WALAILAK JOURNAL
http://wjst.wu.ac.th Information Technology

Scalable Data Integration System using Representational State Transfer

Pornthip MENSIN[1,*], Phongphun KIJSANAYOTHIN[2] and Worajit SETTHAPUN[1]

[1]*Asian Development College for Community Economy and Technology, Chiang Mai Rajabhat University, Chiang Mai 50300, Thailand*
[2]*Faculty of Engineering, Naresuan University, Phitsanulok 65000, Thailand*

([*]Corresponding author's e-mail: pornthip.wong@gmail.com)

Received: 18 November 2015, Revised: 26 April 2016, Accepted: 9 May 2016

Abstract

Data integration from remote data sources is complicated, due to a large number of heterogeneous information sources that are usually stored in different data schema. This topic is challenging and requires the management of information system enterprises to be able to integrate distributed data and applications effectively. Numerous methodologies and systems have been proposed and developed to address the related issues from different aspects. However, there is still a lack of methodological support and appropriate data integration development in an internet environment that requires good performance and scalability on large-scale data execution. The concept of centralized design for the integration of distributed data repositories as server-based processing solves the traditional problem of tight coupling, but does not address the integrated logic task. To solve this problem, this paper describes a data Integrated Client-Rich approach (ICR) and implementation of Thesis Core System (TCS) data integration system that follows the Representational State Transfer (REST) architectural principle for system implementation. REST is a new principle for system development that has good support for development tools to meet the needs of data integration, and obtain the advantages of simplicity, scalability, and performance. We design a testing scheme to evaluate and analyze the scalability of traditional data integration with server-based integrated processing and TCS data integration. The results show that our approach is more suitable, scalable, and practical for Internet-scale distributed data integration.

Keywords: Data integration, distributed computing, architectural approach, REST-based

Introduction

The goal of data integration is to provide a unique, transparent and homogenous view of a set of heterogeneous data sources. Since the increase in distributed data providers, data integration has become even more necessary, because of the difficulty in answering the desired question, which requires the combination of information from different providers. Consider the task of trip planning; all information which helps make the decision for these tasks can come from multiple service providers, such as airlines, car rental companies, and hotels. Each of the service providers has its own syntax and semantics for information (e.g., price for renting a car per week, the price for a deluxe room per day, the price for a one-way flight). Since the emergence of e-commerce (i.e., all of the required information being able to be found on the Internet), humans can plan a trip without using a data integration service such as Expedia [1], but it remains a very time-consuming job.

Generally, data integration may be created with different approaches and paradigms, such as Materialized [2,3], or Mediated [4-6] approaches. The first approach, Materialized, is the process of creating an integrated result of the query processing and storing it into one global data source (i.e. a data warehouse). This stored integrated result is called a materialized view. In contrast, the Mediated approach

generates logical views onto the integrated data without storing it. These concepts transform user queries over a mediated schema into queries over local schemas that reside on the local data source. As such, its integration results get updated instantaneously as new data arrives. This differs from the Materialized approach, which can only be updated periodically (e.g., daily) in the data warehouse and through the process of 3 general operations: Extracts, Transfers, and Loads (ETL). With the limit of the overhead, the integrated result is difficult to update frequently. As a result, the Mediated approach is a more preferred option [7] due to being able to ensure validity, accuracy, and consistency. These approaches operate quite well in a closed application environment. However, in an open distributed environment, with the number of the requests remote to the server, the traditional approach may cause problems due to the high computation of integrated logic. Hence, a generated query may need to combine large results from a distributed data source (e.g., to calculate the total number of available rooms of a certain kind of hotels from multiple providers during a certain time period), which may lead to performance issues. Moreover, a higher number of operations lead to the problem of interface complexity. Designing in order to satisfy the scalability and limiting the type of existing resource (hardware, operating system, data source) is a critical issue. Significant active research areas have been published for decades of areas such as Load-Balancing [8], parallel approaches [9,10], and query optimization [11].

Keeping with the traditional architecture approach, the complete integrated result is executed on the server side that has a limit of resource usage. To address this problem, some recent publications have begun to provide an architectural style for networked hypermedia applications, called Representational State Transfer (REST) [12] (e.g., [13,14]). In the REST style, some architectural properties used to enhance advantages such as scalability, low coupling, and addressability, are the keys to the success of the web [15]. It provides the system with simple development, a lightweight quality, and high performance [16,17]. However, none of these approaches support dimension application logic on the server, which is essential for data integration. A major difficulty is the significant performance issues with large-scale integrated result sets generated on the server side. In this paper, we propose an approach to achieve scalable data integration in real time when handling a large number of user queries. In addition, a prototype to integrate distributed data sources will be developed using REST architecture. The outcome of this work will provide guidelines for system developers and architects to be able to use data from multiple data sources to meet their scalable objectives.

Related works

Internet-based computing is a 3-tier architecture commonly used for reliable data integrating application [18,19]. This architecture has successfully provided higher system performance, availability, and flexibility [20,21]. Its design explicitly separates application logic outside data tiers as the middle-tier, which is more flexible to use and supports changing businesses. Therefore, many studies have proposed to design a system using 3-tier architecture for cloud computing [22], decision support [23], web services, and enterprise application integration, etc., to provide a well scalable and maintainable system. However, these architectures are not used for developing an application as a server-side scripting language. In a typical server-based computing system, the application or integrated logic is responsible for transforming and combining the retrieved data into the total integrated result. After all integrated result are executed, integrated logic sends it back to its client for displaying and rendering. In this architecture property, the clients are thin [24], able to store data, and execute all of the applications. In addition, a data rule is placed at remote servers on the network. In the case of a server-centric application which communicates and exchanges data with other clients on the network, these developments do not provide guarantees for performance issues with large-scale integrated result sets generated on the server side, because they may lead to application performance problems [25].

Research papers have been written in the area of optimization approaches for resource management in distributed computing. This simple method improves the performance of a system in distributed computing, such as the Load balancing technique [8]. Various studies have revealed the concept of a parallel approach to data integration from remote web services. An algorithm is proposed to reduce the total data integration time, with parallelization of data integration tasks from remote heterogeneous

sources [10]. In a similar setting, Distributed Handler Architecture applied parallel handler execution to improve web services [9]. This approach separated the handlers from logic, to run them on different cores or computers in order to provide additional computing power.

