Performance of scalable, distributed database system built on multicore systems with deterministic annealing clustering

Kangseok Kim[a,b,∗,1], Marlon Pierce[b], Geoffrey Fox[a,b]

[a]*Computer Science Department, Indiana University, Bloomington, IN 47404, USA*
[b]*Community Grids Laboratory, Indiana University, Bloomington, IN 47404, USA*

Abstract Many scientific fields routinely generate huge datasets. In many cases, these datasets are not static but rapidly grow in size. Handling these types of datasets, as well as allowing sophisticated queries necessitates efficient distributed database systems that allow geographically dispersed users to access resources and to use machines simultaneously in anytime and anywhere. In this paper we present the architecture, implementation and performance analysis of a scalable, distributed database system built on multicore systems. The system architecture makes use of software partitioning of the database based on data clustering with deterministic annealing, termed the SQMD (Single Query Multiple Database) mechanism, a web service interface and multicore server technologies. The system allows uniform access to concurrently distributed databases over multicore servers, using the SQMD mechanism based on the publish/subscribe paradigm. We highlight the scalability of our software and hardware architecture by applying it to a database of 17 million chemical structures. In addition to simple identifier based retrieval, we will present performance results for shape similarity queries, which is extremely, time intensive with traditional architectures.

Keywords Distributed database system; Data clustering; Web service; Multicore Performance

1. INTRODUCTION

In the last few years, we have witnessed a huge increase in the size of datasets in a variety of fields (scientific observations for e-Science, environmental sensors, data fetched from Internet, and so on) (Hey, 2003; SALSA, 2008). This trend is expected to continue and future datasets will only become larger. Given this deluge of data, there is an urgent need for technologies that will allow efficient and effective processing of huge datasets. With the maturation of a variety of computing paradigms and with the advance of a variety of hardware technologies such as multicore servers, we can now start addressing the problem of allowing geographically dispersed users to access resources and to use machines simultaneously in anytime and anywhere.

The problems of effectively partitioning a huge dataset and of efficiently alleviating too much computing for the processing of the partitioned data have been critical factor for scalability and performance. In today's data deluge the problems are becoming common and will become more common in near future. The principle "Make common case fast" (or "Amdahl's law") (Hennessy, 1995) can be applied to make the common case faster since the impact on making the common case faster may be higher, while the principle generally applies for the design of computer architecture.

∗ Corresponding author. Tel.: +82 10 6223 3881.
[1]Present address: Department of Knowledge Information Security, Graduate School of Ajou University, Suwon, Korea
E-mail addresses: kangskim@ajou.ac.kr (K. Kim), marpierc@indiana.edu (M. Pierce), gcf@indiana.edu (G.Fox).

To achieve scalability and maintain high performance, we have developed a distributed database system on multicore servers. The databases are distributed over multiple, physically distinct multicore servers by fragmenting the data using two different methods: data clustering with deterministic annealing and horizontal partitioning to increase the molecule shape similarity and to decrease the query processing time. Each database operates with independent threads of execution over multicore servers. The distributed nature of the databases is transparent to end-users and thus the end-users are unaware of data fragmentation and distribution. The middleware hides the details about the data distribution. To support efficient queries, we used a Single Query Multiple Database (SQMD) mechanism which transmits a single query that simultaneously operates on multiple databases, using a publish/subscribe paradigm. A single query request from end-user is disseminated to all the databases via middleware and agents, and the same query is executed simultaneously by all the databases. The web service component of the middleware carries out a serial aggregation of the responses from the individual databases. Fundamentally, our goal is to allow high performance interaction between users and huge datasets by building a scalable, distributed database system using multicore servers. The multicore servers exhibit clear universal parallelism as many users can access and use machines simultaneously (Qiu, 2007). Obviously our distributed database system built on such multicore servers will be able to greatly increase query processing performance. In this paper we focus on the issue of data scalability with our software architecture and hardware technology.

This paper is organized as follows. Section 2 presents the problem and our general approach to a solution. We discuss related work in Section 3. Section 4 presents the architecture of our scalable, distributed database system built on multicore systems and briefly describes a database of 3D chemical structures, which we use as a case study for our architecture. Section 5 presents experimental results to demonstrate the viability of our distributed database system. Finally we conclude with a brief discussion of future work and summarize our findings.

