PaaS (CDV PaaS) introduced the following two key technologies to achieve its simplicity, high scalability and transparency.

(1) Virtual SQL engine;

(2) Data Grid distributed computing infrastructure.

Based on this distributed computing architecture, the CDV PaaS turns the discrete data sources into the data grid nodes. Different from other data integration approaches, the data grid moves necessary computing works to the location where data resides, rather moving the data to a centralized server for processing. Therefore, integrating a new discrete data source into CDV PaaS is equivalent to extend the data grid with a new data and computing node. For instance, there are databases and applications in the "Site 2", as shown in Figure 1, which needs to be integrated by CDV PaaS. A CDV PaaS client agent, marked as "Client Node 2", can be installed and configured inside of Site 2, either on an existing local computer or an new exclusive server machine. This CDV PaaS client agent enables application to use CDV PaaS data services just like connecting to another relational database. Meanwhile, properly mapping the local database(s) into CDV PaaS makes loacl data sources being integrated into CDV PaaS's virtual data services.

As shown in Figure 1, the CDV PaaS is composed of one master node and n client nodes. All nodes are interconnected via a unique bus based on shared JVM (Java Virtual Machine) technology on the top of TCP/IP network stack. Shared JVM architecture enables the data grid nodes working in one virtual machine environment. This architecture also greatly simplifies the implementation of CDV PaaS as a parallel and distributed computing machine.

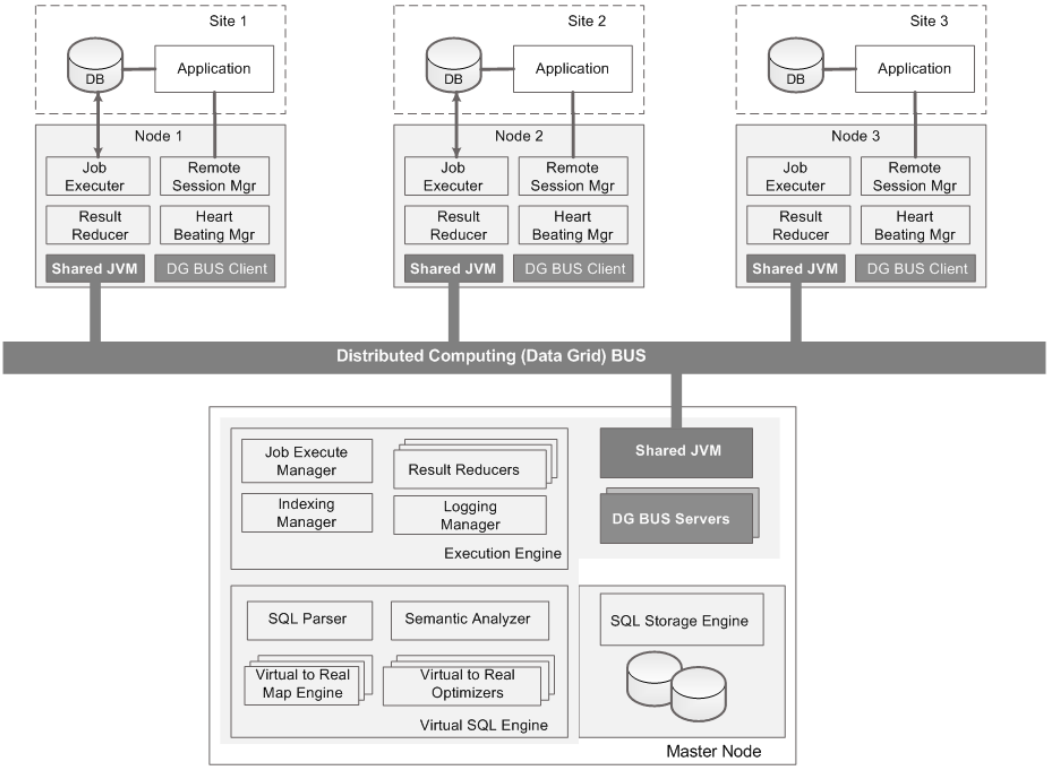

Figure 1. Architecture of Cloud Data Virtualization PaaS

The master node is the core of CDV PaaS and this node is composed of the following four main functional units.