Researchers have developed algorithms for optimization query processing [26,27]. The approach of Chen and colleagues used algorithms in features of interface or representations for data sources, in order to reduce inter-source processing of data across sources [27]. Karnstedt and colleagues focused on queries over RDF data. They proposed decentralized indexing and query processing approaches to apply a sophisticated cost model to enable cost-efficient query planning and query execution focusing on database management [28]. In data integration system (DIS), from the work of Chen and colleagues, combining the concept of optimization query processing and parallel approach was applied to the query execution model with a group of parameterized queries as the sub-queries. The approach was effective in reducing the communication overhead and the average query completion time [11]. However, most approaches have been proposed for in grid computing, cloud computing, and database domain that uses the concept of Centralized Approach. These methods do not provide availability guarantees.

In the decision support domain, Kang and Hong proposed a BIM-GIS-based architecture model for effective integration into a geographic information [29]. The results showed that it has benefits, such as reusability and extensibility. However, this architecture is an ETL-based approach that uses OLAP query based on the static data cube approach [30] that has successfully managed large data retrieved from internal databases. This was not designed for real-time operation, and could lead to the problem of up-to-date information.

The business domain integration model was required to solve the problem of heterogeneity of data representation in the applications that often cross organizational boundaries and are related to service-based approaches [31]. A major aim of the web service platform is the integration of existing software and information systems. Pahl introduced a technique for improving scalability through modularity that builds mediator service into a web service-oriented information system architecture. It could automatically generate layered rules guided by the taxonomic hierarchy of the underlying ontology [32]. Nevertheless, these techniques are not designed for the computational task of integrated logic, and may require the involvement of third-parties or centralized services. With new distributed computing technology and service-based approaches, it is used for interoperation among heterogeneous platforms and enterprise application integration on the network system.

As web technologies evolved, REST-based approach has become one of the most important technologies for web-based distributed applications [33], because it provides tight business process integration with cost-effectiveness and efficient alternatives [34]. The REST principles introduce a new kind of abstraction, the resource, with a resource-centric conceptualization [35]. Resources are identified through URI (Uniform Research Identifiers) for interface uniformity. Accessing the resource of REST style uses the original architecture of World Wide Web, which relies on the HTTP protocol. Its design focuses on only the roles of Resource URI that are addressed and transferred over HTTP. Application implementation is much easier with basic HTTP methods (`get`, `put`, `get`, and `delete`) to access a resource. With the important characteristic of REST style being statelessness, designed systems have gained scalability and performance [16]. Many prominent internet companies, such as Google, Amazon, Yahoo, and eBay, used REST for designing their REST web services.

For many years, data integration development that built upon the REST-based architecture was hidden by the limitations of the features, for example, the lack of metadata descriptions, having little vendor tool support, and programming interface difficulty [36]. Several works have been proposed to improve the limit of these features, such as REST+T [37]. As a result of this shift to REST-based data integration application, the evolution of features has increased the development of REST application [26,38]. Many researches in service-based systems used REST architectural style in implementation [33,39,40]. Fuentes-Lorenzo and colleagues proposed SWIAD architecture that used the advantages of the REST-based approach for solving data integration process problems [41]. The main concept was algorithms that support mapping and querying, including 2 modules: the Mapping module and the Access module. This was an efficient approach, because it applies a complete RESTful manner to access the

resource directly and support real-time data integration. However, this concept focused on the ontologies domain. Feng *et al.* and Hamad *et al.* [15,17] analyzed and compared the performance of REST-based research in the mobile environment, REST-based architecture is better for server-based application in terms of scalability, higher flexibility, and performance due to lower overhead. Meng and colleagues proposed that data integration implementation of an internet solution in large scale with REST architectural style similar to our work [13,26]. However, these approaches did not consider the problem of performance constraints in the dimension of application logic on the server. This is essential for data integration in distributed system. In the next section, we will present a new approach for data integration system development to achieve interoperability and scalable objective.

Materials and methods

Experimental research methodology was applied in this work [42]. The main goal was to propose a new architectural design style for developing data integration systems, to prove the legibility of the proposed concept with the implementation of the prototype system, and to evaluate the legibility of the result in scalability by benchmarking with a based-line and comparing with traditional architecture. Our research instrument are tools related to the implementation of a system prototype. Prototype design is based on 4 phases: Analysis, Design, Development, and Deployment (ADDD Model). The ADDD Model is a generic step by step process to develop systems for designers and developers. The model is the guideline for planning and creating their systems. In the data collection and analysis step, the quantitative experimental data comprise the test results from the impact of the user's request simulation, compared with our proposed approach and traditional approach. They are statistically analyzed using linear regression analysis [43]. The results are statistically and descriptively presented as the developed system, according to our proposed process.

Proposed Approach: We introduce a new design style for developing a data integration system at the architectural level. The primary goal of our design style is to act as an access provider to distributed data resources without an integrated process. Afterwards, the output data response is returned to the client in order to integrate the data.

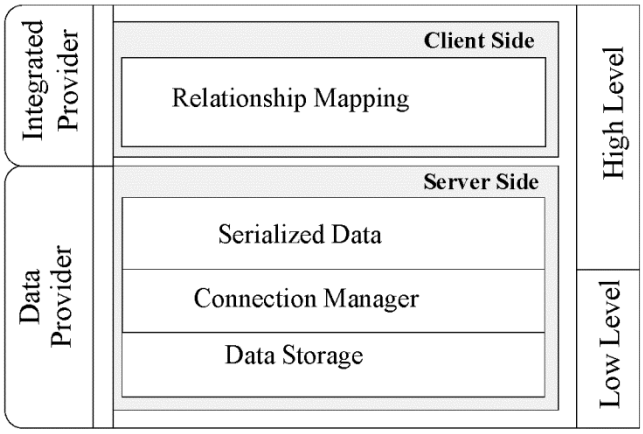

Figure 1 Integrated Client-Rich (ICR) model.

The model of data integration is called an Integrated Client-Rich (ICR) Model, consisting of a data provider layer and an integrated provider layer, as shown in **Figure 1**. This architecture is based on the client-server architecture, which has the common design styles of in web applications, and suits our

requirements. The environment used is distributed processing. In the subsequent details, we will describe the purpose of each layer and the relationships between the layers.

