THESIS APPROVAL

GRADUATE SCHOOL, KASETSART UNIVERSITY

Master of Engineering (Computer Engineering)
DEGREE

Computer Engineering	Computer Engineering
FIELD	**DEPARTMENT**

TITLE: Design and Modeling of A Parallel Database System

NAME: Mr. Suphachan Phakhawirotkul

THIS THESIS HAS BEEN ACCEPTED BY

(Assistant Professor Putchong Uthayopas, Ph.D.) **THESIS ADVISOR**

(Miss Thitiwan Srinark, Ph.D.) **COMMITTEE MEMBER**

(Assistant Professor Kemathat Vibhatavanij, Ph.D.) **COMMITTEE MEMBER**

(Mr. Pirawat Watanapongse, Ph.D.) **DEPARTMENT HEAD**

APPROVED BY THE GRADUATE SCHOOL ON April 4, 2006

(Associate Professor Vinai Artkongharn, M.A.) **DEAN**

THESIS

DESIGN AND MODELING OF A PARALLEL DATABASE SYSTEM

SUPHACHAN PHAKHAWIROTKUL

A Thesis Submitted in Partial Fulfillment of
the Requirements for the Degree of
Master of Engineering (Computer Engineering)
Graduate School, Kasetsart University
2006

ISBN 974-16-1333-4

Suphachan Phakhawirotkul 2006: Design and Modeling of A Parallel Database System. Master of Engineering (Computer Engineering), Major Field: Computer Engineering, Department of Computer Engineering. Thesis Advisor: Associate Professor Putchong Uthayopas, Ph.D. 47 pages.
ISBN 974-16-1333-4

Parallel database system was introduced as an economical solution for speeding up the data access. It exploits I/O parallelism by connecting multiple commodity computers together with high speed interconnection network to process queries in parallel.

This thesis proposes the design of a parallel database system called Parallel Virtual Database (PVDB). PVDB offers good performance and cost effective approach by using commodity PC hardware. A proof of concept implementation of PVDB has been built. The experiment shows that this implementation can achieve speedup ratio of 3.8 for 5 database nodes. An improved design of PVDB called Distributed Parallel Virtual Database (DPVDB) also proposed. DPVDB consider distributed environment in its design. The performance of DPVDB is evaluated using a queuing model. A simulation program for DPVDB design is developed and uses to study performance of DPVDB. Finally, this work can be used to speeding up database processing in modern enterprise IT environment.

Student's signature Thesis Advisor's signature 4 / 4 / 2006

ACKNOWLEDGEMENTS

I would sincerely like to acknowledge the efforts of many people who contributed to the research and this thesis in particular. Without them, the work would never been undertaken.

First, I would like to thank my mom, who working so hard to allow me to pay all my attentions to this thesis. This thesis is dedicated to you.

Next, I would like to thank my thesis advisor, the man I call ar-charn, Asst. Prof. Dr. Putchong Uthayopas, who always gives me the knowledge, chances and any other great contributions. I would like to thank all of my thesis committee, Asst. Prof. Dr. Arnon Rungsawang, Asst. Prof. Dr. Kemathat Vibhatavanij and Dr. Thitiwan Srinark for their time and valuable comments. I also would like to thank Asst. Prof. Dr. Anan Phonphoem for his great queuing theory class.

I want to give a very big thank to Miss Tritaporn Peerabool, Jubjang, who always cheer me up and push me to concentrate on the thesis. I would like to thank Mr. Chiluck and his family along with Mr. Chaiwit for your kindly helps and suggestions when I'm in critical situation. I want to thank you all of my friends from my high school and Thammasat University for your cheer up. I would like to thank all staffs and friends at HPCNC who give his contributions and cheer up to me. Thank you, Mr. Sugree, Gee, for your cool simulation library. Thank you, Mr. Thanapol for your helps during thesis submission process.

Lastly, I want to tell you my father, with all of my attentions spent on this thesis, I a little bit grow up now.

Suphachan Phakhawirotkul

April 2006

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Cont'd)

DESIGN AND MODELING OF A PARALLEL DATABASE SYSTEM

INTRODUCTION

Relational database systems are widely used as main data storage for current applications. These applications generate numerous data operations to the database system in a day. These operations include both simple transaction and complex data query. Each simple transaction can be finished using short processing time but database system should simultaneously handle large number of transaction. A complex query processes large amount of data to extract information from the database. Database system should finish these operations as fast as possible to maintain usability of client applications.

The development of storage technology trends to increase capacity faster than data transfer speed. As CPU speed increasing, performance of database system does not increase as expected because limited I/O bandwidth. The easiest way to solve this problem is to use I/O device that has highest transfer speed available on the market. This method solves the problem by just raise bandwidth limit to a higher level. Anyway, the system may reach the limit again especially when data size grows up. Additionally, such devices are available at very expensive price.

Parallel database system was introduced to solve I/O bandwidth problem at lower price. It exploits I/O parallelism by connecting multiple commodity computers together with high speed interconnection network to process queries in parallel. Parallel database achieves total I/O bandwidth equal to summation of bandwidth from all nodes in the system. Furthermore, when the I/O bandwidth reaches the limit, simply add a new node to increase system I/O bandwidth.

Key technique that enables I/O parallelism within parallel database is data declustering. Data declustering is the process that partition data into smaller pieces then place these pieces on nodes in the system. When system processes a query, each node in the system are simultaneously process their local data. This significantly reduces query processing time due to multiple parts of data are processed in parallel.

However, data declustering technique also have an important drawback, unbalanced load. Load unbalancing is resulted from data declustering scheme used does not evenly distribute load to all nodes in the system. Highly loaded nodes have longer query processing time than lightly loaded nodes and increase overall system query processing time.

This thesis studies performance behavior of shared nothing parallel database system. The study methodology is proposing designs of parallel database system and evaluates implementation of the design and simulation.

Objectives

1. Studying performance behavior of shared nothing parallel database system by proposing proof of concept designs

2. Developing implementation and performance model of proposed designs those use for evaluation

Research Scope

1. Proposing design of a parallel database system called Parallel Virtual Database (PVDB)

2. Proposing design and performance model of a parallel database system called Distributed Parallel Virtual Database (DPVDB)

3. Evaluating proposed design by using the implementation or simulation program

LITERATURE REVIEWS

This section describes background on parallel database system, proposed data declustering schemes from previous works and architecture of Parallel Virtual Database.

Parallel Database System

Parallel database (DeWitt and Gray, 1992) is a database system that has multiple processors connected together with high-speed interconnection network to process queries in parallel. Parallel query processing helps parallel database finish each query faster. Additionally, it allows parallel database system to process the query on a larger database. This is the main advantage of parallel database over normal database system.

There are three different types of architecture for parallel database system: shared-memory, shared-disk and shared-nothing. This determines by level of resources sharing. Shared-memory is the system that every processors share memory and disk storage with the others. In shared-disk architecture, each processor has its own memory but shared disk storage. For shared-nothing, each processors has its own memory and disk storage but still connected to interconnect network to exchange messages with the others.

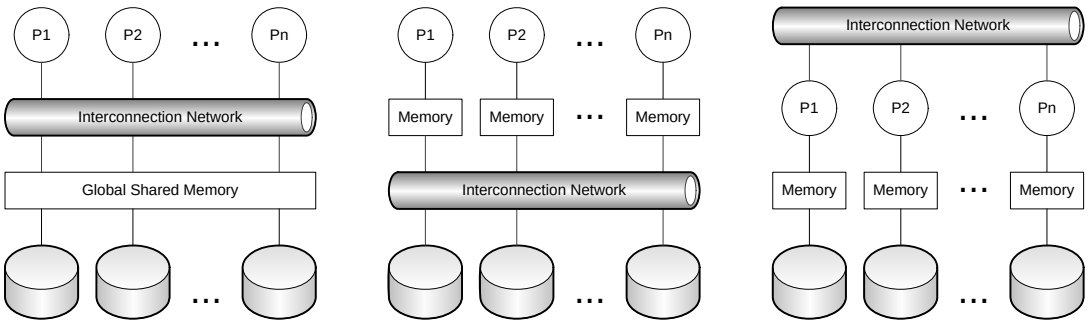


Figure 1 Parallel database system architectures

Figure 1 demonstrates physical connection of processors, memories and disks in parallel database system by architecture. In shared-memory architecture, all processors connected to global shared memory and disks via interconnection network. This means that all raw memory and raw disk traffics from all processors should pass the interconnection network. In shared-disk architecture, each processor accesses its memory using its local bus but accesses shared disk pool via interconnection network. So that only raw disk access traffics passing the interconnection network. For shared-nothing architecture, each processor accesses both memory and disk via its local bus.