(1) Virtual SQL engine, which supports a full-blown SQL relational database with extended capabilities to handle cloud data virtualization, including virtual data modeling, mapping discrete data sources to the virtual data models, processing queries from client applications based on virtual data models, *etc.;*

(2) CDV PaaS execution engine, which is responsible for building distributed jobs, managing execution of jobs over the data grid, monitoring job execution and handling exceptions and errors, aggregating final results and returning to the caller application;

(3) Local database storage engine, which provides data storage services for both virtual and concrete data storage;

(4) Data Grid servers, which is the core of distributed computing bus. The data grid servers provide an infrastructure to cluster all data grid nodes via shared JVM. This mechanism enables all data grid nodes communicating just like inter-thread communication in a single Java virtual machine.

Importantly, a CDV PaaS client node, also known as the data grid agent, is responsible for processing CDV jobs which related to the data located in the client site. These works will be carried out by the Job Executer unit in the CDV PaaS client node. Meanwhile, a CDV PaaS client node also works as one connection server for applications in the local client site, the Remote Session Manager in the agent enables client applications to establish a connection to the CDV PaaS via the LAN, or even local machine if the CDV PaaS agent is installed and configured on the machine.

Besides, a data grid agent also share extra responsibilities of CDV, handling "reducing" jobs by the Result Reducer unit for aggregating results in parallel is an example.

3. Design of CDV PaaS

As one of core Big Data technologies, the MapReduce Model is a proven programming model to streamline the distributed computing with commodity servers [8]. Although target problems of CDV PaaS are different from the Big Data technologies, the scalability requirements, the capability for high concurrent requests are similar. The principle and strategy of MapReduce technology can be borrowed to simplify the implementation of CDV PaaS distributed computing infrastructure. The lifecycle of processing a typical query request in CDV PaaS is comprised of four main phases:

(1) Parse and analyze the query statement(s);
(2) Map the query to corresponding executable jobs for each related data grid node;
(3) Execute jobs by corresponding data grid agents in parallel;
(4) Reduce output results to aggregated final results by data grid agents in parallel.

3.1. Virtual SQL Engine for Parsing and Mapping

First two processing phases, parsing and mapping are carried out by the Virtual SQL engine in the master node, following execution of jobs, and reducing results are performed by the data grid client agents in parallel. Figure 2 exhibited the workflow of CDV PaaS query processes.

The Virtual SQL engine is one of core units in the CDV PaaS and this unit is comprised of the following two key components.

(1) SQL parser with data virtualization extensions;
(2) Mapper for transforming a virtual query into node-specific executable jobs.

CDV PaaS has selected a SQL engine to manage the virtual data models. Meanwhile, a standard-compliant SQL API can be utilized as the primary API of CDV PaaS.