The data provider is a layer providing access data from the distributed data source with local schemas, and transforms the data to generic schemas before sending the result back (transformed data) to the next layer. This layer consists of data storage, a connection manager, serialized data and the components of server security. For the bottom level, the data storage, the information from the distributed data source is stored in many organizations. This information has a different data structure and a heterogeneous data storage vendor, depending on each business requirement. With the benefit of internet and other communication networks for exploiting data sharing, creating a data connection remotely to each data source is used via a layer called the connection manager. The connection manager hides several existing data by representing the standardized view, which is accessed through the so-called integration view (as part of the integrated provider layer). The standardized view specifies how data are served and how data can be queried. All data from each of the data sources is transformed in a form of standardized data, namely, serialized data. The serialized data hides the logical content of each source by transforming them to the view of data in the generic meta-model. The layer provides a view of the concept of the relationship of the structural metadata for data residing in the primary source to the data model in the virtual source. The result represents a standardized data support used to combine elements from the different data sources to the integrated scheme on a virtual source through so-called relationship mapping (as part of the integrated provider layer).

The integrated provider contains logic to integrate the retrieved data from the connection manager. It is responsible for combining data and merges the results into a single view. Relationship mapping describes the relationship among individual data formats from multiple local data sources. It is important to consider that the relationship is created without any consideration of existing data heterogeneity expressed in modeling languages. The encapsulated structure of the data is different from data sources. It allows access to integrated data through a virtual view of data integration as a conceptual data view. This view describes data requirements from the business point of view and provides data representation for system development at the application level. Retrieved data from the data provider is executed and rendered to integrate the results with integrated query processing via the data integration process logic that resides in this layer. This is the logic for the client application.

The Prototype Implementation: In order to validate our proposed concept, a system prototype is implemented following the proposed approach, called a Thesis Core System (TCS). The TCS has the objective to support the tracking of thesis progress for graduate students. The system requirement needs to be associated with many data sources which reside in different locations: the Graduate Student Registration Information DB (RegnuDB), the Teacher Information DB (HrnuDB) and the Thesis Information DB (GradnuDB) (Oracle for graduate student and teacher and MySQL for thesis tracking). The 4 phases of this research are designed as follows:

Analysis Phase, to study the theoretical framework of related theories, including the model of problem-solving in the context of the heterogeneous data source with the ICR architecture style and REST principle. Several data integration systems in many research works have implemented REST principle into their systems. The ICR approach used REST principle for the design of data integration. The main aim of TCS is to develop a tracking and reporting system as a step-by-step timeline of the thesis progress report. This system will aid the graduate students in finishing their research in a timely manner. The goal is to access primary data sources with a common view, while the end users are allowed to access the integrated result. A list of boundaries consists of 5 groups: *Administrator*, *Teacher*, *Graduate student*, *Faculty staff* and *Grad staff*. These requirements have to interact with multiple types of data sources in the external organization, which are relational database tables. We can specify the requirements of TCS as being access to multiple data sources, accuracy of information, and efficiency and scalability management.

Design Phase, designing a model according to the proposed approach. This method will solve the efficiency problem of data integration processing, as illustrated in **Figure 2**. We illustrate our approach using the concept of providing data for independent integration. The model shows the overview of system architecture related to control and data flows within the system. From the end-user perspective, the client

will send requests to attach the needed information for each query via URI interface to the server. Then, the application on the server will distribute commands remotely to the multiple data sources. The data communication exchanges between client and server application is a JSON object which is formatted to represent resources providing a simple explanation of metadata for the data structure in the data source. The TCS consists of a 2 step design: (1) Backend design for providing data (2) Frontend design for integrating data. We will elaborate on each design.

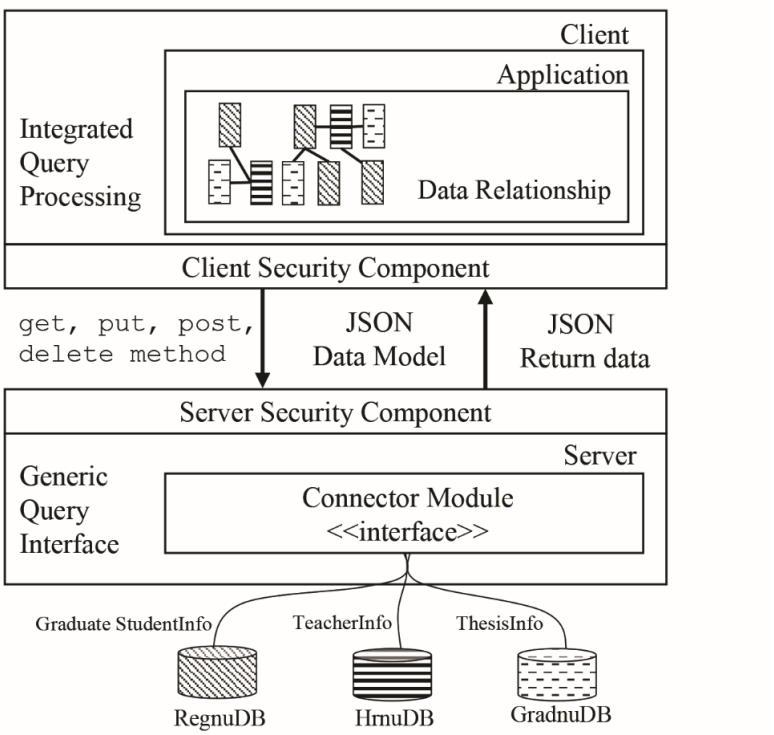

Figure 2 Data flow between the integrating provider and data provider.

Backend design for providing data: The Connector Component, consisting of connection functions was implemented as a general interface driver to access the database and query function in manageable data. The connection function is responsible for translating logic and accessing the database. The query function is responsible for exposing "`get`, `put`, `get`, and `delete`" operations. The main operations are defined according to the principle of REST. Generic interface with HTTP methods was used to design the interface for the operation and interaction of remote data source via logical resource. Since this interface is based on HTTP, it therefore makes for easy implementation of client application and is independent from the need of special library frameworks. The backend design consists of 3 processes as follows:

- Design connector module for accessing the database: able to be used to run SQL commands directly at the resource without DBMS of database vendor using JavaScript query. In TCS, the driver was implemented in order to access 2 database vendors, including Oracle and MySQL. Each of the databases was designed for a basic queries module in manipulating information over the database, which included "read (filter on one field, find all documents, and find by condition), insert, update, and delete."