2. PROBLEM STATEMENT

With the advances in a variety of software/hardware technologies and wire/wireless networking, coupled with large end-user populations, traditionally tightly-coupled client-server systems have evolved to loosely-coupled three-tier systems as a solution for scalability and performance. The workload of the server in two-tier system has been offloaded into the middleware in three-tier system in considering bottlenecks incurred from: increased number of service requests/responses, increased size of service payload, and so on. Also with the explosion of information and data, and the rapid evolution of Internet, centralized data have been distributed into locally or geographically dispersed sites in considering such bottleneck as increased workload of database servers. But in today's data deluge, too much computing for the processing of too much data leads to the necessity of effective data fragmentation and efficient service processing task. One solution to the problem is to effectively partition large databases into smaller databases. The individual databases can then be distributed over a network of multicore servers. The partitioning of the database over multicore servers can be a critical factor for scalability and performance. The purpose of the multicore's use is to facilitate concurrent access to individual databases residing on the database server with independent threads of execution for decreasing the service processing time. We have already encountered the scalability problems with a single huge dataset – a collection of 3D chemical structures (Chembiogrid, 2006) during our research work. We believe that the software and hardware architecture described in Section 4 will allow for effective data fragmentation and efficient service processing resulting in a scalable solution.

3. RELATED WORK

The middleware in a three-tier distributed database system also has a number of bottlenecks with respect to scalability. We do not address the scalability issues for middleware in this paper since our system can be scaled well in size by a cluster (or network) of cooperating brokers (Uyar, 2006; Gadgil, 2006). In this paper we focus on the issue related on data scalability. For data scalability, other researchers showed a database can be scaled across locally or geographically dispersed sites by using such fragmentation methods as vertical partitioning, horizontal partitioning, heuristic GFF (Greedy with First-Fit) (Sacca, 1985), hash partitioning (Baru, 1995), range/list/hash partitioning (Baer, 2007), and so on. On the other hand, we address the problem of partitioning a database over multicore servers, based on data clustering (Data Clustering, 2007) such that intra-cluster similarity (using the Euclidean metric) is greater than inter-cluster similarity. We performed the clustering using deterministic annealing algorithms developed by the SALSA project (SALSA, 2008) at the CGL (CGL, 2001). The details of the deterministic annealing clustering method are described in (Qiu, 2007). Also in our work we utilized multicore servers which enable multithreading of executions to provide scalable service to our distributed system. The utilization of the hardware device to aid data scalability was not addressed yet. The partitioning of the database over the multicore servers have emerged from a necessity for the new architectural design of the distributed database system from scalability and performance concerns against coming data deluge. Our architecture is similar in concept to that of SIMD (Single Instruction stream, Multiple Data stream) (Kumar, 2003), in that a single unit dispatches instructions to each processing unit. The SQMD uses the data parallelism in a manner similar to that of SIMD, via a publish/subscribe mechanism. In this paper we discuss data scalability in the distributed database system with the software and hardware architecture, using a collection of more 17 million 3D chemical structures.

4. ARCHITECTURE FOR SCALABLE DISTRIBUTED DATABASE SYSTEM BUILT ON MULTICORE SYSTEMS

Fig. 1 shows a broad 3-tier architecture view for our scalable distributed database system built on multicore systems. The scalable, distributed database system architecture is composed of three tiers – the web service client (front-end), a web service and message service system (middleware), agents and a collection of databases (back-end). The distributed database system is a network of two or more PostgreSQL (PostgreSQL, 2008) databases that reside on one or more multicore servers or machines. Our hardware (multicore machines) and software (web service, SQMD, and deterministic annealing clustering) architecture concentrates on increasing scalability with increased size of distributed data, providing high performance service with the enhancement of query/response interaction time, and improving data locality. In order to decrease the processing time and to improve the data locality of a query performed as the size of data increases, we used MPI (Message Passing Interface) (MPI, 1995) style multi-threading on a multicore machine by clustering data through clustering program developed by CGL and associating the clustered data with each of threads generated within the database agent. But the threads do not communicate with each other as MPI does. The multithreading within the database agent multitasks by concurrently running multiple databases, one on each thread associated with each core.