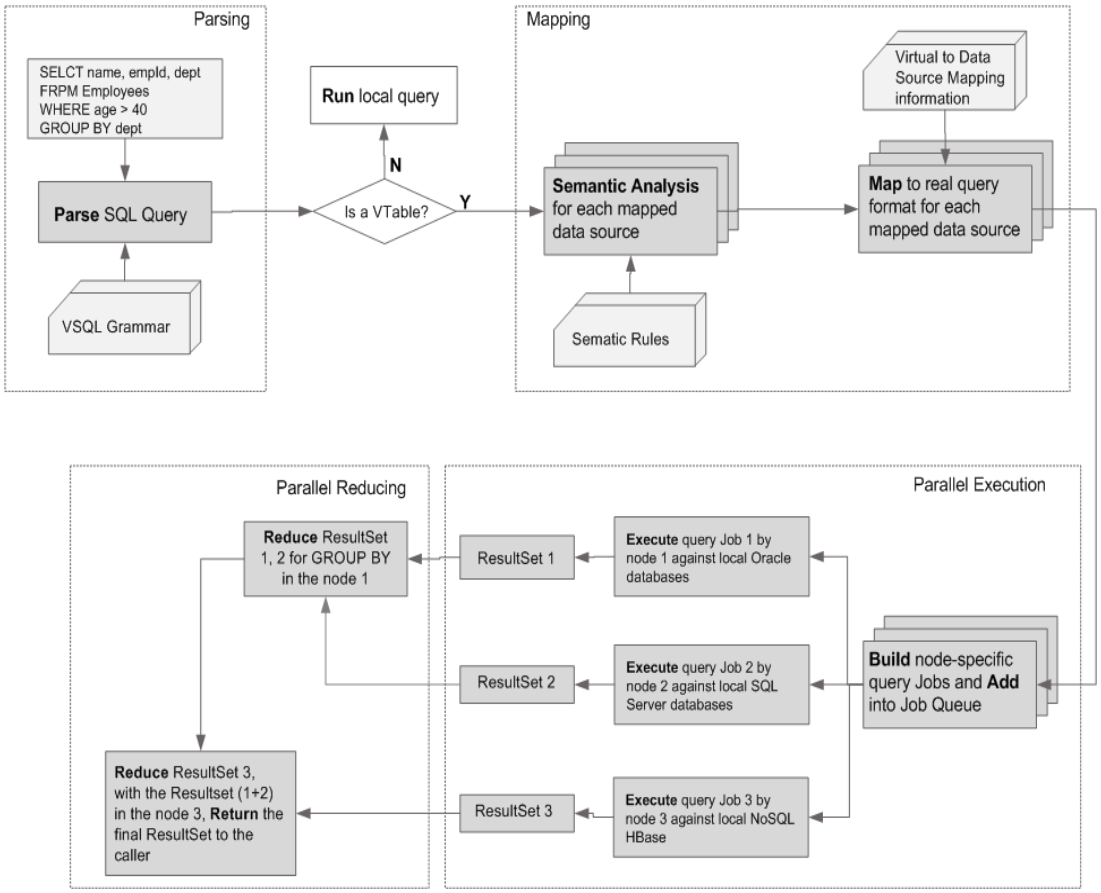

Figure 2. Query Processing Workflow in CDV PaaS

The SQL API makes existing applications easy to be migrated into cloud. This strategy could significantly lower the learning curve for adoption of CDV PaaS by traditional enterprise users. DDL (Data Definition Language) of the SQL engine has been extended for data virtualization. For instance, to create a virtual Table, a new DDL command, CREATE VTABLE can be used. A regular table also can be altered to a virtual Table by ALTER <table_name> INTO VIRTUAL command. CDV PaaS kept the DML (Data Manipulate Language) unchanged to minimize the effort for migrating applications on CDV PaaS.

Another key component in the virtual SQL engine is the Mapper. It is responsible for transforming a SQL query against the virtual data model into corresponding executable jobs to run on each mapped data source. As shown in Figure 2, the mapping SQL query statements into executable jobs includes following four main steps:

Step 1 Retrieve mapping information for all tables and columns appeared in the query statement. Each virtual table might map to n concrete tables, each column in the virtual table mapped to n columns in the corresponding concrete tables. Mapping information should be captured when a new data source was mapped to the CDV PaaS. Mapping information is persisted in the internal database of the CDV PaaS;

Step 2 Conduct semantic analysis for each segment of parsed query statement, apply semantic rules if need, prepare required information to map each virtual table to the concrete tables, map each virtual column to real columns.

Step 3 Map the virtual query to concrete queries. Each concrete query object represents a query which is executable for mapped data source. If the mapped data source is an HBase data store, the concrete query will be mapped to a series of HBase API calls in a Java class.