- Design Resource Interface by HTTP request: the Data Provider Interface was designed in 3 parts: the graduate student registration interface (Regnu), the thesis interface (Gradnu), and the faculty interface (Hrnu), as shown in **Table 1**.

Table 1 Resource Interface for ICR implementation.

Method	Path	Basic database operation	Description	Database
GET	/{databaseName}/{tableName}	Read()	A query for search to match all data from relation database	RegnuDB, HrnuDB, GradDB
GET	/{databaseName}/{tableName}/{condition}	Read()	A query for selecting some data by condition	
PUT	/{databaseName}/{tableName}/update/{condition}	Update()	A query for updating by condition	
POST	/{databaseName}/{tableName}/create	Insert()	A query for creating row in relation database	
POST	/{databaseName}/{tableName}/delete/#id	Delete()	A query for deleting row in relation database	

- Design Data Modeling: to represent structure data. All data from the connection manager (a part of the bottom layer) was transformed to the structure of the common data model as JSON. The response of the query is given by generic interface in JSON format for passing the value to the next layer for developing the application. The datatype supported in the JSON schema consists of: string, boolean, numeric, object, and array. However, this implementation selected String as the type of communication over HTTP protocol. The information from the query result is represented as JSON objects containing the structure of key/value pairs for the model attributes/references, without information about type of the data (e.g., Integer, `INT` or variable character, `VARCHAR`) from the local resource as appeared in general relational databases, which defines table, row, and column. The column is key, and the value in the row is value. Keys are the name of the attribute/reference of the concept, and values are their textual representation in one of the datatypes. For attributes, the values are mapped according to the corresponding JSON supported data type or string when there is not a direct correspondence. When the attribute is multi-valued, its values are represented using the array datatype. For references, the value is the URI of the addressed resource within the server. Requested data is sent through the REST-API for querying corresponding data from the data source. Every request of query command is sent as query parameter message over to the HTTP protocol, as shown in **Figure 3**.

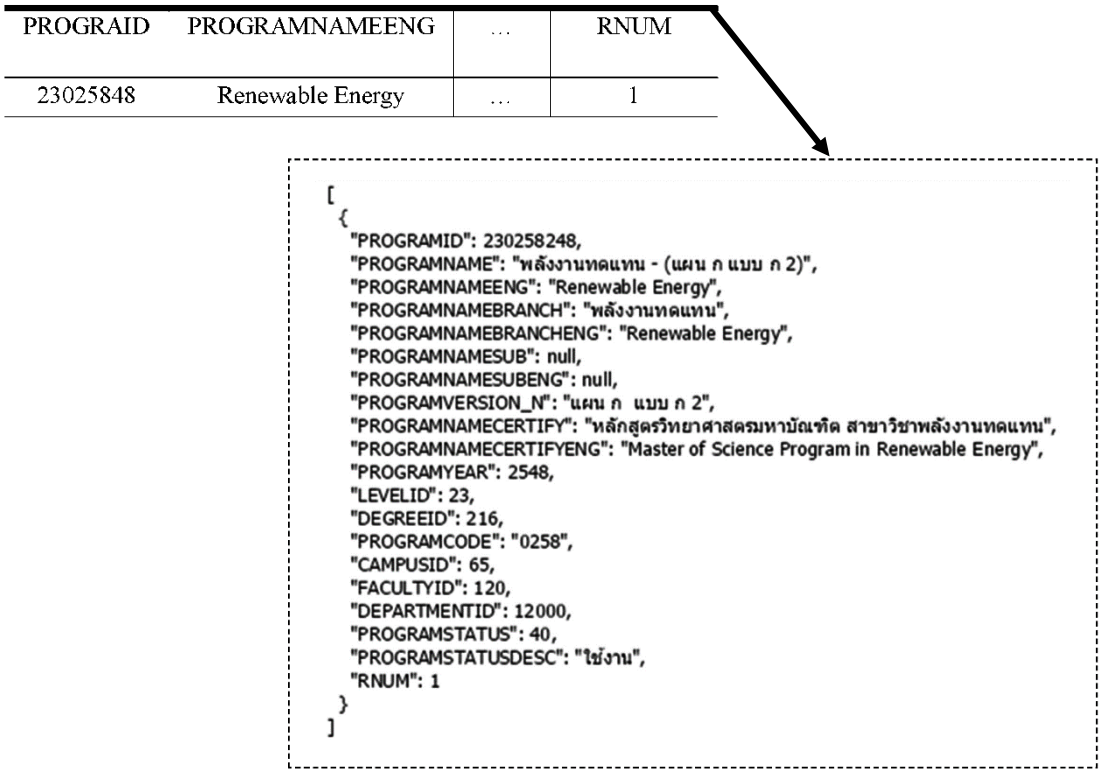

Figure 3 Transforming data source to JSON file for serialized data.

Frontend design for integrating data: For the frontend design for integrating data (the second step of TCS design), the data are passed by querying of connector module within the Data Provider layer into a JSON document. Then, it is used for defining data relationships in the business requirement. The metadata processing in the data integration system requires a common schema. Therefore, the JSON document is used to create a common understanding for different sources, representing, sharing, and reusing descriptions. JSON used in TCS are implemented in scripting language as JavaScript, which can be written in any programming language, especially supporting client-side language. It is formatted to the data content and structure without considering heterogeneous local schemas for each business domain. JSON data can be written in any programming language.