Our message and service system, which represents a middle component, provides a mechanism for simultaneously disseminating queries to and retrieving the results of the queries from distributed databases. The message and service system interacts with a web service which is another service component of the middleware, and database agents which run on multicore machines. The web service acts as query service manager and result aggregating service manager for heterogeneous web service clients. The database agent acts as a proxy for database server. We describe them in each aspect in the following subsections.

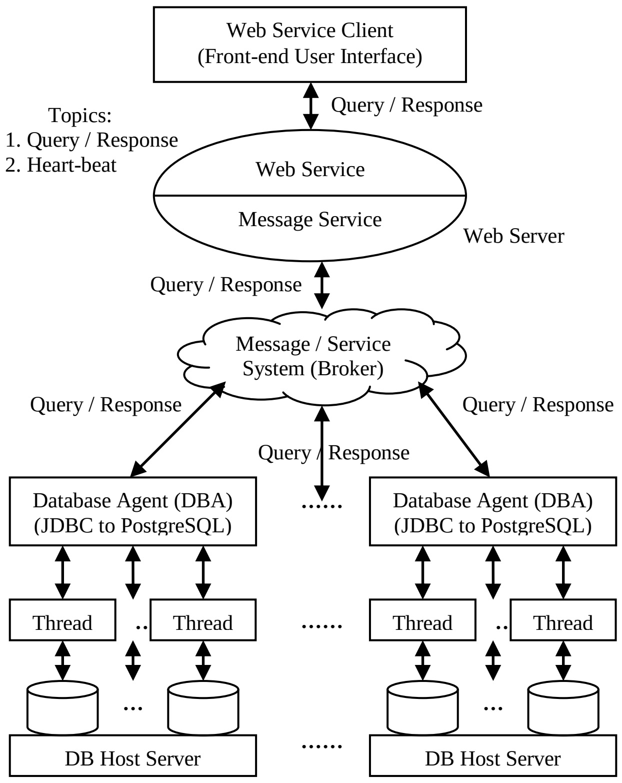

Fig. 1. Scalable, distributed database system architecture - three tiers: web service client, web service and broker, and agents and a collection of databases.

4.1. Web Service Client

Web service clients can simultaneously access the data in several databases in a distributed environment. Query requests from clients are transmitted to the web service, disseminated through the message and service system to database servers via database agents which reside on multicore servers. A web service client (front-end user interface) for Pub3D (Pub3d, 2008) service was developed by the ChemBioGrid project (Chembiogrid, 2006) at Indiana University.

4.2. Message and service middleware system

For communication service between the web service and middleware, and the middleware and database agents, we have used NaradaBrokering (Pallickara, 2005) for message and service middleware system as overlay built over heterogeneous networks to support communications among heterogeneous communities and collaborative applications. The NaradaBrokering from Community Grids Lab (CGL) is adapted as a general event brokering middleware, which supports publish/subscribe messaging model with a dynamic collection of brokers and provides services for Multicast. In this paper we use the terms “message and service middleware” and “broker” interchangeably.

4.3. Database Agent (DBA)

The DBA as a proxy for database server accepts query requests from front-end users via middleware, translates the requests to be understood by database server and retrieves the results from the database server. The retrieved results are presented (published) to the front-end user via a broker and web service. Web service clients interact with the DBA via middleware, and then the agent communicates with PostgreSQL database server. The agent has responsibility for getting responses from the database server and performs any necessary concatenations of responses occurred from database for the aggregating operation of the web service. As an intermediary between middleware and back-end, the agent retains communication interfaces and thus can offload some computational needs. Also the agent generates multiple threads which will be associated with multiple databases to improve query processing performance.

4.4. Database Server

A number of data partitions split by deterministic annealing clustering are distributed into PostgreSQL database servers. The partitioned data is assigned to a database which is associated with a thread generated by database agent. According to the number of cores supported by multicore servers, multiple threads can be generated to maximize high performance service.

