ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

Performance of Distributed Database System built on Multicore Systems

Kangseok Kim[1, 2]

ABSTRACT

Recently, huge datasets have been generating rapidly in a variety of fields. Then, there is an urgent need for technologies that will allow efficient and effective processing of huge datasets. Therefore the problems of partitioning a huge dataset effectively and alleviating the processing overhead of the partitioned data efficiently have been a critical factor for scalability and performance in distributed database system. In our work we utilized multicore servers to provide scalable service to our distributed system. The partitioning of database over multicore servers have emerged from a need for new architectural design of distributed database system from scalability and performance concerns in today's data deluge. The system allows uniform access through a web service interface to concurrently distributed databases over multicore servers, using SQMD (Single Query Multiple Database) mechanism based on publish/subscribe paradigm. We will present performance results with the distributed database system built on multicore server, which is time intensive with traditional architectures. We will also discuss future works.

☞ keyword : Distributed Database System, Data Clustering, Web Service, Multicore

1. Introduction

Recently, huge datasets have been generating routinely and rapidly in a variety of fields (environmental sensors, Internet data, and so on) [1]. Given this deluge of data, there is a need of distributed database system for processing huge datasets efficiently and effectively. Therefore the problems of partitioning a huge dataset effectively and alleviating the processing overhead of the partitioned data efficiently have been a critical factor for scalability and performance in distributeddatabase system. To achieve scalability and high performance, we developed a distributed database system built on multicore servers. The databases are distributed over distinct multicore servers by fragmenting data using data clustering method with deterministic annealing to increase a molecule shape similarity and horizontal partitioning method to decrease a query processing time. In our work the use of the multicore can allow users to access datasets and to use servers simultaneously in anytime and in anywhere. We also used a Single Query Multiple Database (SQMD) mechanism [2] which was developed for building a scalable, distributed database system using virtualization technology based on OpenVZ in our previous work. The mechanism transmits a query that is operated simultaneously on all the databases via middleware and agents using a publish/subscribe paradigm. The web service component aggregates the responses of the individual databases. In this paper we fundamentally focus onhigh performance interaction between users and huge datasets with a scalable, distributed database system built on multicore servers.

The remainder of this paper is organized as follows. We describe related works in Section 2. The architecture of the scalable, distributed database system built on multicore systems is presented in Section 3. Section 4 presents experimental results to demonstrate the viability of the distributed database system. Finally this paper is concluded with future research directions in Section 5.

2. Related works

For data scalability, researchers showed that a database can be scaled across distributed sites by using such fragmentation methods as vertical, horizontal, hash, range, list partitioning, and so on [3, 4, 5, 6]. On the other hand, we address the partitioning problem of a database over multicore servers. The partitioning is based on data clustering

1 Dept. of Cyber Security, Ajou University, Suwon, 16499, Korea
2 Dept. of Data Science, Ajou University, Suwon, 16499, Korea
* Corresponding author: kangskim@ajou.ac.kr
[Received 26 May 2017, Reviewed 30 May 2017, Accepted 1 September 2017]

for data similarity and horizontal partitioning. We performed the clustering using deterministic annealing algorithms [7] developed by SALSA project [8]. Also in our work we utilized multicore servers to provide scalable service to our distributed system. The partitioning of database over multicore servers have emerged from a need for new architectural design of distributed database system from scalability and performance concerns in today's data deluge. Different middleware systems such as OGSA-DAI [9], OGSA-DQP [10], and Open-Gate [11] for distributed database systems have been proposed. The middleware systems provide a uniform access interface to distributed systems. However, the middleware systems did not address on taking advantage of the benefits provided by using multicore technology. In this paper we present our experience of using publish/subscribe for integrating query results from distributed database servers over multicore systems. The main objective is to provide an efficient distributed database system with the characteristics of scalability and high performance using data partitioning and multicore technologies.

This paper proposes a new middleware system architecture that takes advantage of the benefits provided by using multicore technology, which is not addressed in the previous papers.

3. Architecture for distributed database system built on multicore systems

Figure 1 shows a broad architecture view for distributed database system built on multicore systems. The system is composed of three tiers – web service client, a web service and message service system, agents and a collection of databases. The distributed database system is composed of two or more PostgreSQL [12] databases that reside on one or more multicore servers.