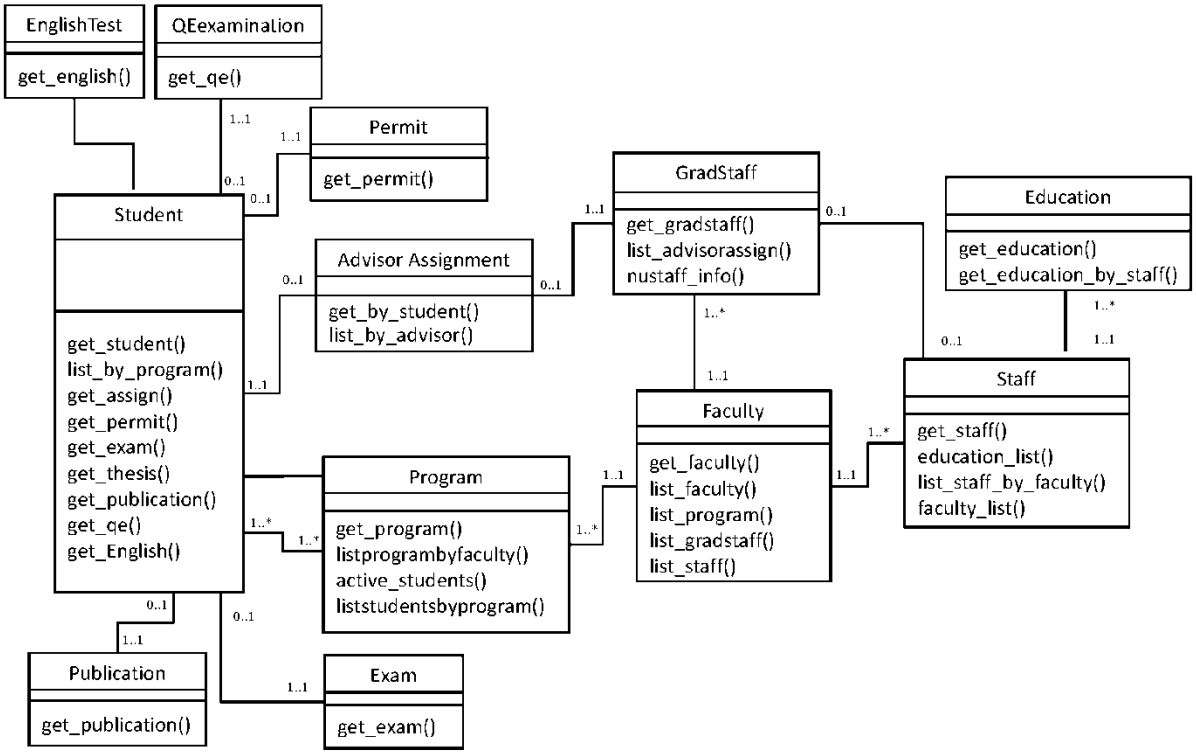

Figure 4 Data relationship of TCS.

Figure 4 represents the data relationship of TCS. The relationship is built from integrated logic as the conceptual model. A coding program of integrated query, processing integrated logic of TCS, was used as a client logic for mapping between JSON data stored on the client application. The advantage of this data is that it can be used to implement a platform independent from the client application.

System Requirement of TCS consists of 5 modules: *Login*, *Search Teacher*, *Management Publication Information*, *Filter Thesis Progress*, and *Display Thesis Progress Report*. In this paper, we show an example of 2 modules, *Filter Thesis Progress* and *Display Thesis Progress Report*, as illustrated in **Figure 5**.

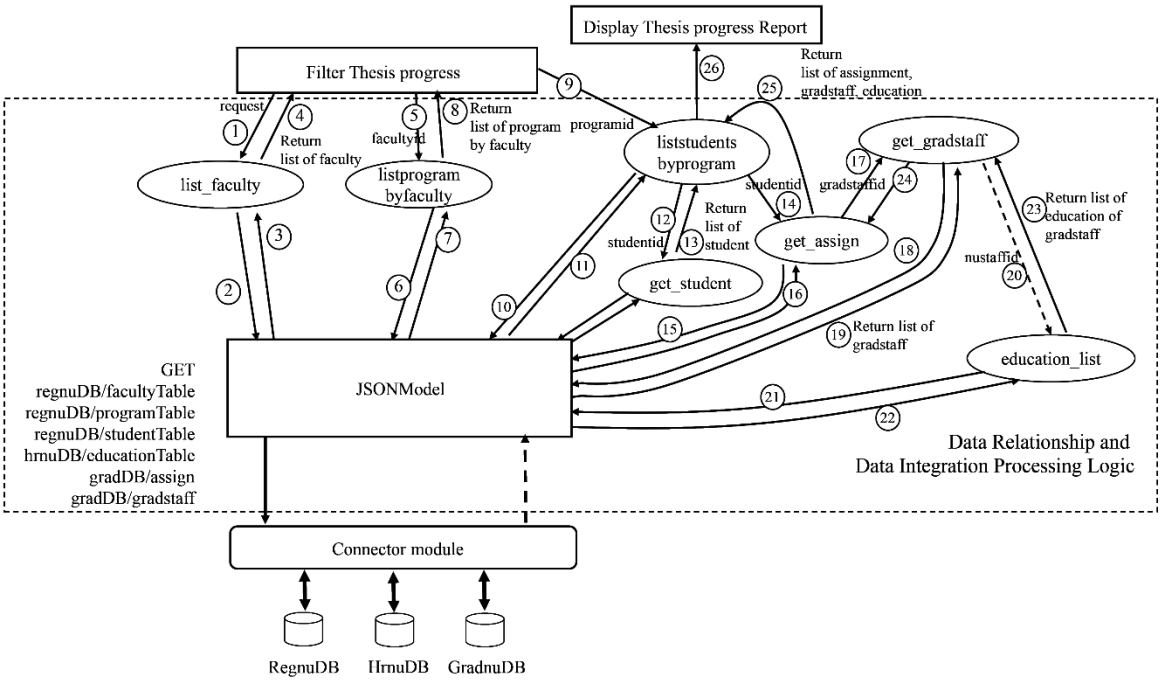

Figure 5 Example of processing of TCS in filter thesis process and display thesis process.