There are raw access traffics passing the interconnection network. Interconnection network is used to exchange messages among processors only.

Increasing number of processors does not always increase system performance. There are three major threats those limit scalability of parallel database system: startup, interference and skew. Startup is the overhead to start a parallel execution. This overhead is normally occurred in parallel processing. As the number of processor increased, the startup time will dominate the execution time. Interference is happened due to multiple processors accessing limited shared resources in the same time. If the resources busy, those processors have to wait until they are freed. Waiting for share resources increases processing time without generate any results. Level of interference is increased as the number of processor is increasing. Skew is the situation that variance of processing time of all processors in the system is high. This happened because unbalance in workload among processors. Different in hardware performance and data distribution and workload pattern mismatch may cause unbalanced workload. Skew has significant impact on system response time because it is determined by processing of the slowest processor.

Shared-nothing architecture is the most interesting architecture because it offers good performance at lower price. Both shared-memory and shared-disk architecture require superior interconnection network (memory or I/O bus) with very high bandwidth in order to support raw memory and disk accesses from all processors. Such interconnection network is difficult to build and may available at very expensive price. In shared-nothing architecture, all raw memory and disk accesses are locally performed on each processor only. Commodity network devices can sustain message exchange bandwidth among processors. Further more, shared-memory and shared-disk architecture has limited scalability due to interference among processors when accessing shared memories or disks. All processors in shared-nothing system are able to perform I/O operations simultaneously without interfere with other processors.

However, unbalance workload is the important limitation for shared-nothing architecture. Workload distribution can be controlled by data declustering scheme used in the system. Choosing appropriate scheme is the key to balance workload. Details on data declustering will discuss later in this section.

Data Declustering Schemes

Data declustering is the most important mechanism in shared-nothing parallel database system. It enables I/O parallelism by place different pieces of data on different processors. When data needed, all relevant processors scan its local data in parallel then summarize before return to the requester. Target object for declustering is relation (table) in database system or file in file system. Data declustering process consists of two sub-process data partitioning and data placement.

Data Partitioning and Data Placement

Data partitioning is the process that break original data into smaller pieces called partition or fragment. There are two major types of data partitioning: horizontal partitioning and vertical partitioning. Horizontal partitioning partition a relation by record basis while vertical partitioning partition a relation by column basis. Horizontal partitioning uses values from selected attributes called partitioning attributes to partition the relation. If only one partitioning attribute used, we called single-dimension data partitioning. If more than one partitioning attribute used, we called multi-dimension data partitioning.

Data placement is the process that place partitioned data or fragments on processors in the system. So, it is the mapping between a set of fragment to a set of processor. Fragments can be arranged in single-dimension or in multi-dimension but processors are arranged in single-dimension only. Since each process only accesses data on its local disk, data placement determines workload distribution in the system.

One data partitioning method couples with a data placement method is called data declustering scheme. Some of previous works combined data placement into data partitioning process. These works partitioned data by consider number of processors in the system as main factor. They refer the whole data declustering process as data partitioning only. Anyway, this research uses the term data declustering to refer both data partitioning and data placement process.

Data reorganization

Data reorganization is the process that migrate data among processors is needed in order to maintain the system at optimal state. Data placement after data declustering process is called initial data placement. Initial data placement cannot efficiently support all workload patterns. Organizing data to specially support current workload can improve performance of the system.

However, data reorganization is considered a very expensive operation. It consists of following steps, monitoring system and load parameters, determine when operations needed, select processors for the operation, calculate amount of data to be migrated, migrate data and update system settings. All these steps need to perform while system is online. Perform offline reorganization consider more expensive operation and reduce data availability.

Data declustering schemes that consider reorganization are called dynamic declustering scheme. Data declustering schemes that do not consider reorganization are called static declustering. Dynamic declustering not always better than static declustering due to reorganization cost.

Single-dimension Declustering Schemes

There are three basic data declustering schemes: round-robin, hash and range (Ghandeharizadeh and DeWitt, 1990a). All of these schemes are horizontal partitioning. Records distribution for these schemes is demonstrated in Figure 2.

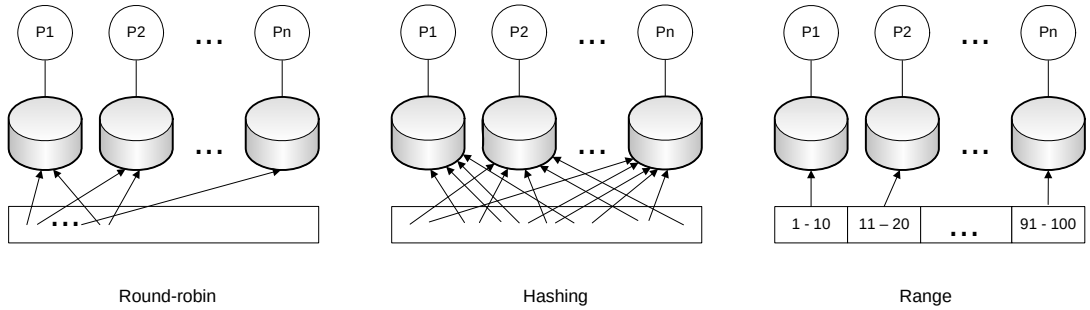


Figure 2 Basic data declustering schemes

Round-robin scheme distributes records one by one to all processors in round-robin order. It has good load balancing because records are evenly distributed to all processors. In round-robin scheme, all processors have to involve in any type of queries because it has no knowledge on data placement.

Hash scheme applies a hash function on a partitioning attribute to determine destination processor for each record. It suit for exact match query because target processor for the query is easily determined by applying hash function on the search key. In hash scheme, all processors have to involve when processes a range query because it does not concern data clustering. Data clustering is the property that adjacent data are placed on the same processor.

Range scheme divides the range of possible values of partitioning attribute into smaller value ranges. The number of divided range equals to number of processor in the system. For example, if range of possible value of partitioning attribute is 1 – 100 and there are ten processors then divided ranges are 1 – 10, 11 – 20, 21 – 30 … and 91 – 100. These value ranges are kept in a table called range table which maps a value range to a destination processor. All records in the relation then assigned to processor by consulting range table. Range scheme offers good performance for both exact match and range query. It has both knowledge on data placement and data clustering property. However, range scheme has poor load balancing. Processors those hold data in range that has high access frequency will have more workload than the others. These processors significantly increase total execution time.

Hybrid Range Partitioning Strategy (HRPS) (Ghandeharizadeh and DeWitt, 1990b) combines data clustering property of range scheme with good load balancing of round-robin or hash scheme. This is the first scheme that separates data partitioning and data placement process.