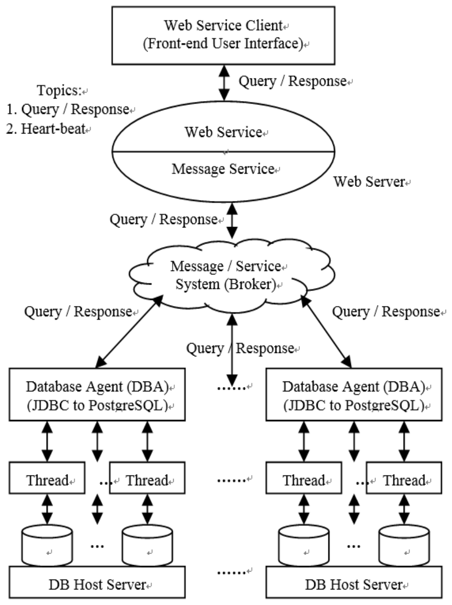
The middleware interacts with a web service and database agents which run on multicore machines. The web service acts as a query service manager and a result aggregation service manager for heterogeneous web service clients. We describe them in the following subsections.

3.1 Web service client

A query from clients are disseminated through the message and service system to database servers through database agents. Web service clients can simultaneously access the data in all the databases in a distributed environment.

3.2 Message / Service middleware

We have used a message and service middleware system [13] which supports publish/subscribe messaging paradigm as a middleware. The message and service system provides a mechanism for simultaneously disseminating queries and retrieving the results of the queries to and from distributed databases.



(Figure 1) An overall architecture view

3.3 Database agent

The agent accepts query requests from users, transfers the requests to database server and retrieves the results from the server. The agent performs concatenations of responses occurred from database. The agent has communication interfaces for offloading computational needs. Also the agent generates multiple threads associated with multiple database.

3.4 Database server

A number of data partitions split are distributed into database servers using PostgreSQL. The partitioned data is assigned to a database. The database is associated with a thread generated by database agent. Multiple threads based on the number of cores supported by multicore servers can be generated to maximize high performance service.

4. Performance analysis

In our experiment, databases are distributed over eight distinct multicore servers by using two different data fragmentation methods: deterministic annealing clustering and horizontal partitioning. First, we show the latency incurred from an interaction between a client and a centralized database. Then we show query processing cost in time among distributed databases. The horizontal partitioning method was chosen due to easy-to-split and easy-to-use factors. In our experiments we used an example query shown in Figure 2. The distance R in the figure means a cutoff to retrieve those points to the query point from the database. Table 1 shows the total number of hits obtained from varying R using the query. In Section 4.1 we show overhead time incurred from processing a query in the distributed database system. Section 4.2 presents the performance results for query processing in a centralized database. Section 4.3 presents the performance results from interactions between a client and distributed databases.

```
select * from (select cid, momsim, 1.0 / (1.0 +
cube_distance ( ('3.0532197952271,
1.0399824380875, -0.092431426048279,
3.0814106464386, 1.0752420425415,
-0.49167355895042, 5.3552670478821,
5.1984167098999, -0.41230815649033,
4.9449820518494, 4.9576578140259,
-0.093842931091785') ::cube, momsim)) as sim from
pubchem_3d where cube_enlarge (('3.0532197952271,
1.0399824380875, -0.092431426048279,
3.0814106464386, 1.0752420425415,
-0.49167355895042, 5.3552670478821,
5.1984167098999, -0.41230815649033,
4.9449820518494, 4.9576578140259,
-0.093842931091785'), R, 12) @> momsim order by
sim desc ) as foo where foo.sim != 1.0;
```

(Figure 2) An example query

(Table 1) The total number of response data

Distance R	# of Hits	Size in Bytes
0.3	495	80,837
0.4	6,870	1,121,181
0.5	37,049	6,043,337
0.6	113,123	18,447,438
0.7	247,171	40,302,297

4.1 Overhead timing considerations

The cost in time to access data from databases distributed over multicore servers has the following overheads.

- Network cost (Tclient2ws) – The time to send a query and receive a response to and from web service.
- Web service cost (Tws2db) – The time between sending a query from a web service component to all the databases and retrieving the responses from all the databases.
- Aggregation cost (Taggregation) – The time spent for aggregating responses in the web service.
- Query Processing cost (Database agent cost) (Tagent2db) – The time between submitting a query from an agent to a database server and retrieving responses from the database server.

4.2 Query processing cost in a centralized database

In the experiments we show the round trip latency cost incurred from processing a query with a centralized database in performing queries.