The third phase (Development Phase): the system prototype runs in the context of the JavaScript development environment, which is lightweight, for server-side development. Databases from 3 organizations were used: Graduate Student Registration Information from the Division of Academic Affairs with Oracle database, Teacher Information DB from the Personnel Management Division with Oracle database and Thesis Information from the Graduate School with MySQL database. The TCS application ran in the AngularJS framework inside the Node JS application server.

http://www.db.grad.nu.ac.th/core/**faculty**

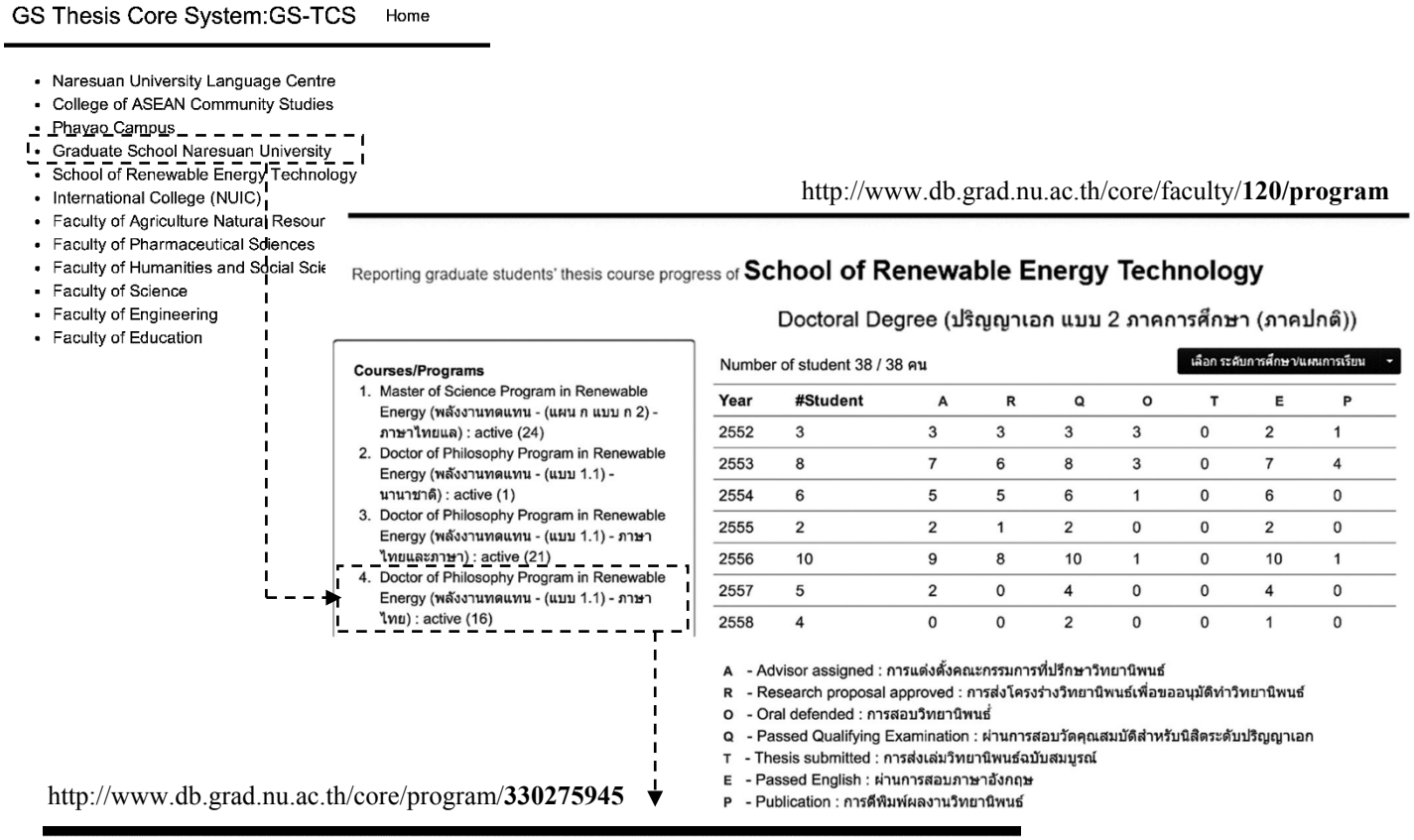

Figure 6 User interfaces of TCS.

Deployed Phase: as shown in **Figure 6**, this system prototype was deployed at the Graduate School, Naresuan University, for reporting graduate students' thesis course progress. The data was collected from the academic year 2009 to present. In the next section, the experiment and evaluated results are elaborated upon.

Empirical result and discussion

The proposed approach has demonstrated the capability of a distributed system for creating and running highly scalable web applications. For the performance aspect, we compared our approach with the traditional approach by implementing both systems with distributed data sources. The effect of an integrated task on the total average response time was evaluated when the number of concurrent users remote to the system was continually increasing.

Experimental Setup: the experiment environment consisted of 2 machines: (1) Client is an Intel Core i5-4200 1.60 GHz processor with 4 GBs RAM and (2) Server is a 3.20 GHz processor with 8 GBs RAM. A data package size of 10 k was used in the setup. The size mentioned in the paper (10 k) is the size of a message sent from server to client. It is not the size of the table in the database. The message size is a list of records that satisfied the condition. To measure the performance in a distributed computation, we needed to simulate a large number of client requests to perform a job on the provided system. We repeated the testing case 10 times and used the average response time for each testing as the response time for the testing case. The results included the integrated processing time, the communication time, and the latency time. In the experiment, JSON was chosen as the data format for both the traditional and the proposed method based on REST-based implementation. Three databases (2 Oracle, 1 MySQL) were in operation with client requests to the application. The response time was the difference in the time that the request was submitted to the server and the time that we received the integrated result. We simulated the client request starting from 100 until the system could not handle the request (i.e., system failure). For each step, we increased the number of requests by 100.

The result for each 2 test set is shown in **Figure 7**. Traditional Data Integration is normally only executed with an application logic on the server side to integrated data. The proposed approach has flexible options that can execute results with integrated logic on the client side. With the scalability objective, the testing focused on the performance effect of runtime integrated system between the server and client processing.