Step 4 Build executable jobs for CDV PaaS client agent based on each concrete query object. Then, add these executable job objects into the New Job Queue for execution by corresponding client agent.

3.2 Distributed Computing Infrastructure for Parallel Job Execution and Reducing

The infrastructure to support distributed query job execution is another core unit of CDV PaaS. This infrastructure is called Data Grid Bus in CDV PaaS, or short for DG BUS. The DG BUS is built on the shared Java virtual machine technology. This technology enables Java objects can be shared between connected CDV PaaS nodes seamlessly. Figure 3 demonstrated how the CDV PaaS master node and client agents communicate each other via this mechanism.

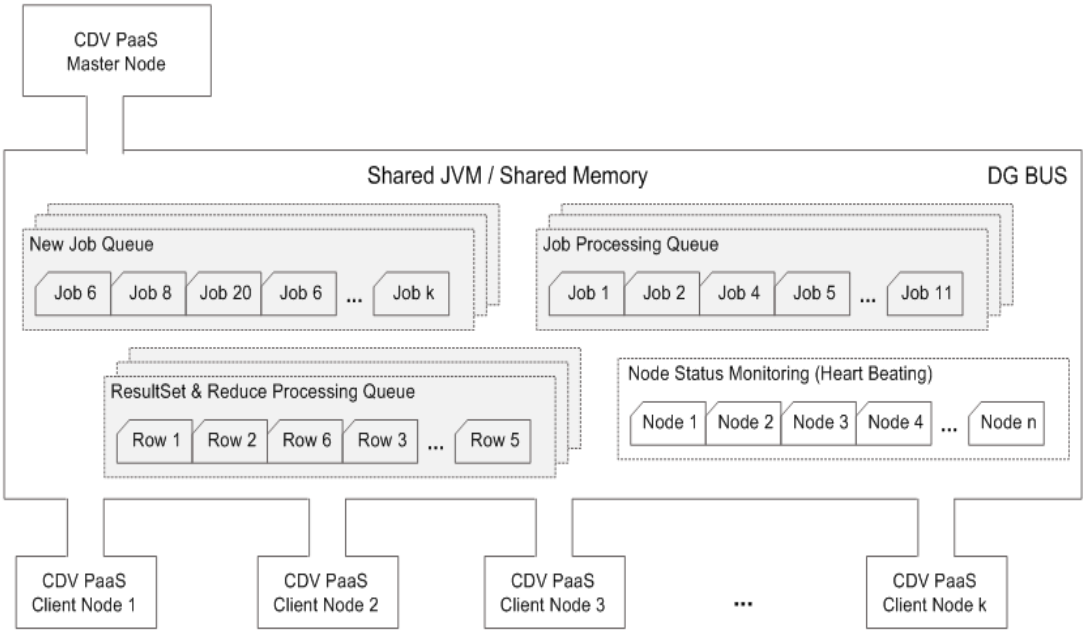

Figure 3. Architecture of CDV PaaS Data Grid Bus

Because the DG BUS is built on a shared JVM, the shared Java objects, such as the "New Job Queue" in above diagram can be treated as an inter-thread shared object in a single JVM. This mechanism substantially simplified the communication and collaboration among nodes and parallel computing processes. Although DG BUS is built on shared JVM, only necessary objects are shared among CDV PaaS nodes. This "Share-something" architecture can minimize the network traffic. There are the following three types of objects are shared via DG BUS.

(1) Job objects and their Queue container, including the new job queues and the processing job queues. Each query statement will have its own job queues.

(2) Results objects from job executions. Usually, a job execution may return a *ResultSet* object in which contains multiple Rows. A virtual query may produce *n*

ResultSet objects from *n* client agents. These ResultSet objects need to be aggregated based on GROUP BY or ORDER BY conditions via multiple reducing jobs.