4.5 Pub3D Database

PubChem is a public repository of chemical information. To access the chemical information or structure, one aspect that is not currently addressed by PubChem is the issue of 3D structures. Though a 2D representation is sufficient to understand the composition and connectivity of a molecule, many applications in chemoinformatics require that one has a 3D structure of a molecule. Furthermore, given a set of 3D structures one would then like to be able to search these structures for molecules whose 3D shape is similar to that of a query. To address the lack of 3D information in PubChem, to provide 3D shape searching capabilities and to allow efficient queries, Pub3D database employing a 12-D shape representation coupled with an R-tree (Guttman, 1984) spatial index was created. Then, given a 12-D point representation of a query molecular shape, we retrieve those points from the database whose distance to the query point is less than some distance cutoff, R.

5. PERFORMANCE ANALYSIS

In our experiment, we used deterministic annealing clustering software, developed by SALSA to partition a huge dataset. The deterministic annealing clustering algorithm is a modification of the K-means algorithm (K-means clustering), using deterministic annealing (Rose, 1998). Experimental results with the software show considerable gains for scalability and performance to cluster the 10 million chemicals in NIH PubChem and the 6 million people in the state of Indiana (Qiu, 2008). Databases are distributed over eight, physically distinct multicore servers by fragmenting the data using two different methods: deterministic annealing clustering and horizontal partitioning. Each database operates with independent threads (cores) of execution over multicore servers. The algorithm is described in more detail in (Qiu, 2007; 2008). First, in this section we show the latency incurred from query/response interaction between a web service client and a centralized Pub3D database via a middleware and an agent. Then we show the viability of our architectural approach to support efficient query processing in time among distributed databases into which the Pub3D database is split, with horizontal partitioning and data clustering based on deterministic annealing respectively. The horizontal partitioning in our experiments was chosen due to such convenience factors as easy-to-split and easy-to-use. In our experiments we used the example query shown in Fig. 2 as a function of the distance R from 0.3 to 0.7. The choice of the distance R between 0.3 and 0.7 was due to excessively small size of the result sets (0 hits for R=0.1 and 2 hits in R=0.2) for small values of R and the very large result sets, which exceeded the memory capacity (for the aggregation web service running on Windows XP platform with 2 GB RAM) caused by the large numbers of responses in the values bigger than 0.7. Table 1 shows the total number of hits for varying R, using the query of Fig. 2. In Section 5.1 we show overhead timing considerations incurred from processing a query in our distributed database system. In Section 5.2 we show the performance results for query processing task in a centralized database. In Section 5.3 we show the performance of a query/response interaction mechanism (SQMD using publish/subscribe mechanism) 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;
```

Fig. 2. An example query used in our experiment, varying R from 0.3 to 0.7, where the R means some distance cutoff to retrieve those points from the database whose distance to the query point.

Table 1
The total number of response data occurred with varying the distance R in the query of Fig. 2.

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

5.1. Overhead timing considerations