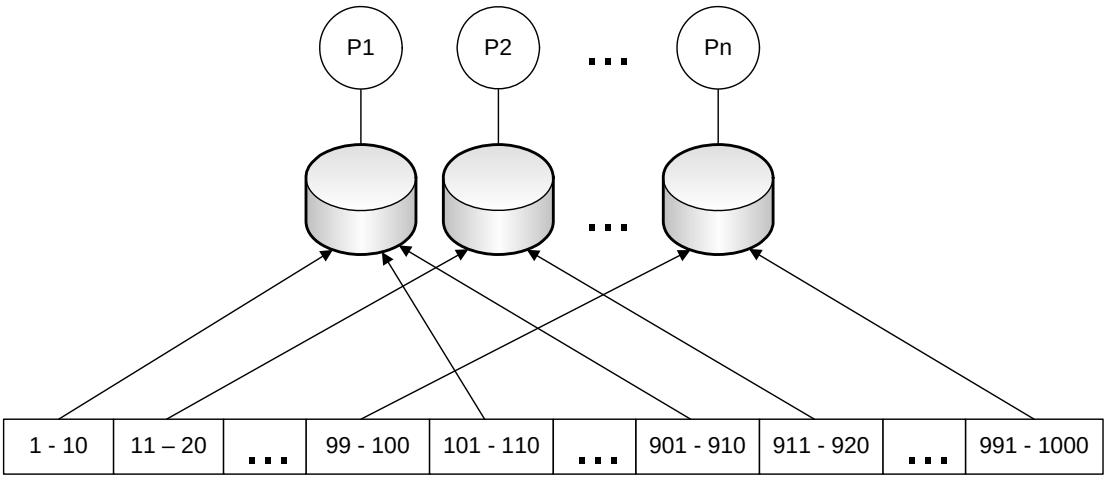


Figure 3 Hybrid Range Partitioning Strategy

In HRPS, relation is partitioned into small pieces called fragment. The partitioning process is the same method as range scheme but regardless to number of processor in the system. Number of fragment should be much more than the number of processor. So, fragment size should be smaller than partitions in range scheme. HRPS uses average query size divided by number of processors to determine optimal fragment size. Each fragment then assigned to processors using round-robin or hash function. With this combination, HRPS can localize the execution of those queries that have minimal resource requirements to a few processors while directing queries with high resource requirements to a large number of processors.

Multi-dimension Declustering Schemes

All data declustering schemes described above use only one partitioning attribute. When a query search data on other attributes than partitioning attribute, all processors involve in that query. Multi-dimension data declustering scheme uses multiple partitioning attributes to partition data. Fragments from multi-dimensional partitioning are called tiles or grids. The problem is how to place data grids on to single-dimension array of processors so that system processes queries at optimal performance. Many data placement methods are proposed to solve this problem.

The Disk Modulo (DM) (Du and Sobolewski, 1982) assigns tile (x, y) to disk number (x + y) mod M. The Field Exclusive-Or (FX) (Kim and Pramanik, 1988) assigns tile (x, y) to disk number dec(bi(x) (+) bi(y)) mod M. Here bi(x) is the binary representation of x and dec(z) is the decimal number corresponding to the binary representation z. These schemes are two of the earliest works on multi-dimension data declustering. Both schemes were intended for partial match queries, a special case of range queries where the range on each dimension is either a single value or covers the entire value range (the latter must occur in at least one of the dimensions). The Hilbert Curve Allocation Method (HCAM) (Faloutsos and Bhagwat, 1993) is a declustering scheme for range queries that is based on a space-filling curve. HCAM was shown to outperform DM and FX in general for range queries and gives comparable

performance to a scheme based on error-correcting codes (ECC), which exists only for a few values of processor and grid sizes. A scheme called Linear Allocation (LA) (Zhou et al., 1994) that maps 2-dimension data tiles (x, y) to disk x + h*y mod M, where h is a predetermined integer and M is the number of processor. When M = 8 and h = 3, the LA scheme outperforms HCAM, DM, and random assignments, for range queries. The paper does not elaborate on how to find the best h for any given M. Cyclic Declustering (CD) (Prabhakar et al. 1998) generalizes LA scheme to multi-dimensional and proposed strategies to search for good h. CD maps d-dimension data tiles (x0, x1, . . . , xd-1) to disk x0 + h1x1 + h2x2 + . . . + hd-1xd-1 mod M, where hi, called skip values, are predetermined integers depending on M. Two heuristics were proposed to find good skip values for any given M. The first algorithm, GFIB, is based on Fibonacci numbers and the second algorithm, EXH, is based on an almost exhaustive search of the best skip values. Both schemes outperform HCAM. Coloring is a scheme that is defined only when M is a power of two. In two dimensions, it guarantee that the response time of any query is at most O(log M) more than the optimal response time. Generalized Multidimensional Data Allocation (GeMDA) (Lo et al., 2001) is the scheme that uses $\lfloor \sqrt[d]{M} \rfloor^i$ as the skip value for dimension i, i = 0, 1, … , d -1. GeMDA outperforms DM and HCAM for range queries. Golden Ratio Sequent (GRS) (Bhatia et al., 2000) is 2-dimension declustering scheme that uses golden ratio to place data tiles on M processors. GRS claims that it is defined for all value of M and outperforms all previous schemes. GRS is extended to n-dimension declustering by using Kronecker sequences (Chen et al., 2003)

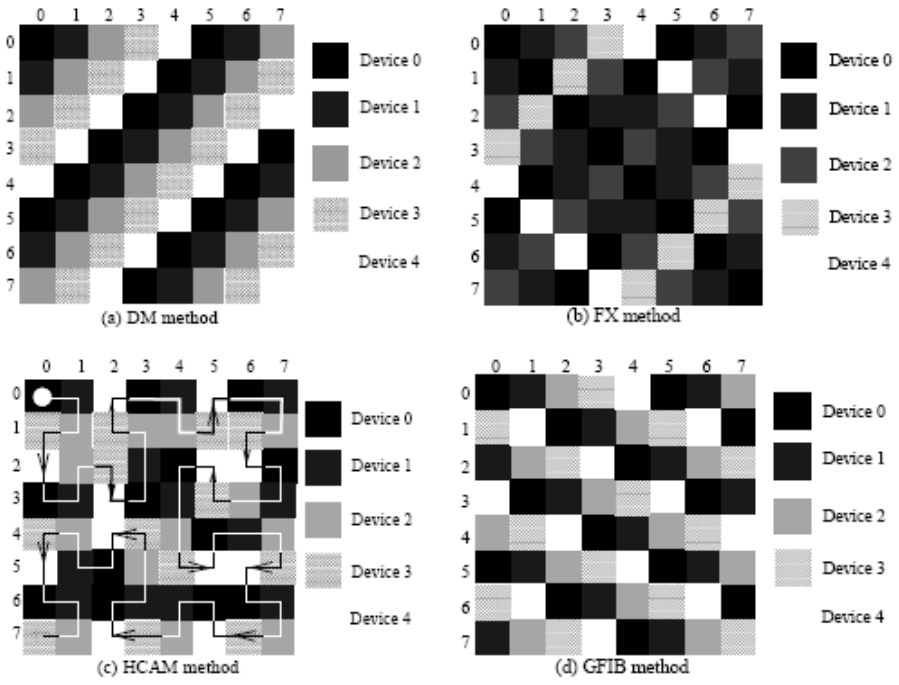


Figure 4 Data placement map of multi-dimension declustering schemes

Figure 4 shows data placing patterns those generated by DM, FX, HCAM and GFIB schemes onto system with five processors. The relation is partitioned into 2-dimension data tiles with eight values in each dimension.

Dynamic Declustering Schemes

Some researches believe that well balanced static declustering techniques do not enough. Reorganizing data placement helps the system to maintain its performance at optimal state.

Most early data declustering schemes try to balance data volume among processors. A data placement scheme used in Bubba (Copeland et al., 1988) is the first work that tries to balance heat instead. Heat is defined as the sum of the number of accesses to data per time unit, as determined by statistical observation over a certain period of time. This work also introduces the term temperature as the ratio between heat and size. The concept of balancing heat and temperature is used in many later works.

Disk Cooling Algorithm (DCA) (Scheuermann et al., 1998) is a heat balancing algorithm that works on multiprocessor database or file system. It balances heat by migrating data from the hottest processor to the coldest processor. By running Disk Cooling periodically, heats on all processors are well balanced. This work also proposes a greedy-base data placement heuristic used for initial data placement.

The heuristic places data in the way that heats on all processors are balanced. First, fragments of a relation are sorted by heat value in descending order. Then, each fragment is assigned to the coldest processor, one fragment at time.