(3) DG BUS employed a "hart-beating" mechanism to keep monitoring the important life-index of each node and all network components. The life-index objects for each node are shared.

The work flow diagram in Figure 4 demonstrated how to map, build and initiate a new virtual query via DG BUS. After a new virtual query is parsed, analyzed and mapped to the concrete query objects, the execution engine in the CDV PaaS will create a new job queue in the DG BUS for the query. The execution engine then build the executable jobs based on each concrete query and add jobs into the new job queue. The client nodes will be notified when a new job added into the queue.

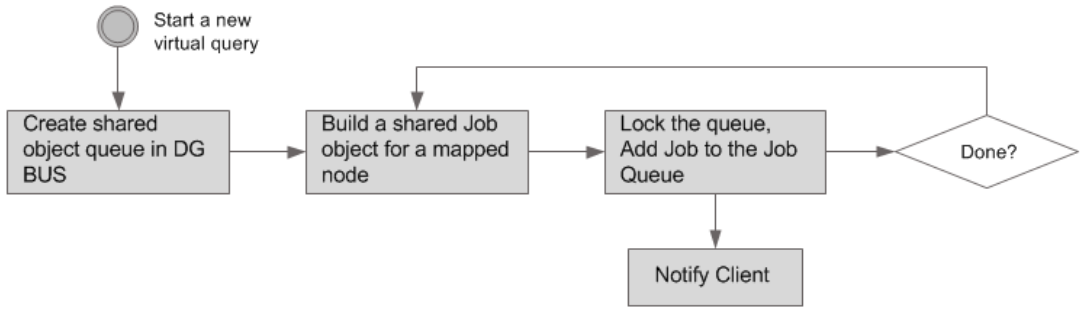

Figure 4. Map, Build and Initiate Distributed Jobs

A client agent keeps checking if there is any query job for it. If so, the job will be removed from the new job queue by the client agent and add it into the processing job queue, then execute it. The processing status of the job is being monitored by the execution engine. The workflow is shown in Figure 5. After the job execution is completed successfully, the job will be removed from the processing job queue. The output results and reducing job will be created and added into the reducing queue. When an exception is thrown during job execution, the job may be retried if the exception is retrievable. If a fetal error encountered, the whole query may be aborted the job on the particular client site may be ignored based on if the query is a "transactional" or not. DG BUS has sophisticated mechanism to cope with any node failing, rebooting and other infrastructural exceptions to ensure the jobs are not lost or run into inconsistent status.

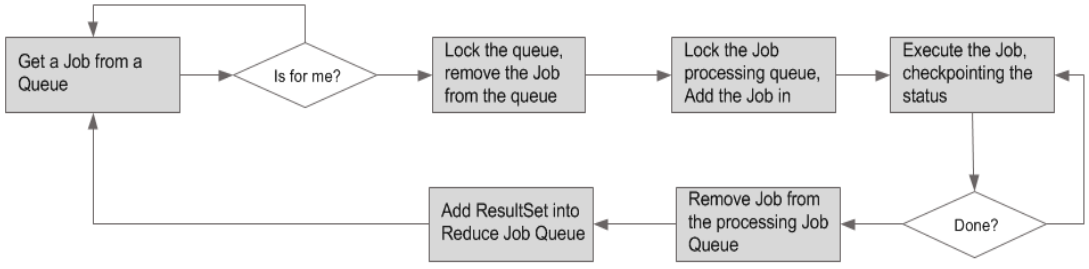

Figure 5. Workflow of Job Execution

Before the results can be returned back to the caller application, the results firstly need to be converted to unified data types for each column, because the data types in the results returned from each client agent may be different. Then, the results need to be aggregated into one result set. Usually, the query comes with GROUP BY and ORDER BY clauses, processing a lot of rows with GROUP BY or ORDER BY is computing intensive. CDV PaaS leverages its parallel computing power and utilizes the "Reduce" technique to accomplish results aggregation and other necessary processing in parallel. In Figure 6, the client agents are responsible for execution of reducing jobs.