Fig. 3 shows a breakdown of the latency for processing SQMD operation between a client and databases in our distributed database system which is a network of eight PostgreSQL database servers that reside on eight multicore servers respectively. The cost in time to access data from the databases distributed over multicore servers has four primary overheads. The total latency is the sum of transit cost and web service cost.

- Transit cost ($T_{client2ws}$) – The time to transmit a query (T_{query}) to and receive a response ($T_{response}$) from the web service.
- Web service cost (T_{ws2db}) – The time between transmitting a query from a web service component to all the databases through a broker and agents and retrieving the query responses from all the databases including the corresponding execution times of the middleware and agents.
- Aggregation ($T_{aggregation}$) cost – The time spent in the web service for serially aggregating responses from databases.
- Database agent service cost ($T_{agent2db}$) – The time between submitting a query from an agent to and retrieving the responses of the query from a database server including the corresponding execution time of the agent.

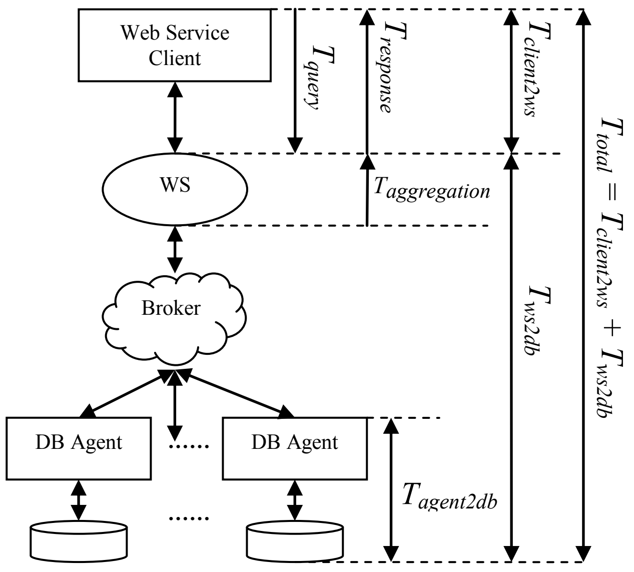

Fig. 3. Total latency (T_{total}) = Transit cost ($T_{client2ws}$) + Web service cost (T_{ws2db})

5.2. Performance for query processing task in a centralized database

In this section we show the performance results of latency incurred from processing a query between a web service client and a centralized database. Note that the results are not to show better performance enhancement but to quantify the performance for a variety of latencies induced with the centralized database. In our experiments, we measured the round trip time in latency involved in performing queries between a web service client and database host servers via middleware and database agents. The experiment results were measured from executing a web service client running on Windows XP platform with 3.40 GHz Intel Pentium and 2 GB RAM connected to Ethernet network, and executing a web service and a broker running on Windows XP platform with 3.40 GHz Intel Pentium and 2 GB RAM connected to Ethernet network. Agents and PostgreSQL database servers ran on each of eight 2.33 GHz Linux with 8 core / 8 GB RAM connected to Ethernet network as well.

Fig. 4 show the mean completion time to transmit a query and to receive a response between a web service client and a database host server including the corresponding execution time of the middleware and agents, varying the distance R described in Section 4.5. As the distance R increases, the size of result set also increases, as shown in Table 1. Therefore as the distance R increases, the time needed to perform a query in the database increases as well, which is shown in the figure and thus the query processing cost clearly becomes the biggest portion of the total cost. We can reduce the total cost by making the primary performance degrading factor ($T_{agent2db}$) faster. To make the primary degrading factor faster, the result which motivated our research work will be used as a baseline for the speedup measurement of the experiments performed in the following section.

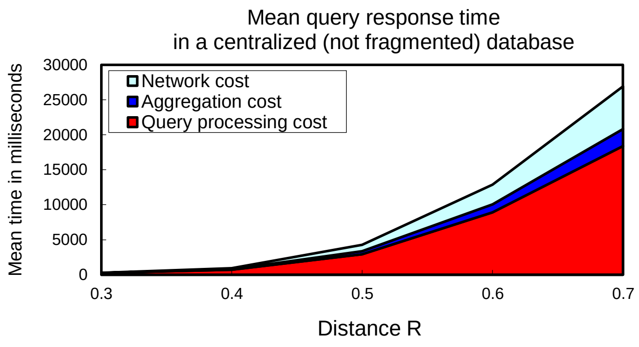

Fig. 4. Mean query response time between a web service client and a centralized database host server including the corresponding execution time of the middleware and agents, varying the distance R.

5.3. Performance for query processing task in distributed databases (Data clustering with deterministic annealing vs. Horizontal partitioning vs. Data clustering with deterministic annealing + Horizontal partitioning) over multicore servers