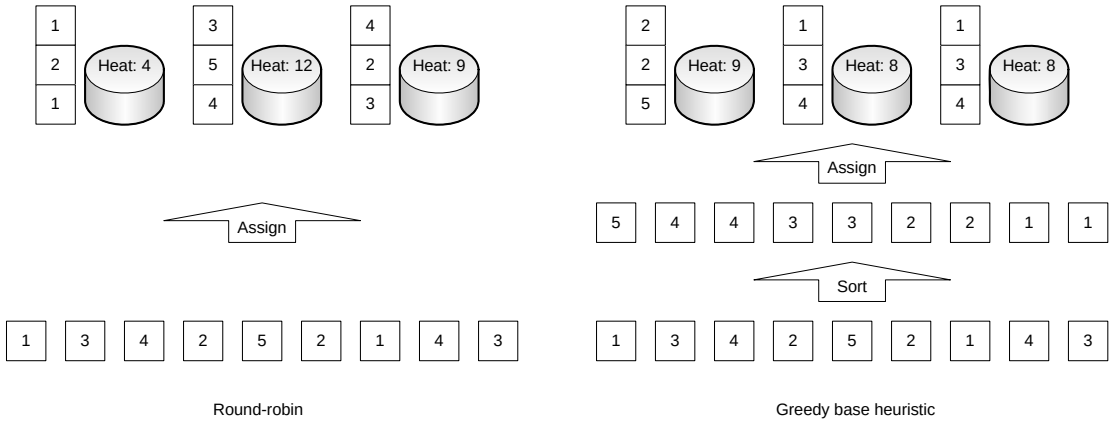
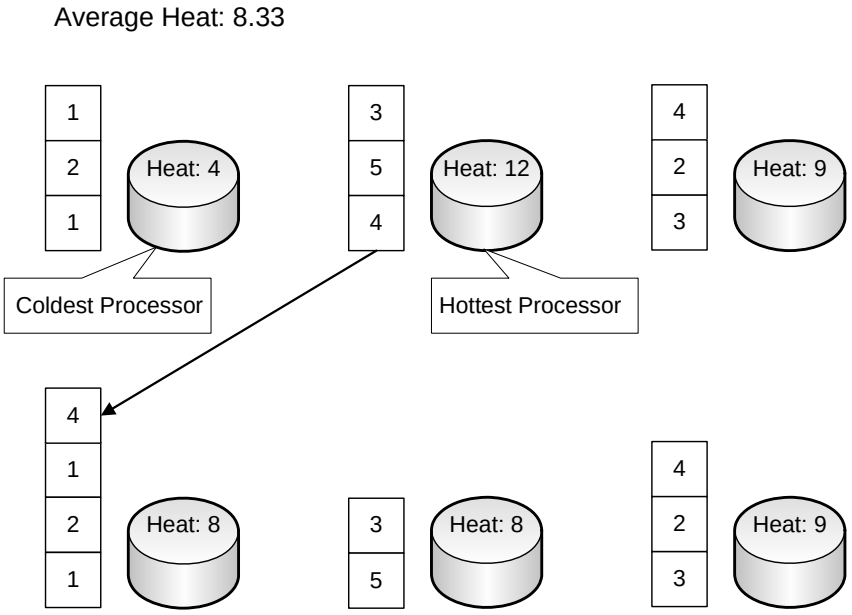


Figure 5 Initial data placement for Disk Cooling

Figure 5 illustrates the process to place data fragments using round-robin and greedy base heuristic. Each small rectangle represents a data fragment. The number in rectangle denotes heat value of that fragment. As present in figure 5, heats on system

that uses greedy base placement are well balanced, compared to one that uses round-robin.

After initial placement, heat on each processor is tracked to be used in DCA. DCA process is started in fixed interval. When it starts, it checks whether the hottest processor has more heat than average heat at significant amount. The algorithm continues only if this condition is satisfied. It then finds the highest temperature fragment from hottest processor that does not make the coldest processor to have higher heat than hottest processor. If such fragment found and hottest processor is



idle, the fragment is moved to the coldest processor and update heat information for both processors.

Figure 6 Disk Cooling process

Figure 6 demonstrate an example of DCA process. This example applies a DCA process on round-robin data placement that shown in figure 5. It assumes the fragment size is 1. So temperature for each fragment is equal to its heat. This process moves the fragment with heat = 4 from hottest processor to coldest processor because moving the fragment with heat = 5 cause the coldest processor has higher heat than current hottest one.

Full Windows algorithm (FWA) (Feelifl et al., 2000b) is an improve heat balancing algorithm from Disk Cooling. FWA only applies on range scheme where each processor is able to migrating data with its neighbor only. It claims that in this configuration DCA will enter infinite balancing loop. FWA analyzes data movement requirement from all processors before make real migration. This is the main different point from disk cooling that use greedy base algorithm to make migration decision.

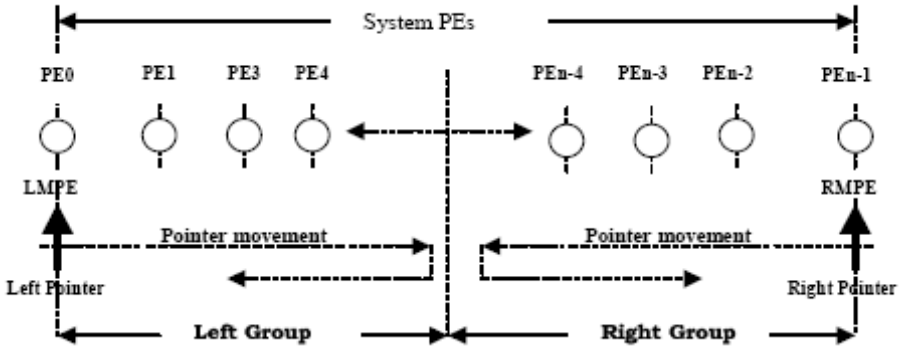


Figure 7 Full Windows algorithm

Figure 7 shows a process of Full Windows algorithm. FWA starting by places left pointer on left-most processor, this work called LMPE, and also places right pointer on right-most processor, called RMPE. It compares processor that pointed by the left pointer with its right neighbor, called RNPE and records data migration task on migration table. The processor pointed by right pointer also does the same things but compares with its left neighbor, called LNPE. After finish both comparisons, the left pointer is moved to its RNPE and the right pointer is moved to its LNPE. The same comparison is performed until RNPE of left pointer equals to right pointer. FWA then performs data migration tasks as recorded in migration table.

Two-bags algorithm (TBA) (Feelifl and Kitsuregawa, 2000a) is also balancing heat on parallel database using range scheme but with different configuration from FWA. In TBA, all processors are configured as a ring. This means that it possible to migrating data between left-most processor and right-most processor. In FWA, processors are configured using linear configuration where left-most processor cannot exchange data with the right-most processor.

However, data reorganization is considered as a very expensive operation. It increases workload on migrating processors those already have heavy load. There is an interesting simulation result that using full declustering is better than partial declustering with data reorganization (Mehta and DeWitt, 1997). Before this work, research community believes that full declustering leads to reduce performance. Changing in hardware technology reduces effects of startup and communication overheads on system performance. Full declustering provides high parallelism and good load balance. Combine with dynamic query delay enable system to adapt itself to support workload change without need of data reorganization.

Types of Parallelism

There are two types of parallelism existed in parallel database: intra-query parallelism and inter-query parallelism. Intra-query parallelism is the parallelisms within a query that achieve by simultaneously execute multiple operators for such query on different processors. Inter-query parallelism is the parallelism among queries that currently execute on the system. The goal of intra-query parallelism is to

minimize system response time while inter-query parallelism is to maximize system throughput. Intra-query parallelism consumes system resources as much as possible to quickly finish single query. On the other hand inter-query parallelism keeps resources for each query minimum in order to maximize number of queries running on the system.

Each data declustering scheme has both types of parallelism but at different level. Round-robin is good example scheme that gives high intra-query parallelism because it involve all processors to process a query. Range scheme give high inter-query parallelism because it localizes execution of query to only relevant processors. So, it depends on type of parallelism that application required when choosing a data declustering scheme. Complex queries from DSS applications that process large volume of data require high intra-query parallelism. In contrast, large numbers of simple queries from OLTP applications need high inter-query parallelism.