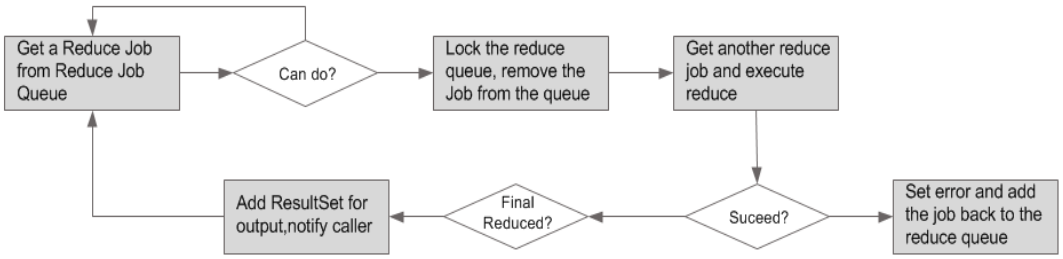

Figure 6. Workflow of Reducing

A reduce job always take two result sets. The result sets can be the results output from a query job execution, or an intermediate result set from previous reducing jobs. A client agent pick one reduce job from the queue, check if it is capable to execute the job. If the job is workable, the reduce job will be removed from the queue and executed. Otherwise, it will try next job until no more job available.

4. City-wide Healthcare Appointment System Based on CDV PaaS

It is very challenge to make the limited healthcare resources utilized wisely in a world-largest city populated with more than 30 million people. A city-wide appointment system to facilitate patients finding a medical specialist in convenient time and location is an essential application of Shanghai Health Cloud project. The appointment system is a large-scale, public-facing, real-time and mission-critical information system which serves up to a million of requests in a single day. The major challenges include the following two ones:

(1) Scalability challenge: step by step add over 600 hospitals and thousands of primary care providers into the system;

(2) Performance and availability challenges: ensure the system is capable to handle very high concurrent load in the peak time, also capable to effectively utilize computing resources during non-peak time.

Due to variety of reasons, a centralized SaaS architecture is not an option for this city-wide healthcare appointment system. The districts in the city have built their own healthcare appointment applications which better fit their customized requirements. Therefore, a distributed architecture based on CDV PaaS becomes an appropriate choice. First, the city-wide appointment system on the CDV PaaS can simply grow and scale out by adding new district-centric appointment applications in the district cloud centers. The district-centric appointment application can be customized to meet the special requirements uniquely for the health care services in the district. Second, the high concurrent request load can be distributed into multiple appointment applications, which increased the scalability for increasing the request load and enhanced the overall

availability of the system, so that the failure of one district-centric appointment application will not affect the entire network.

The diagram in Figure 7 exhibits the architecture of the appointment system. With this architecture, it becomes fully transparent for users to access whole city's data without knowing the complexity. Users can find available medical specialist in expected region and time frame in entire city. For instance, to find a diabetes expert who has available slot in current week, all specialists who are the expert of diabetes will be searched in each site in parallel via CDV PaaS. A district-centric appointment application can be built in the local Health Cloud center, as shown in Figure 7. The application server(s), web server, including CDV client agent are deployed in an elastic computing IaaS (Infrastructure-as-a-Service) bucket, which provides elastic computing power and network bandwidth for effective resource demands. A dedicated IaaS bucket is used for databases, which guarantees exclusive computing resources always available for the databases. A CDV PaaS client agent is deployed in the local (same network segment or even local server machine) of district-centric appointment application. The CDV PaaS is running on the dedicated network which interconnects the Health Cloud centers. This makes the district-centric appointment application to access data resides in other Health Cloud centers as accessing another local relational database. The significance of this architecture is its scalability. It enables the system seamlessly expending its geological coverage without suffering from customization limitations commonly existing in SaaS applications.