The Pub3D database is split into eight separate partitions by horizontal partitioning method and deterministic annealing data clustering method developed by SALSA project at CGL. Each of partitions of the database is distributed across eight multicore physical machines. Table 2 shows the partitioned data size in number by the data clustering based on deterministic annealing.

Table 2
The data size (in number) in the fragmentations into which the Pub3D database is split by clustering with deterministic annealing (Note that each database in the fragmentations by horizontal partitioning method has about 2,154,000 dataset in number)

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 3
The number of responses in segments occurred with varying the distance R, where S, D, and H mean segment number, data clustering with deterministic annealing, and horizontal partitioning respectively.

	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

Examining overhead costs and total cost, we measured the mean overhead cost for 100 query requests in our distributed database system. We tested three different cases with two different partitioning methods: data clustering with deterministic annealing vs. horizontal partitioning vs. data clustering with deterministic annealing and horizontal partitioning, varying the distance R in the example query which is shown in Fig. 2. The results are summarized in Table 3 with the mean completion time of a query request in the considerations of overhead timings between a client and databases.

By comparing the total costs for the three different cases with the total cost incurred from a centralized database system, we computed the speedup gained by the distribution of data with the use of multicore devices:

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

$$= 1 / ((1 - (T_{agent2db\,(1db)} / T_{total\,(1db)})) + ((T_{agent2db\,(1db)} / T_{total\,(1db)}) / (T_{agent2db\,(1db)} / T_{agent2db\,(8db)}))) \qquad (2)$$

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

(1) means 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. (2) means the speedup gained by incorporating the un-enhanced and enhanced portions respectively. Fig. 5 shows the overall speedup obtained by applying (1) to the test cases respectively. For brevity we explain the overall speedup with the distance 0.5 as an example. In case of using horizontal partitioning, the overall speedup by (1) is 1.62. The speedup by (2) is 1.93. This means some additional overheads were incurred during the query/response. We measured the duration between first and last response messages from agents, and that between first and last response messages arriving into web service component in middleware for global aggregation of responses. As expected, there was a difference between the durations. The difference is due to network overhead between web service component and agents, and the global aggregation operation overhead in web service that degrades the performance of the system since the web service has to wait, blocking the return of the response to a query request client until all database servers send the response messages. From the results with the example query in our distributed database system, using horizontal partitioning is faster than using data clustering with deterministic annealing since fragments partitioned by the data clustering can be different in the size of data as shown in Table 2. Then obviously as the responses occurred in performing a query in a large size of cluster increase, the time needed to perform the query in the cluster increases as well, which is shown in the graph in Fig. 6. But the responses hit by a query may not be occurred from all the distributed databases, then the data clustering will benefit more, increasing data locality while resulting in high latency. Therefore there may be unnecessary query processing with some databases distributed by the data clustering, using