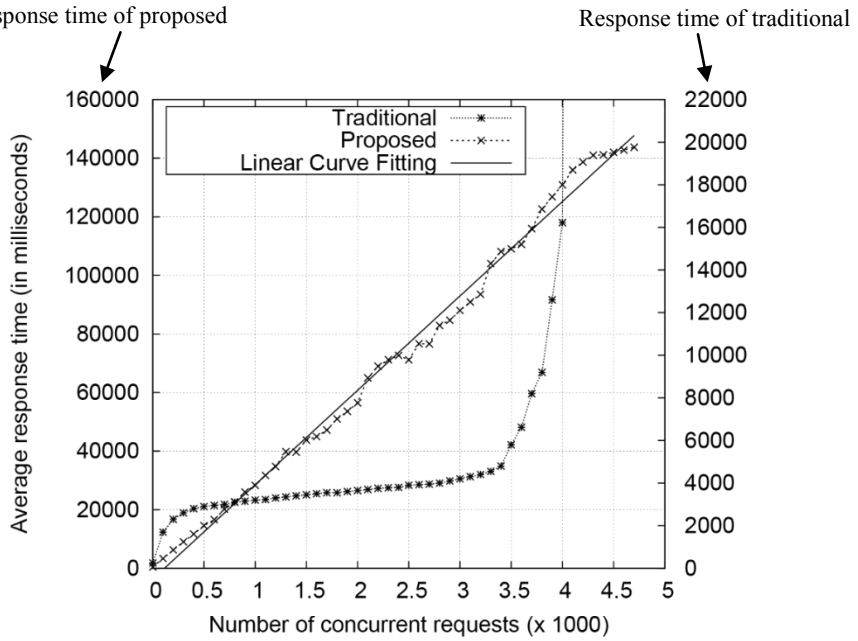

Figure 7 Comparing average response time of integrated processing between traditional and proposed approach.

As shown in **Figure 7**, the response time of the traditional (~16,000 ms) approach is better than the response time of the proposed (~130,000 ms) approach when the concurrent request is 4,000. However, the proposed approach can handle a larger number of the concurrent request than the traditional approach, as shown in **Figure 7**, where the traditional approach cannot handle a request of more than 4,000, while the proposed approach can handle the request. Moreover, as shown in **Figure 7**, the response time of the traditional approach is exponentially increasing when request is of more than 3,500, compared with polynomial increasing in response time for all numbers of request in the proposed approach. The overall results indicate that executing data integration process logic on the server side in the case of the traditional approach has affected the system's performance when the visiting scale is large.

Discussion

In this research, several key findings were revealed, as per the following: the results indicated that the proposed approach based on REST is better in scalability, coupling, and performance. The response time ratio is linear when a number of concurrent users is very large (the system remains stable with Limited Memory) with an increasing number of concurrent users, as per **Figure 7**. Therefore, the proposed approach is better at handling system growth with linearity than the approach from Meng [13]. In addition, the approach is more suitable and scalable than the work from Fuentes-Lorenzo and colleagues, which only addressed related issues to provide semantic mapping and query management on heterogeneous data sources using REST-based implementation [41]. REST-based implementation from our results showed the advantage of the developed data integration system that moves the data integration process logic from the server side. Total integrated results are sent back to the client for creating and combining the result by the client's own logic. Therefore, the server will only retrieve data from multiple data sources. The large amounts of data among transmission data are divided for reducing the large total data integration time [10] and lead to bottleneck problems on the server side. It offers growth for the number of users to the application without adversely affecting the system.

The data integration management task in the server with REST-based approach has several benefits. The clear advantage of the server is that it offers less processing time, leading to lower consumption, since REST directly uses HTTP protocol in data exchanges via HTTP Standard Features on the standard protocol. Designing an interface for each operation is not complex. It can be easy to maintain in the long term. The transferred data format is simple, which makes the client efficiently use metadata in the message used for data exchange. With statelessness interaction, each client request contains all of the application states necessary to independently understand logic between the server and the client. It does not keep the state information on the server; therefore, the cost of the system scaling is reduced.

Conclusions

We proposed an alternative data integration based on the REST approach for internet-based scale. To compare it, the Thesis Core System prototype was implemented in both directions (proposed and traditional). The experimental results showed that the proposed approach can outperform the traditional approach in term of scalability. The proposed approach can reduce the high cost of running data integration processes of the system task in the data source server, which is one of the important factors that causes high error risk. The advantage consists of distributed tasks for the data integration process. The contribution of this research delivers significant system performance enhancement to enable server capability to handle a large-scale workload. As a result, the system developed under the proposed approach can scale up to a large number of concurrent requests.