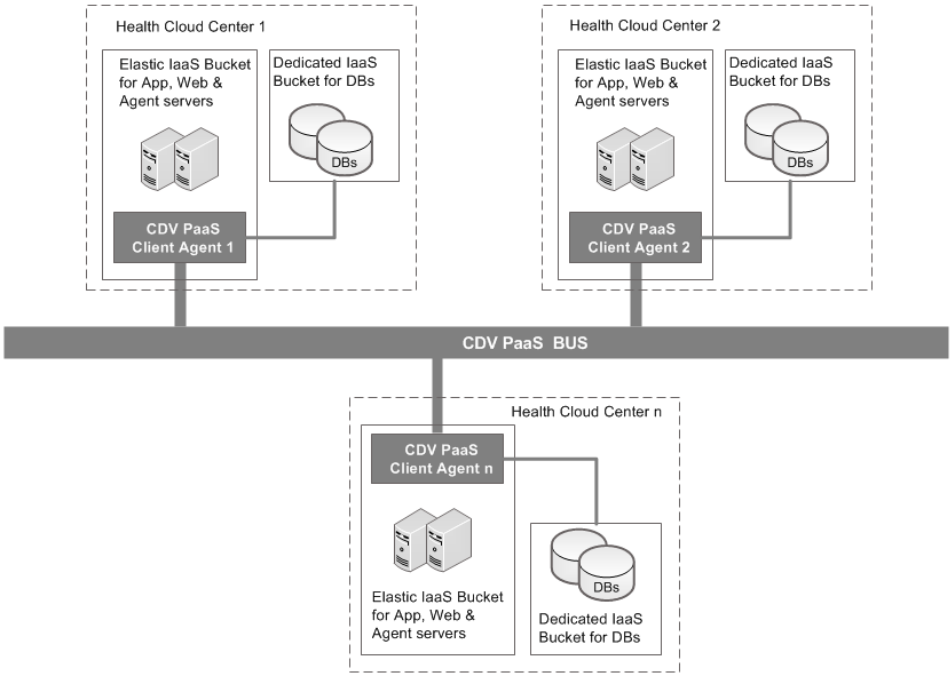

Figure 7. Architecture of Appointment System

Adding a new district-centric appointment application is relatively independent on others, this enables the system is easily built to cover one district by another. The distributed architecture is also benefit to cope with soaring up higher concurrent request load. Beside those patients who knows which doctor they want to make an appointment

with, a large portion of patients have to find out which medical doctor or specialist is better to fit their needs. Find a suitable and available specialist may take 70% of time to make an appointment. Figure 8 demonstrated a workflow to find available medical specialists by given disease type.