MATERIALS AND METHODS

Materials

1. One of 6 nodes cluster machine
    1.1. Dual Pentium4 1.8 GHz
    1.2. 1 Gigabyte memory
    1.3. 80 Gigabytes hard disk drive
    1.4. Fast Ethernet adapter
2. Two of 16 ports Fast Ethernet switches
3. Client workstation
    3.1. Pentium4 1.8 GHz
    3.2. 1 Gigabyte memory
    3.3. 80 Gigabytes hard disk drive
    3.4. Fast Ethernet adapter
4. Development workstation
    4.1. Pentium4 1.4 GHz
    4.2. 512 Megabytes memory
    4.3. 40 Gigabytes hard disk drive
    4.4. Fast Ethernet adapter
5. Software
    5.1. Operating system
        5.1.1. Linux Fedora Core 1
        5.1.2. Windows XP Professional
    5.2. Database management system
        5.2.1. MySQL version 4.0

5.3. Development tool and library

5.3.1. GNU C/C++ complier and utilities version 3.2

5.3.2. Python version 2.4 for Windows

5.3.3. Hypersim simulation development library for Python version 1.0

5.3.4. SSH Secure Shell version 3.2.9

Methods

Parallel Virtual Database

Parallel Virtual Database (PVDB) is a proof-of-concept parallel database system developed as an alternative to expensive multiprocessor database system. It also used for studying performance characteristics of parallel query execution on multiprocessor database system. The goal of PVDB is to use commodity hardware devices to build high performance database system at low price. This section discusses about architecture and design of PVDB system to support its goal.

The PVDB architectural design is divided into two parts: system architecture and software architecture. System architecture design concerns on system components and their organization, interconnection network and multiprocessor architecture used to build a PVDB system based on design assumption. The goal of system architecture is to establish a concrete operating environment for software part of PVDB. Software architecture design concerns on functionality of software components and how they interact to process user requests. The goal is to make PVDB software simple, quickly developed, easy to maintain and has good performance.

The architectural design of PVDB system and software are discussed in next sections. The limitation of the design is discussed. The proof-of-concept implementation of the design also reviews at the end of section.

PVDB System Architecture

PVDB design is based on shared-nothing architecture (SNA) because it offers high performance, high scalability and low cost. SNA offers high performance from I/O parallelism. I/O parallelism is the ability to perform multiple I/O operations in the same time which greatly speedup the query execution. SNA offers high scalability from low interference among processors in the system. Each processor in SNA accesses its local disk when process a query and only high level messages are exchanged over the interconnection network. SNA offers low cost from utilize only commodity computing and networking devices.

A PVDB system consists of two parts: server side and client side. Client side is the part that generates SQL queries to the server side. It consists of two components: Clients and Public Network. The Clients are workstations those translate user commands into SQL queries and send them to server side. The Public Network is any networks between Clients and server side. It carries SQL queries from Clients to server side and results from server side back to Clients. Server side is the part that processes SQL queries and return results back to client side. It consists of three components: Front-end, Private Network and Database Nodes. Front-end Node receives user's queries and coordinates Database Nodes to execute them in parallel. Private Network connects the Front-end and all Database Nodes together and transfers messages for them. The Database Nodes are the collection of node those have data stored. They process sub-queries on their stored data as requested by the Front-end Node.

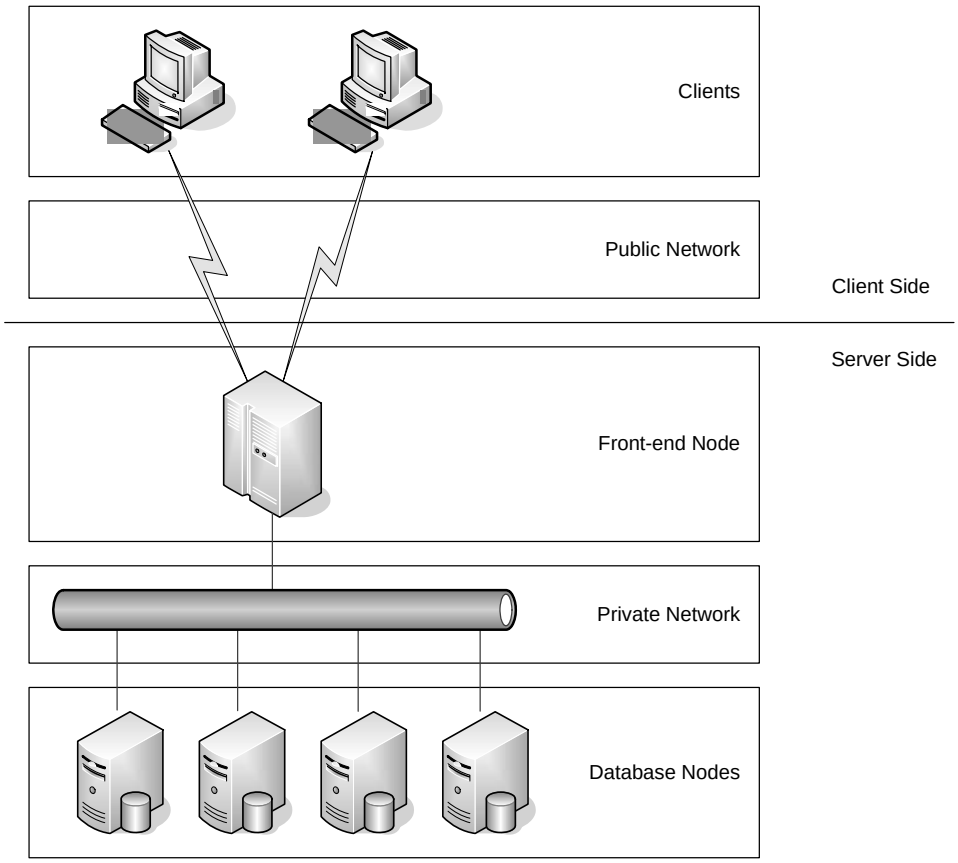


Figure 8 PVDB system architecture

Figure 8 demonstrates PVDB system architecture as described in above paragraph. Notice that the PVDB system architecture arranges components based on direction of data flow. The Database Nodes which is the data producer component is placed at the bottom of architecture. The Clients which is the data consumer component are placed on the top of architecture. Other components are placed between Database Nodes and Clients in same order as data flow through during query execution.

Client Side

PVDB Client side is the part that generates SQL queries to the server side. It consists of two components: Clients and Public Network.

The Clients is the component that interacts with the user. It translates user's command into SQL queries then submit to server side and display the returned results. There are two components within Clients: Client Application and PVDB Client Library. Client Application is the component that interacts with the user. The PVDB Client Library is the component that communicates with the PVDB Server side. Figure 9 illustrates the organization of components within the Clients.

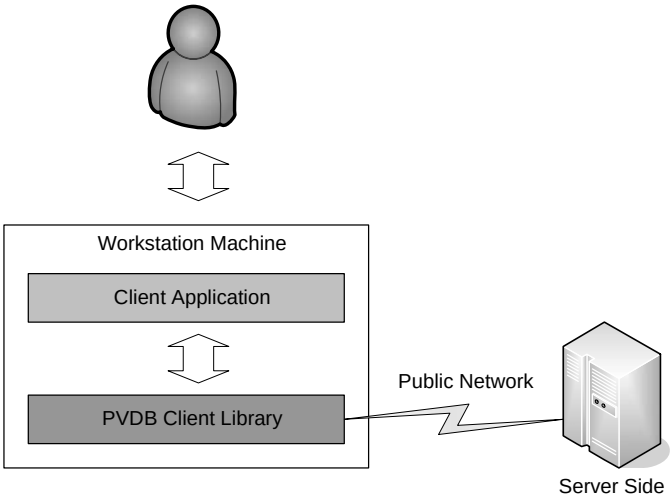


Figure 9 PVDB client side

In figure 9, client application uses PVDB Client Library to communicate with the server side. PVDB client library contains routines those necessary to communicate with server side and provides APIs for database applications to utilize such routines. Client applications are able to send requests to PVDB server side by calling client library only.