References

[1] P Budny, S Govindharaj and K Schwan. Worldtravel: A testbed for service-oriented applications. *In*: Proceedings of the 6th International Conference Sydney on Service-Oriented Computing 2008. Springer-Verlag, Sydney, Australia, 2008, p. 438-52.
[2] S Finkelstein. Common expression analysis in database applications. *In*: Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM Press, Orlando, Florida, USA, 1982, p. 235-45.
[3] JA Blakeley, PA Larson and FW Tompa. Efficiently updating materialized views. *In*: Proceedings of the ACM SIGMOD Conference on Management of Data. ACM Press, Washington DC, USA, 1986, p. 61-71.
[4] JJ Lu, G Moerkotte, J Schue and VS Subrahmanian. Efficient maintenance of materialized mediated views. *SIGMOD Record.* 1995; **24**, 340-51.
[5] B Ludäscher, Y Papakonstantinou and P Velikhov. Navigation-driven evaluation of virtual mediated views. *In*: Proceedings of 7th the International Conference on Extending Database Technology 2000. Springer-Verlag, London, UK, 2000, p. 150-65.
[6] A Silberschatz, M Stonebraker and JD Ullman. Database systems: Achievements and opportunities. *Commun. ACM* 1995; **34**, 110-20.
[7] L Zamboulis, N Martin and A Poulovassilis. Query performance evaluation of an architecture for fine-grained integration of heterogeneous grid data sources. *Future Gener. Comput. Syst.* 2010; **26**, 1073-91.
[8] O Rihawi, Y Secq and P Mathieu. Load-balancing for large scale situated agent-based simulations. *Proc. Comput. Sci.* 2015; **51**, 90-9.
[9] B Yildiz and GC Fox. Toward a modular and efficient distribution for Web service handlers. *Concurr. Comput. Pract. E* 2013; **25**, 410-26.
[10] J Kampars and J Grabis. An approach to parallelization of remote data integration tasks. *Sci. J. Tech. Univ.* 2011; **45**, 24-30.
[11] G Chen, Y Wu, J Liu, G Yang and W Zheng. Optimization of sub-query processing in distributed data integration systems. *J. Netw. Comput. Appl*. 2011; **34**, 1035-42.
[12] RT Fielding. 2000, Architectural Styles and the Design of Network-Based Software Architectures. Ph. D. Dissertation. University of California, Irvine.
[13] M Jian, M Shujun and Y Zhao. RESTful web services: A solution for distributed data integration. *In*: Proceedings of the International Conference on Computational Intelligence and Software Engineering. Wuhan, China, 2009, p. 1-4.
[14] S Zhang, F Zhang and B Wang. Data integration based on REST and IEC61970 for smart grid. *Electric Power Autom. Equip.* 2012; **32**, 124-9.
[15] X Feng, J Shen and Y Fan. REST: An alternative to RPC for web services architecture. *In*: Proceedings of the 1st International Conference on Future Information Networks 2009. Beijing China, p. 7-10.
[16] F Belqasmi, R Glitho and C Fu. RESTful web services for service provisioning in next-generation networks: A survey. *IEEE Comm. Mag.* 2011; **49**, 66-73.
[17] H Hamad, M Saad and R Abed. Performance evaluation of RESTful web services for mobile devices. *Int. Arab J. e-Technol.* 2010; **1**, 72-8.
[18] G Alonso, F Casati, H Kuno and V Machiraju. *Web Services - Concepts, Architectures and Applications*. *Data-Centric Systems and Applications*. 1st ed. Springer Publishing Company, New York, 2004.
[19] M Frehner and M Brandli. Virtual database: Spatial analysis in a Web-based data management system for distributed ecological data. *Environ. Model. Softw.* 2006; **21**, 1544-54.
[20] WW Eckerson. Three Tier Client/Server architecture: Achieving scalability, performance, and efficiency in client server applications. *Open Inf. Syst. J.* 1995; **3**, 10.
[21] A Tarhini. Concept of Three Tier Architecture. Available at: https://alitarhini. wordpress.com/ 2011/01/22/concepts-of-three-tier-architecture, accessed April 2014.

[22] S Ramani. A modern architecture for open source cloud computing. *Int. J. Adv. Res. Comput. Sci.* 2011; **2**, 95-8.
[23] H Zhou, Q Duan and Z Liang. A multiple-tier distributed data integration architecture. *In*: Proceedings of the 7th World Conference on Integrated Design and Process Technology. Texas, 2003, p. 153-9.
[24] P Vongsumedh and A Sukstrienwong. A framework for developing the Web-based data integration tool for Web-Oriented data warehousing. *In*: Proceedings of the 2013 International Conference on Systems, Control and Informatics. Greece, 2013, p. 130-5.
[25] C Cimen, Y Kavurucu and H Aydin. Usage of thin-client/server architecture in computer aided education. *Turk. Online J. Educ. Tech.* 2014; **13**, 181-5.
[26] A Mesbah and AV Deursen. A component- and push-based architectural style for Ajax applications. *J. Syst. Softw*. 2008; **81**, 2194-209.
[27] D Chen, R Chirkova, F Sadri and TJ Salo. Query optimization in information integration. *Acta Inform.* 2013; **50**, 257-87.
[28] M Karnstedt, KU Sattler and M Hauswirth. Scalable distributed indexing and query processing over Linked Data. *Web Semant.* 2012; **10**, 3-32.
[29] TW Kang and CH Hong. A study on software architecture for effective BIM/GIS-based facility management data integration. *Autom. Constr.* 2015; **54**, 25-38.
[30] A Datta and H Thomas. A conceptual model and algebra for on-line analytical processing in data warehouses. *In*: Proceedings of the 7th Workshop for Information technology and Systems. Citeseer, 1997, p. 91-100.
[31] S Lynden, A Mukherjee, AC Hume, AAA Fernandes, NW Paton, R Sakellariou and P Watson. The design and implementation of OGSA-DQP: A service-based distributed query processor. *Future Gener. Comp. Syst.* 2009; **25**, 224-36.
[32] C Pahl and ZHU Yaoling. Data integration in mediated service compositions. *Comput. Inform.* 2012; **31**, 1129-49.
[33] M Lablans, A Borg and F Uckert. A RESTful interface to pseudonymization services in modern web applications. *BMC Med. Inform. Decis.* 2015; **15**, 2.
[34] H Lee and MR Mehta. Defense against REST-based web service attacks for enterprise systems. *Comm. Int. Inform. Manag. Assoc.* 2013; **13**, 57-68.
[35] J Parra, MA Hossain, A Uribarren and E Jacob. RESTful discovery and eventing for service provisioning in assisted living environments. *Sensors* 2014; **14**, 9227-46.
[36] C Pautasso. RESTful Web service composition with BPEL for REST. *Data Knowl. Eng.* 2009; **68**, 851-66.
[37] A Dey, A Fekete and U Rohm. REST+T: Scalable transactions over HTTP. *In*: Proceedings of the 2015 IEEE International Conference on Cloud Engineering. Tempe Arizona, USA, 2015, p. 36-41.
[38] OF Luis. 2010, Design and Development of a REST based Web Service Platform for Applications Integration. Master in Artificial Intelligence. Universitat Politecnica de Catalunya, Barcelona, Spain.
[39] M Hourani, Q Shambour, A Al-Zubidy and A Al-Smadi. Proposed design and implementation for RESTful web server. *J. Softw.* 2014; **9**, 1071-80.
[40] JC Delgado. Architectural styles for distributed interoperability. *Inform. Resour. Manag. J.* 2013; **26**, 40-65.
[41] D Fuentes-Lorenzo, L Sanchez, A Cuadra and M Cutanda. A RESTful and semantic framework for data integration. *Softw. Pract. Ex*. 2015; **45**, 1161-88.
[42] Y Yuan, Y Wu, X Feng, J Li, G Yang and W Zheng. VDB-MR: MapReduce-based distributed data integration using virtual database. *Future Gener. Comput. Syst.* 2010; **26**, 1418-25.
[43] E Platen and D Heath. *A Benchmark Approach to Quantitative Finance.* 2nd ed. Springer-Verlag, Berlin, Germany, 2010, p. 1-720.