the SQMD mechanism as shown in Table 3. We thus identified the problems, data locality and latency, from our experimental results. To reduce the latency with increasing data locality in using the data clustering, we combined the deterministic annealing clustering with the horizontal partitioning to maximize the use of multicore with independent threads of execution by concurrently running multiple databases, one on each thread associated with each core in a multicore server. Figs. 7, 8, and 9 show the experimental results with deterministic annealing data clustering, horizontal partitioning, and the combination of both methods respectively. Our experimental results show there is a data locality vs. latency tradeoff. Compare the query processing time in Fig. 7 with that in Fig. 8, with Table 3. Also while the figures show that the query processing cost increases as the distance R increases, the cost becomes a smaller portion of overall cost than the transit cost in the distribution of data over multicore servers, with increasing data locality and decreasing query processing cost as shown in Fig. 9. This result shows our distributed database system is scalable with the partitioning of database over multicore servers by data clustering based on deterministic annealing 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, and thus the system improves overall performance as well as query processing performance.

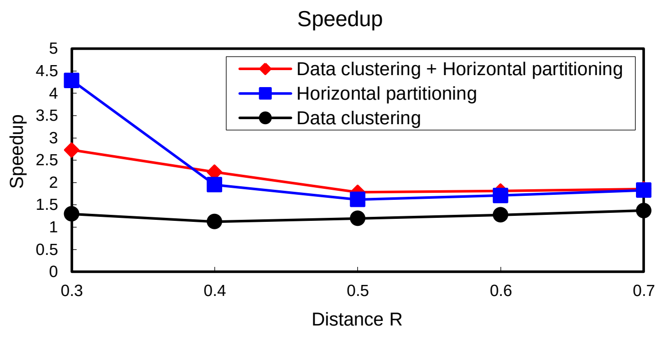

Fig. 5. The value of speedup is the mean query response time in a centralized database system over the mean query response time in a distributed database system.

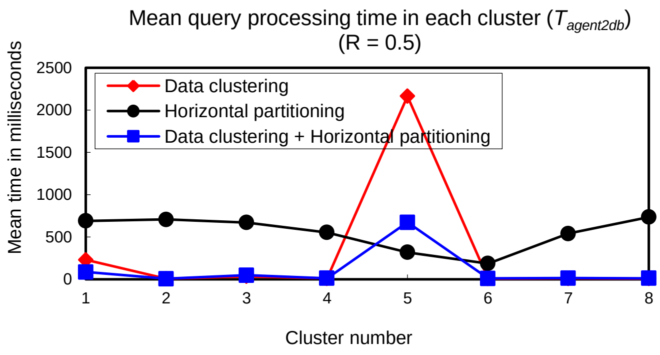

Fig. 6. Mean query processing time in each cluster ($T_{agent2db}$), in the case of the distance R=0.5.

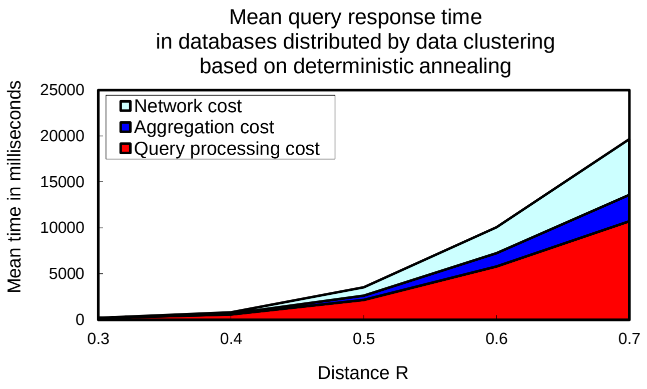

Fig. 7. Mean query response time between a web service client and databases distributed by data clustering including the corresponding execution times of the middleware and agents, varying the distance R.

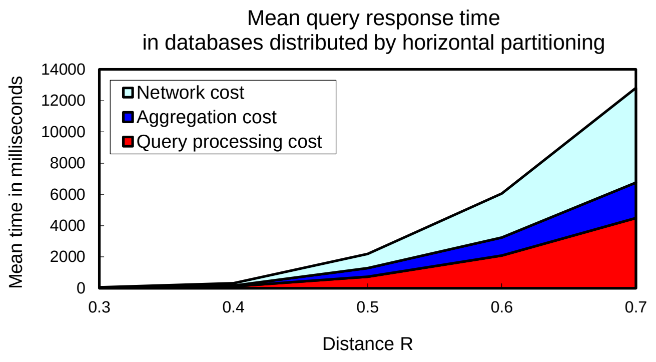

Fig. 8. Mean query response time between a web service client and databases distributed by horizontal partitioning including the corresponding execution times of the middleware and agents, varying the distance R.

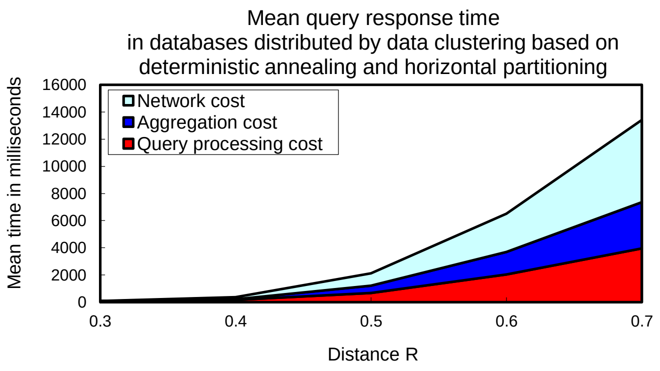

Fig. 9. Mean query response time between a web service client and databases distributed by data clustering and horizontal partitioning including the corresponding execution times of the middleware and agents, varying the distance R.

6. SUMMARY AND FUTURE WORK

We have developed a scalable, distributed database system that allows uniform access to concurrently distributed databases over multicore servers by the SQMD mechanism, based on a publish/subscribe paradigm. Also we addressed the problem of partitioning the Pub3D database over multicore servers for scalability and performance with our architectural design. Our experimental results show our distributed database system is scalable with the partitioning of database over multicore servers by data clustering with deterministic annealing 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 our experiments with our scalable, distributed database system, we encountered a few problems. The first problem occurred with the global aggregation operation in web service that degrades the performance of the system with an increasing number of responses from distributed database servers. In future work we will consider asynchronous invocation web service and also redesign our current distributed database system with MapReduce (Dean, 2004) style data processing interaction mechanism by moving the computationally bound aggregating operation to a broker since the number of network transactions between web service and broker, and the workload for the aggregating operation are able to decrease. The second problem was found in extra hits. We will investigate the use of the M-tree index (Ciaccia, 1997) which has been shown to be more efficient for near neighbor queries in high-dimensional spaces such as the ones being considered in this work.