Server Side

Server side of PVDB is the part that process user queries and reply results back to Clients. It consists of three components: Front-end Node, Private Network and Database Nodes. Figure 10 illustrates the design of PVDB server side.

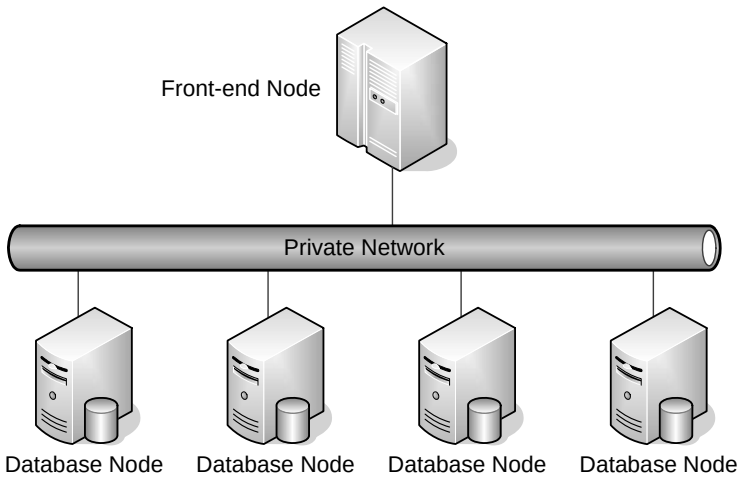


Figure 10 PVDB server side

The PVDB server side is designed base on shared-nothing architecture which no share resources among processor in the system. All processors in PVDB are called

node. Each node is a set of personal computer which is comprised of commodity microprocessor, memory, disk controller, network adapter and disk storage. All nodes are connected with high speed private network. There are no special devices those connect nodes to shared resources.

A PVDB system can have more than one Database Node but only one Front-end Node. Database Node is the components that store and manage data for PVDB. Each Database Node stores subset of data those resulted from data declustering process. Combines data from all Database Nodes will reconstruct the full set of data. Database Nodes wait for sub-queries from Front-end Node. When a query arrives at a Database Node, the query is executed against local stored data and replies the result back to the Front-end Node. This result is only intermediate result from one Database Node. It needs to be combined at Front-end Node with results from the others.

The Front-end Node is the core of PVDB system. It waits for queries from client side and coordinates with Database Nodes to execute those queries in parallel. When a query arrives at the Front-end Node, the Front-end Node selects subset of database nodes to participate in the execution. The Front-end then submit the query to all selected nodes and waits for results. After results from all selected Database Nodes are returned, the Front-end Node summarizes all intermediate results into final result. The final result is then returned to the user.

All nodes in server side are connected using private high speed network. This network is used to transfer queries from Front-end Node to Database Nodes and transfer intermediate results from Database Nodes back to Front-end Node.

Database Node Organization

PVDB also concerns on load balancing and fault tolerant by organizing database nodes into node groups. Nodes group is collection of database node that contains the same data. When data need to updated, all processors in node group should receive the update command to maintain group integrity. Only one processor in the group is selected to execute query.

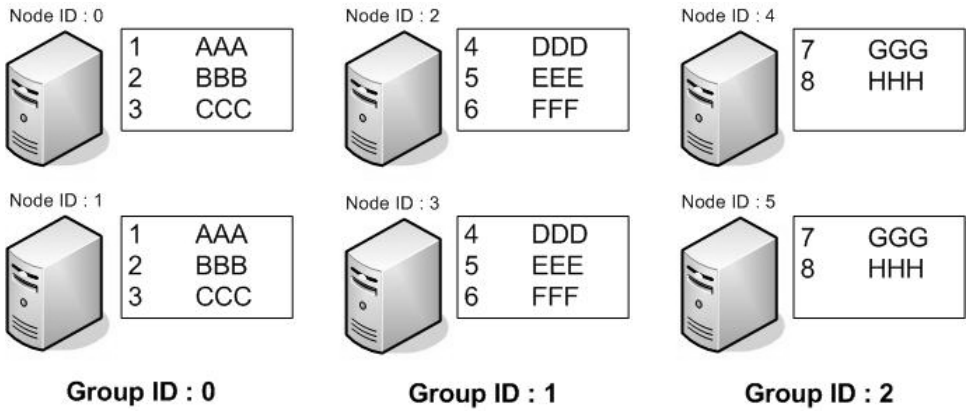


Figure 11 Node grouping

Figure 11 shows an example of node group organization for a PVDB system. This system has six database nodes. All nodes are organized into three different node groups. Each group contains different part of data depending on data declustering scheme used. Round-robin scheme is applied in this example. All processors in a node group contain the same data. When the system needs to update record number 2, the update command is sent to all nodes in group 0 those are node 0 and node 1.

There are three patterns to organize node group: distribute, parallel and mixed. Distribute pattern assigns all database node into one group. Parallel pattern assigns each node to its own group. Mixed pattern assign nodes to more than one group and at least one group has more than one node. Distribute pattern offers highest level of fault tolerant but does not speedup queries execution. However, it able to distributes requests to all nodes when supports heavy workload. Parallel pattern speedups the queries execution by partitioning data to all database nodes but it lacks of fault tolerant. Mixed pattern combines good speedup of parallel pattern and fault tolerant with load distribution of distribute pattern together. Mixed pattern is recommended pattern for most system.

PVDB Software Architecture

The software part of PVDB is the part that controls query execution mechanism within PVDB. PVDB software architecture is divided into three layers: Data, System and User. Data layer focuses on data storage management and basic data operations. The system layer focuses on user session management, system configurations and parallel query execution mechanism. User layer focuses on mechanism to access PVDB for the users.

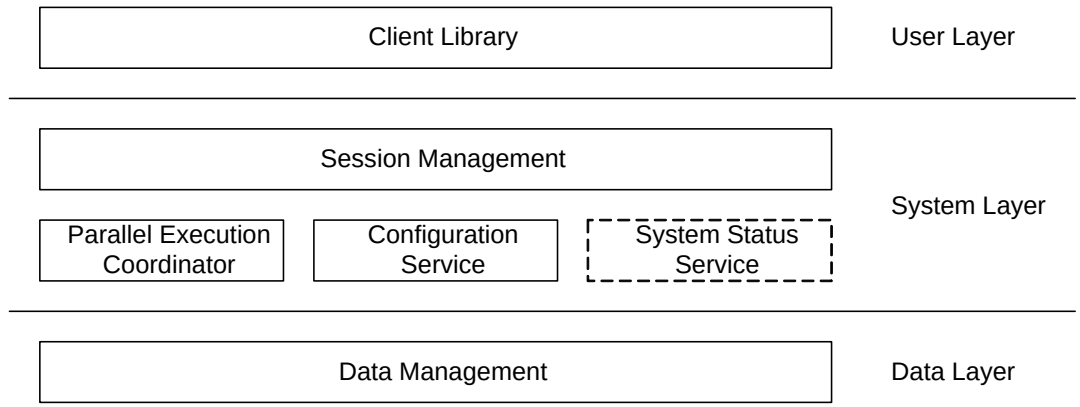


Figure 12 PVDB software architecture

Figure 12 illustrates the software architecture of PVDB. In the User Layer, there is only one software components that is the Client Library. The Client Library provides a set APIs for applications to communicate with PVDB. The basic operations those should be included in the API is connection management, query submission and result retrieval.

There are three components in the System Layer: Session Management, Parallel Execution Coordinator and Configuration Service. Session Management is the component that handles client connections. It waits for incoming connections and maintains the connections from multiple clients during execution process. Parallel Execution Coordinator is the component that prepares and executes each query in parallel. It generates parallel query execution plan and coordinates with Data Management components to execute the plan. The Parallel Execution Coordinator consults Configuration Service for system information and other decisions in system related issues such as list of Database Node in the execution. The Configuration Service maintains system information and provides information and decisions to other components. Such configurations are database schema, database node organization, and data declustering information.