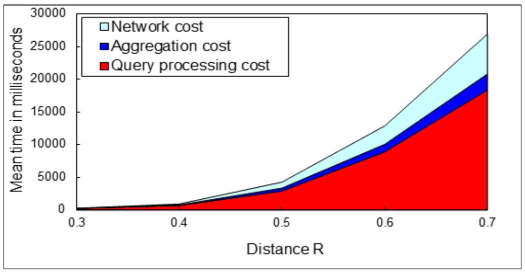
Table 2 shows the experimental environments. The experiment results were measured from executing a web service client running on Windows platform connected to Ethernet network, and executing a web service and a message/service middleware running on Windows platform connected to Ethernet network.

(Table 2) Experimental environments

Components	Specification
Web service client	Windows platform with 3.40 GHz and 1 GB RAM connected to Ethernet network
Message/service middleware	Windows platform with 3.40 GHz and 2 GB RAM connected to Ethernet network
Agents and database servers	eight 2.33 GHz Linux with 8 core / 8 GB RAM connected to Ethernet network

Agents and database servers ran on each of eight cores connected to Ethernet network as well.

Figure 3 shows the mean completion time taken by transmitting a query and by receiving a response between a client and a database server, varying the distance R. As the distance R increases, the size of responses also increases, as shown in Table 1. Therefore when the distance R increases, the query processing cost in the database increases as shown in Figure 3. Thus, by making the performance degrading factor (query processing cost) faster, we can reduce the total cost. The result will be used as a baseline for the experiments performed in the following section.



(Figure 3) Mean query response time in a centralized (not fragmented database), varying R

4.3 Query processing cost in distributed databases over multicore servers

The database is split into eight partitions by deterministic annealing data clustering method and horizontal partitioning method. Each of partitions is distributed across eight multicore servers. Table 3 shows the size of the partitioned data in number.

We measured the mean overhead cost about three different test cases with two different fragmentation methods: deterministic annealing based data clustering vs. horizontal partitioning vs. combined partitioning of deterministic annealing based data clustering and horizontal partitioning by varying R with the example query. In Table 4 the results are summarized with the mean completion time of a request in the considerations of overhead timings between a client and databases.

(Table 3) The fragmented data size (in number) by clustering with deterministic annealing (Note that each of fragmented data size (in number) by horizontal partitioning method has about 2,154,000 dataset)

Segment number	Dataset size in number	Segment number	Dataset size in number
1	6,204,776	5	2,302,272
2	616,133	6	4,634,860
3	507,209	7	785,232
4	2,018,281	8	163,017

(Table 4) The number of responses occurred with varying R on segments (segment number (S), data clustering with deterministic annealing (D), and horizontal partitioning (H))

	S	1	2	3	4	5	6	7	8
	R								
D	0.3	1	0	0	0	494	0	0	0
	0.4	87	0	30	0	6,753	0	0	0
	0.5	1,868	0	570	0	34,611	0	0	0
	0.6	12,926	0	2,720	0	97,477	0	0	0
	0.7	44,388	0	6,571	0	196,212	0	0	0
H	0.3	75	82	77	62	45	27	49	78
	0.4	863	1,133	978	893	667	498	780	1,058
	0.5	4,667	5,686	5,279	4,746	3,615	3,031	4,361	5,664
	0.6	14,089	16,749	15,782	14,650	11,369	9,756	13,559	17,169
	0.7	30,920	35,558	33,862	32,277	25,207	22,268	29,620	37,459

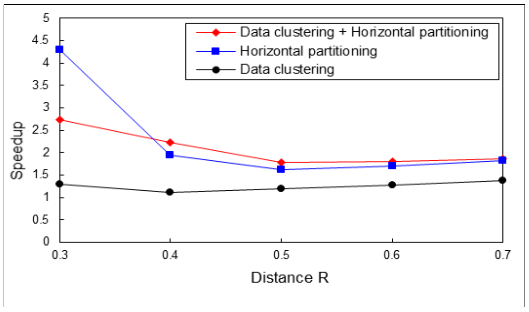
By comparing the total costs, the speedup is follows:

$$Speedup\ (1) = T_{total(1db)} / T_{total(8db)}$$
$$= (T_{client2ws(1db)}+T_{ws2db(1db)})/(T_{client2ws(8db)}+T_{ws2db(8db)})$$
$$Speedup\ (2) = 1 / ((1 - (T_{agent2db\ (1db)} / T_{total\ (1db)})) + ((T_{agent2db\ (1db)} / T_{total\ (1db)}) / (T_{agent2db\ (1db)} / T_{agent2db\ (8db)})))$$

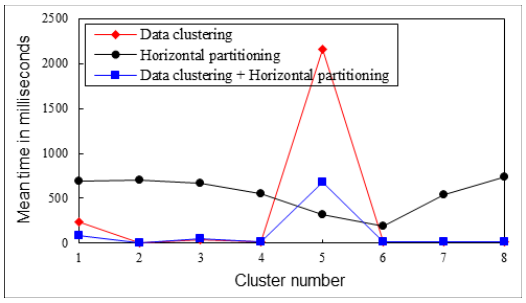
where (*1db*) means a centralized database and (*8db*) means a distributed database.