REFERENCES

Baer, H. (2007). Partitioning in Oracle Database 11g. An Oracle White Paper.

Baru, C. K., Fecteau, G., Goyal, A., Hsiao, H., Jhingran, A., Padmanabhan, S., Copeland, G. P., and Wilson, W. G. (1995). DB2 Parallel Edition, IBM System Journal. Volume 34, pp 292-322.

Community Grids Lab (CGL) (2001). http://communitygrids.iu.edu

Chembiogrid (Chemical Informatics and Cyberinfrastructure Collaboratory) (2006). http://www.chembiogrid.org/wiki/index.php/Main_Page

Ciaccia, P., Patella, M., and Zezula, P. (1997). Proc. 23rd Intl. Conf. VLDB.

Data Clustering (2007). http://en.wikipedia.org/wiki/Data_clustering

Dean, J. and Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. OSDI'04: Sixth Symposium on Operating System Design and Implementation. San Francisco, CA.

Dong, X., Gilbert, K., Guha, R., Heiland, R., Pierce, M., Fox, G., Wild, D.J., J. (2007). Chem. Inf. Model., 47, 1303-1307.

Gadgil, H., Fox, G., Pallickara, S., and Pierce, M. (2006). Managing Grid Messaging Middleware. Proceedings of IEEE Conference on the Challenges of Large Applications in Distributed Environments (CLADE), Paris France, pp. 83–91.

Guttman, A. (1984). ACM SIGMOD, 47-57.

Hennessy, J. L., and Patterson, D. A. (1995). Computer Architecture: A Quantitative Approach. 2nd Edition. Morgan Kaufmann.

Hey, T. and Trefethen, A. (2003). 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 0-470-85319-0.

K-means clustering. http://en.wikipedia.org/wiki/K-means_clustering

Kumar, V., Grama, A., Gupta, A. and Karypis, G. (2003). Instruction to Parallel Computing: Design and Analysis of Algorithms. 2nd Edition. Addison Wesley.

Message Passing Interface Forum (1995). University of Tennessee, Knoxville, TN. MPI: A Message Passing Interface Standard. http://www.mcs.anl.gov/mpi

Multicore CPU (or chip-level multiprocessor) (2008), http://en.wikipedia.org/wiki/Multi-core_(computing)

Pallickara, S., Gadgil, H. and Fox, G. (2005). On the Discovery of Topics in Distributed Publish/Subscribe systems. Proceedings of the IEEE/ACM GRID 2005 Workshop, pp 25-32, Seattle, WA.

PostgreSQL, http://www.postgresql.org/

Pub3d – Web Service Infrastructure (2008), http://www.chembiogrid.org/wiki/index.php/Web_Service_Infrastructure

Qiu, X., Fox, G., Yuan, H., Bae, S., Chrysanthakopoulos, G., Nielsen, H. F. (2007). High Performance Multi-Paradigm Messaging Runtime Integrating Grids and Multicore Systems. Proceedings of eScience 2007 Conference Bangalore India.

Qiu, X., Fox, G., Yuan, H., Bae, S., Chrysanthakopoulos, G., Nielsen, H. F. (2008). Performance of Multicore Systems on Parallel Data Clustering with Deterministic Annealing ICCS 2008: "Advancing Science through Computation" Conference; ACC CYFRONET and Institute of Computer Science AGH University of Science and Technology Kraków, POLAND. Springer Lecture Notes in Computer Science Volume 5101, pages 407-416. DOI

Sacca, D. and Wiederhold, G. (1985). Database Partitioning in a Cluster of Processors. ACM Transaction on Database System, Vol. 10, No. 1, Pages 29-56.

SALSA (Service Aggregated Linked Sequential Activities) (2008), http://www.infomall.org/salsa

Uyar, A., Wu, W., Bulut, H., Fox, G. (2006). Service-Oriented Architecture for Building a Scalable Videoconferencing System, in book "Service-Oriented Architecture - Concepts & Cases" published by Institute of Chartered Financial Analysts of India (ICFAI) University.

Rose, K. (1998). Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. Proc IEEE Vol. 86, pages 2210-2239.