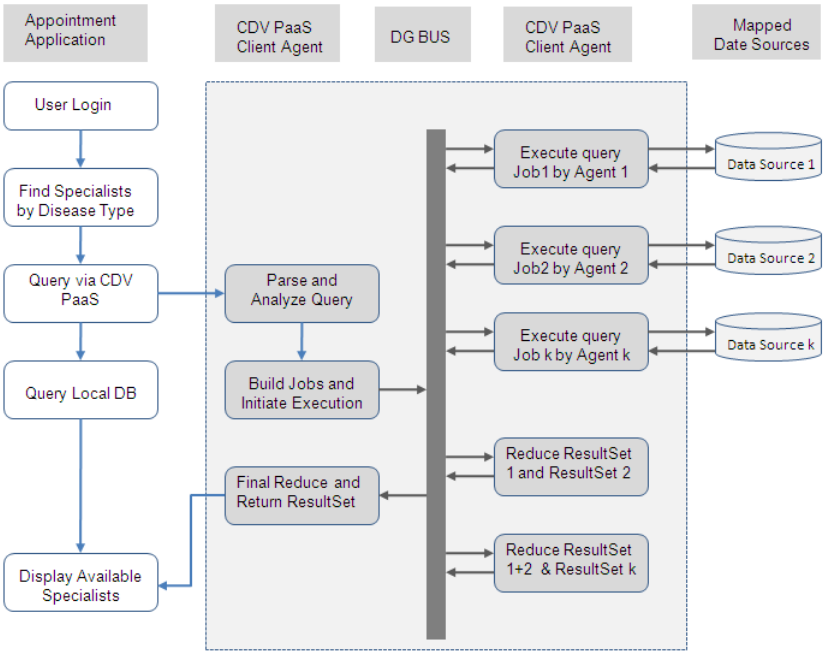

Figure 8. Find Available Medical Specialists by Disease Type

When a patient logins to the appointment system, no matter which district appointment application is routed to, the patient is able to access all medical specialists in other districts once the sites join the CDV PaaS network. As the example workflow shown in the Figure 5.2, searching available medical specialists by given a disease type will go through the local database and the CDV PaaS network respectively. Same query statements will be issued to the local database and the CDV PaaS client agent, the search will be executed in parallel, the aggregated results after multiple reducing will be returned just like returning from the local database.

5. Conclusions

Emerging new IT technologies, including cloud computing, big data, make data integration and data virtualization more complex and cost. The paper introduced a new approach to implement data virtualization for cloud data services and simplify data access for SaaS applications. By combining a virtualized SQL engine with a data grid style distributed computing mechanism, the data virtualization technology is successfully extended to cope with data integration challenges directly or indirectly caused by cloud computing and big data technologies. The technology introduced in the paper has been implemented as a PaaS platform, called CDV PaaS. CDV PaaS makes distributed cloud data in multiple cloud data centers can be virtualized as a single relational database. This technology substantially simplified the data service for any SaaS application which needs to access multiple data sources discrete in different cloud

data centers. This technology makes a large-scale, federated application can simply grow and scale. A city-wide health care appoint appointment system has been designed and implemented based on CDV PaaS.

Acknowledgements

We sincerely thank good advice from the reviewers and the financial support of the grants from the "scientific and technological innovation action plan" major project of the Science and Technology Commission of Shanghai (Shanghai health cloud application demonstration project, Project No.: 11DZ 1500100).

References

[1] M. Vizard, "The Role of Data Virtualization in Cloud Programming Web", (**2012**), http://blog.programmableweb, com/2012/07/31/the-rise-of-data-virtualization-in-the-cloud/.
[2] S. Yaskin, "Thoughts on Big data and Data Virtualization," Cloud Computing Journal, (**2012**), http://cloudcomputing, sys-con.com/node/1803581.
[3] L. F. Rick, "Data Virtualization for Business Intelligence Agility," (**2012**), http://purl.manticoretechnology.com/ MTC_Common/mtcURLSrv.aspx?ID=12917&Key=FE72CA6B-C6D6-4DA1-91D6-5CDE20B85E33& URLID=17966.
[4] J. R. Davis and R. Eve, "Data Virtualization", Going Beyond Traditional Data Integration to Achieve Business Agility, Nine Five One Press, (**2011**).
[5] R. V. D. Lans, "Data Virtualization for Business Intelligence Systems", Revolutionizing Data Integration for Data Warehouses, Morgan Kaufmann Publishers In, (**2012**).
[6] L. Weng, G. Agrrawal and U. Catalyurek, *et al.*, "An Approach for Automatic Data Virtualization," Proceedings. 13th IEEE International Symposium on High performance Distributed Computing, (**2004**), pp. 24-33.
[7] R. Lawrence, "Integration and Virtualization of Relational SQL and NoSQL Systems Including MySQL and MongoDB," Proceedings of 2014 International Conference on Computational Science and Computational Intelligence, (**2014**), pp. 285-290.
[8] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Proceedings of 6th Symposium on Operating Systems Design and Implementation, (**2004**), pp. 137-150.

Author

Ping He, she is a Ph.D. student at Donghua University. Her interests of research include cloud computing, Smart Healthcare Appointment Systems, etc.