There is an optional component in System Layer: System Status Service. Optional means PVDB able to continue its functionality without this component. The System Status Service tracks system statistic information such as access frequency to each fragment, average load of each Database Node, etc. These information are used by Configuration Service to make decision accurately and System Administrators for information of system load.

There is only one component in Data Layer that is the Data Management. Data Management is the component that stores and manages data on each Database Node. It performs actual operations onto the data stored locally on Database Node.

PVDB Implementation

The architectural design of PVDB has implemented on Linux operating system. This is only the proof-of-concept implementation uses to evaluate the design.

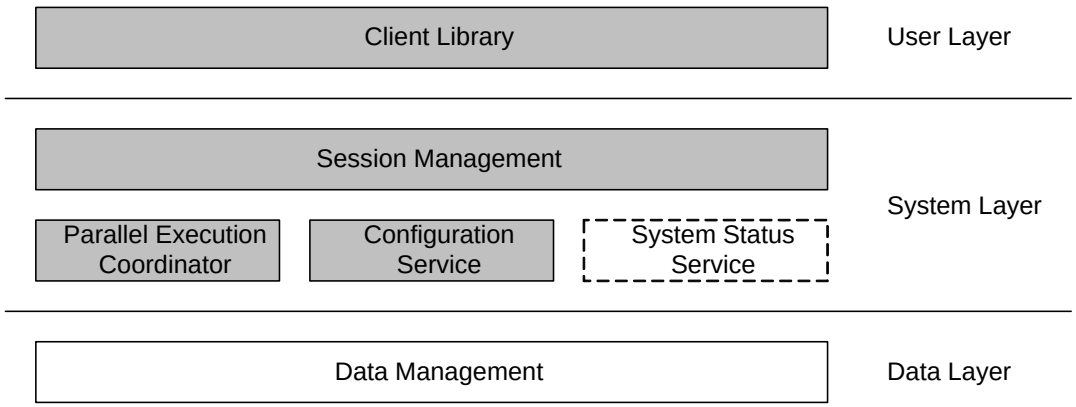


Figure 13 Implemented components

From figure 13, illustrates the PVDB software architecture design with some shaded box. The shaded blocks represent the components those selected to implement in the implementation.

Non-shaded blocks are System Status Service and Data Management. The System Status Service does not include in the implementation because it is only optional component. Data Management is selected not to be implemented because PVDB uses current database management system (DBMS) as the Data Management component instead. Using DBMS helps PVDB implementation bypass many details on data storage management and data processing and focuses only on parallel query execution mechanism. DBMS also offers many abilities in addition to basic data processing such as handle multiple requests in the same time, dynamic resources management and optimized execution plan. Furthermore, PVDB automatically inherits new data processing techniques from the DBMS as they are introduced. The selected DBMS for proof-of-concept implementation is MySQL version 4.0.

The selected components has implemented into two pieces of software: PVDB Client Library and PVDB Server. PVDB Client Library implements the Client Library component in the architecture. It is the software that installed on user's workstations and developer's machine. Developers include PVDB Client Library into their codes and produce client applications. The client applications on user's workstation call PVDB Client Library routines to communicate with PVDB Server. The PVDB Server implements all selected components in System Layer of the architecture. The PVDB Server is implemented as multi-thread sever UNIX daemon. The PVDB Server is installed only on the front-end node. The components organization of PVDB implementation is demonstrated in figure 14.

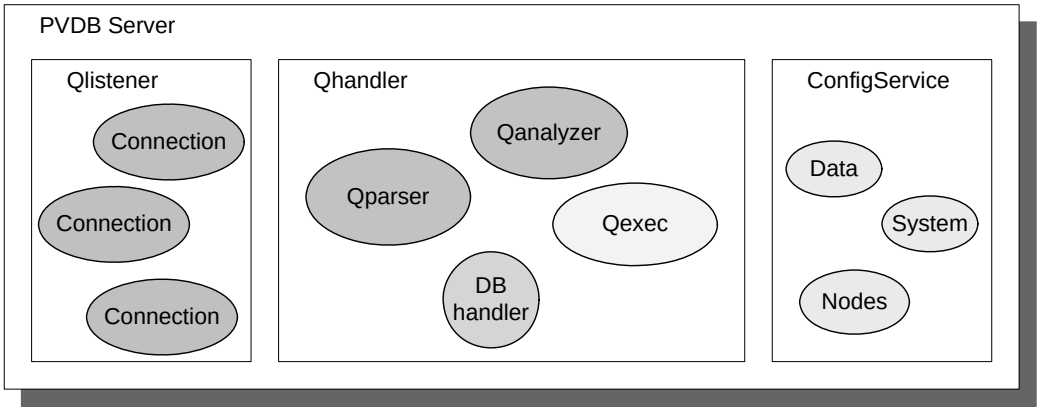


Figure 14 PVDB server implementation

There are three major modules in PVDB Server: Qlistener, Qhandler and ConfigService. Qlisterner implements Session Management component of the architecture. It listen connection on a TCP port and keep status of connection when a connection with client is established. Qhandler implements Parallel Execution Coordinator components of the software architecture. It is created and attached to a thread to handle a query when user submits the query to Qlistener. Details on query execution process are described in the next section. ConfigService implements the Configuration Service of the software architecture. ConfigService is implemented as a

set of function that read and process configuration files to return information to the caller.

Query Execution Process

In PVDB, the whole query execution process is handled at the front-end node. When PVDB receives a query, it creates a attached with a Qhandler (query handler) to execute the query. Figure 15 is the diagram that shows steps and components those involve when Qhandler execute a query.

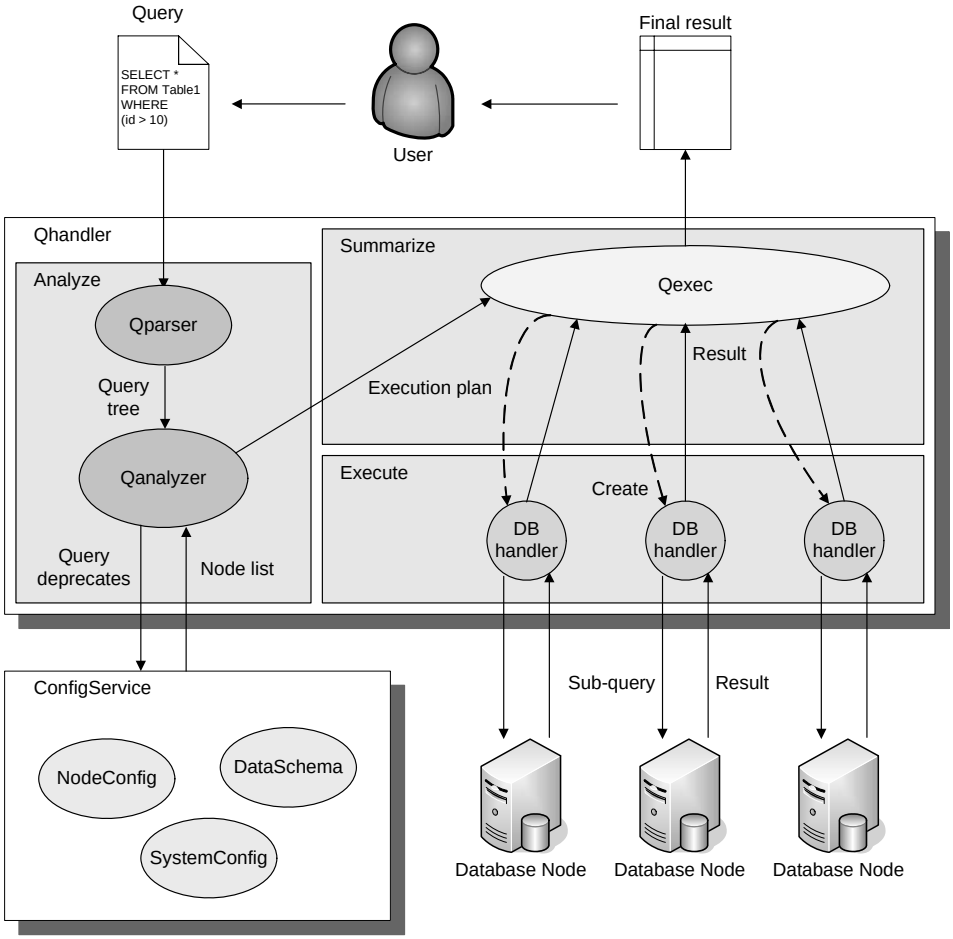


Figure 15 Parallel query execution process of PVDB

Qhandler executes a query by using three phases execution process: analyze, execute and summarize. Analyze phase extract needed information from the query and generates parallel execution plan to be used in the remaining phase. Execute phase takes execution plan from analyze phase and coordinates with database node to execute the plan. Summarize phase summarizes intermediate results from database nodes and returns the final result to the user.