Speedup (1) means that the value of speedup is the mean query response time in a centralized database system over the mean query response time in a distributed database system. *Speedup (2)* means that the speedup gained by incorporating the enhanced and un-enhanced portions respectively.

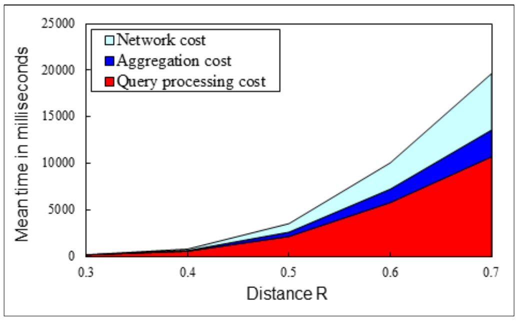
Figure 4 shows the speedup obtained by applying Speedup (1) to the three test cases respectively. For brevity, as an example, we explain the overall speedup in case which the distance is 0.5. In case of horizontal partitioning, the overall speedup by the equation (1) is 1.62. The speedup by



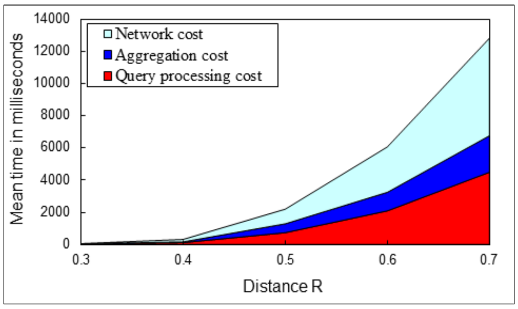
(Figure 4) Speedup according to distance R



(Figure 5) Mean query processing time in each cluster (R=0.5)



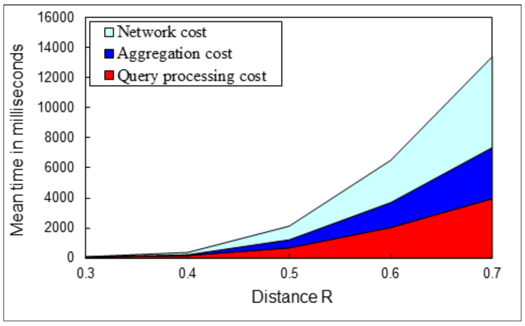
(Figure 6) Mean query response time in databases distributed by data clustering based on deterministic annealing, varying R



(Figure 7) Mean query response time in databases distributed by horizontal partitioning, varying R

the equation (2) is 1.93. The difference comes from some additional overheads incurred during the query/response interaction. We measured the duration between first response message and last response message replied from agents, and the duration between first response message and last response message arriving into web service component for global aggregation of responses. There was a difference between the durations. It is due to two overheads: network overhead and global aggregation overhead. The network overhead happened between web service component and agents. The global aggregation overhead happened since the web service component has to wait all the responses until all database servers send the response messages. From the results with the example query in the distributed database system, using horizontal partitioning method is faster than using deterministic annealing based data clustering since fragments partitioned by similarities in the data clustering can be different in the size of data. For example, as the case of cluster number 5 in the graph of Figure 5, when the number of responses occurred in performing a query in the cluster number 5 increase, the time needed to perform the query in the cluster increases as well. But the responses hit by a query may be occurred in only several distributed databases, then the deterministic annealingbased data clustering will benefit more, with a need of data locality while resulting in high latency. Therefore, from the experimental results we identified the problems, data locality and latency. To reduce the latency with increasing data locality in the deterministic annealing based data clustering, we combined the deterministic annealing based data clustering with the horizontal partitioning.

Figures 6, 7, and 8 show the experimental results with deterministic annealing based data clustering, horizontal partitioning, and the combination respectively. Compare the query processing time in Figure 6 with that in Figure 7, and with Table 4. The query processing cost becomes a smaller portion of overall cost than the transit cost as shown in Figure 8. This result shows that the distributed database system is scalable with the partitioning of database over multicore servers by deterministic annealing based data clustering for increasing data locality, and by horizontal partitioning in each cluster for decreasing query processing cost. Thus the system improves overall performance as well as query processing performance.



(Figure 8) Mean query response time in databases distributed by data clustering based on deterministic annealing and horizontal partitioning, varying R

5. Conclusions

We presented a scalable, distributed database system using SQMD mechanism based on a publish/subscribe paradigm over multicore servers. Also we described about dataset partitioning problem over multicore servers for scalability and performance with our architectural design. The experimental results show that the distributed database system built on multicore servers is scalable with the partitioning of database by deterministic annealing based data clustering for increasing data locality, and with multithreads of executions associated with multiple databases split by horizontal partitioning in each cluster for decreasing query processing cost.

In future work we will apply our proposed system architecture to building on distributed database systems based on heterogeneous data sources.

References

[1] Tony Hey and Anne Trefethen, "The data deluge: an e-Science perspective in Grid Computing: Making the Global Infrastructure a Reality" edited by Fran Berman, Geoffrey Fox and Tony Hey, John Wiley & Sons, Chicester, England, ISBN 978-0-470-85319-1, 2003. https://doi.org/10.1002/0470867167.ch36

[2] K. Kim, R. Guha, and M.E. Pierce, "SQMD: Architecture for Scalable, Distributed Database System Built on Virtual Private Servers", Fourth IEEE International Conference on eScience, pp. 658–665, 2008. https://doi.org/10.1109/eScience.2008.35

[3] IBM DB2, https://www.toadworld.com/platforms/ibmdb2/w/wiki/7341.table-partitioning-overview

[4] MySQL Forums, 2016. http://forums.mysql.com/

[5] Oracle Partitioning with Oracle Database 12c Release 2, Oracle White Paper, 2017. http://www.oracle.com/technetwork/database/options/partitioning/partitioning-wp-12c-1896137.pdf

[6] PostgreSQL Partitioning, https://www.postgresql.org/docs/current/static/ddl-partitioning.html

[7] Qiu, X., Fox, G., Yuan, H., Bae, S., Chrysanthakopoulos, G., Nielsen, H. F., "Performance of Multicore Systems on Parallel Data Clustering with Deterministic Annealing", ICCS 2008: Lecture Notes in Computer Science Vol. 5101, pp. 407-416, 2008. https://doi.org/10.1007/978-3-540-69384-0_46

[8] SALSA (Service Aggregated Linked Sequential Activities), http://salsahpc.indiana.edu/

[9] Xuhong Liu, Yunmei Shi, Yabin Xu, Yingai Tian, Fuheng Liu, "Heterogeneous Database Integration of EPR System Based on OGSA-DAI", in High Performance Computing and Applications LNCS, Vol. 5938, pp. 257-263, 2010. https://doi.org/10.1007/978-3-642-11842-5_35

[10] Helen X. Xiang, "Integrated Queries over a Heterogeneously Distributed Scientific Database using OGSA-DQP", in proceedings of the 6th IEEE Joint International Information Technology and Artificial Intelligence Conference (ITAIC), pp. 421-425, Chongqing,

2011. DOI: 10.1109/ITAIC.2011.6030237

[11] Naglaa M. Reda and Fayed F. M. Ghaleb, "Open-Gate: An Efficient Middleware System for Heterogeneous Distributed Databases", International Journal of Computer Applications, Vol. 45, No. 2, pp. 44-49, 2012. https://doi.org/10.5120/6755-9009

[12] PostgreSQL, http://www.postgresql.org/

[13] S. Pallickara, G. Fox and H. Gadgil, "On the Creation & Discovery of Topics in Distributed Publish/Subscribe systems", Proceedings of the IEEE/ACM GRID 2005 Workshop, pp. 25-32, 2005. https://doi.org/10.1109/GRID.2005.1542720

◐ 저 자 소 개 ◑

Kangseok Kim

Kangseok Kim received Ph.D. in Computer Science from Indiana University at Bloomington, IN, USA. He is currently an associate professor of Cyber Security department and Data Science department at Ajou University, Suwon, Korea. His main research interests include ubiquitous computing, cloud computing, IoT/Smartphone grid, bioinformatics and applied security in big data.

Email: kangskim@ajou.ac.kr