Analyze phase is the first phase in the execution process. The main goal of this phase is to translate a SQL statement into parallel execution plan. First, Qparser (query parser) parses the SQL statement into a query tree and then passes it to Qanalyzer (query analyzer). Qanalyzer analyze the query tree and generates execution plan. It consults the ConfigService for list of nodes those should participate in the query. ConfigService is a PVDB service that provides configurations information for other components. Such configurations are NodeConfig, DataSchema and SystemConfig. NodeConfig describes properties of each database node such as address, DBMS type, processing rate, bandwidth, group membership, etc. DataSchema describes each relation stored in PVDB database. Configuration on each relation also includes data declustering information such as partition attributes, partitioning ranges and current placement of all fragments. PVDB currently support only one database. SystemConfig stores other system settings such as maximum concurrence queries, node selection policy, etc.

The execution plan is a list of tasks to complete the user query. A task is the sub-query to be executed on a database node. In a plan, tasks are ordered by an integer value called execution order to represent steps to complete the query. Execution order is starting from 0. Tasks with lower execution order are executed before those with higher value. Tasks those have the same execution order are executed in parallel. An example of basic execution plan is demonstrated in figure 16. This plan executes the same sub-query on all database nodes in parallel.

Execution Order	Sub-Query	Database Node
0	SELECT * FROM table1	0
0	SELECT * FROM table1	1
0	SELECT * FROM table1	2

Figure 16 Basic query execution plan

In execute phase, all tasks in execution plan are executed by Qexec. It selects all tasks those have lowest execution order from the plan and executes them in parallel. To execute multiple tasks in parallel, Qexec create a new thread for each task. Each thread then executes its assigned task by creating DBhandler object. DBhandler connects to specified database node in the task. After connection is established, DBhandler sends the sub-query to execute on specified database node and stored returned result in its memory buffer.

Summarize phase begins after results from all database nodes are returned. Qexec retrieves intermediate results from all DBhandler objects and summarizes them into final result. The summarization can be the basic data merging operation or applying aggregate functions such as summation, counting, etc. The final is returned to the user. The Qhandler releases all allocated resources and exit.

PVDB Design Limitation

Current design of PVDB assumes that PVDB operates in an organization that works independently from the others. Current PVDB design does not support organization that cooperates with other organization and shares their data together. This happen because the protocol that Front-end Node uses to communicate with Database Nodes are the product specific protocols. Such protocols are optimized for performance which usually has high bandwidth requirement and poor security. These problems are vital when operate in distributed environment. Furthermore, the Database Nodes should dispose to the outside network which leads to many security problems.

Improved PVDB Design

To eliminate the limitation discussed above, the PVDB architecture need to be redesigned to support distributed operating environment.

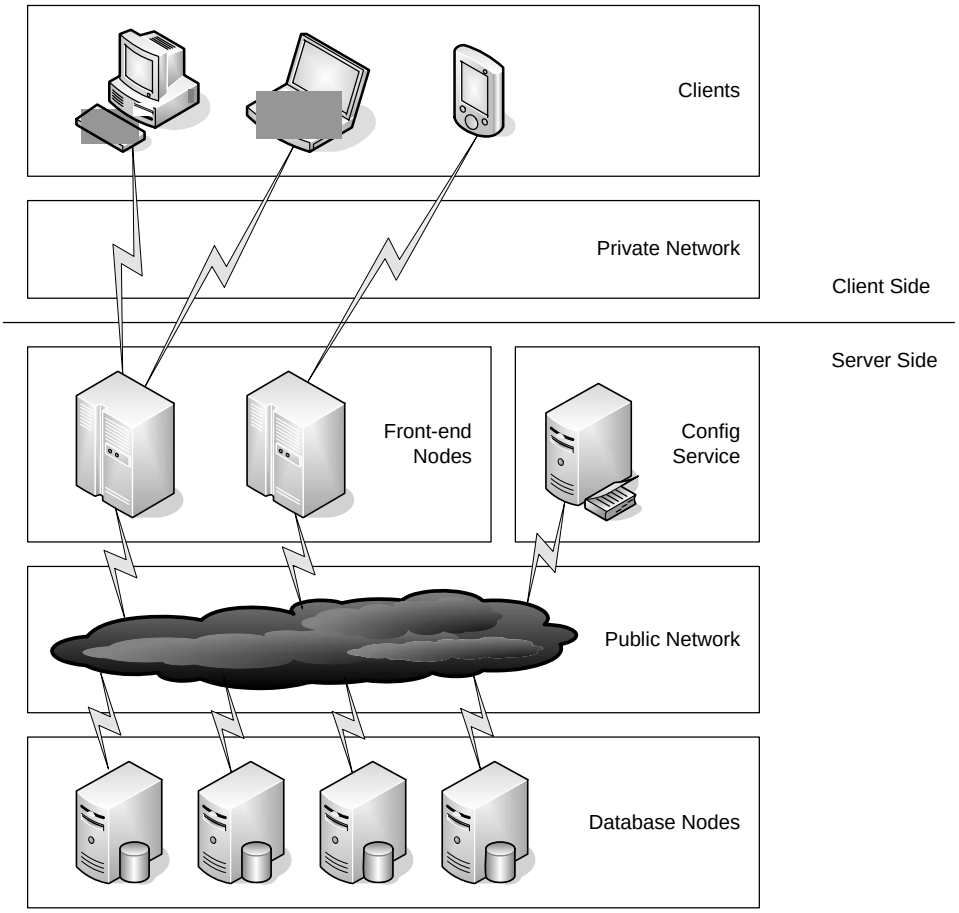


Figure 17 DPVDB system architecture

Figure 17 illustrate the new PVDB design called Distributed Parallel Virtual Database (DPVDB). There are three points those different from PVDB design: the

network point of view, multiple Front-end Nodes and dedicated Configuration Service.

In DPVDB design, the Private Network is the network that connects Clients to Front-end Nodes on the server side. The Public Network is the network that connects all Front-end Nodes and all Database Nodes together. This happened because DPVDB absolutely has different network point of view from PVDB. DPVDB thinks that the Database Nodes can be owned by different organizations and distributed over the public network such as the Internet while the Clients are workstations and devices within each organization those use their corporate network to access Front-end Nodes on their organization.

DPVDB now supports multiple Front-end Nodes in a system. The reason is to have at least one Front-end Node for each organization that joins DPVDB system. Clients on each organization can access to DPVDB only by the Front-end Nodes in its organization. This design intends to reduce complexity on centralize manage Clients and distinguish the network used by client side from one used by server side.

DPVDB dedicates a component called Config Service to host the Configuration Service software component. This resulted from having multiple Front-end Nodes in the system. However, DPVDB allows only one Config Service component in the system to eliminate replication issues.

DPVDB use the same software architecture as PVDB. However, all components in DPVDB should be implemented as web services. Web service is a standard mechanism used to call functions on remote system that widely used in distributed computing. Web service helps architectural design of DPVDB able to operate on distributed environment. Implementing components as web service also standardize the interaction mechanism among components in DPVDB.

Components those should be implemented as web services are Parallel Execution Coordinator, Configuration Service, System Status Service and Data Management. The implementation of Session Management and Client Library are automatically replaced by software in web service framework. The Session Management is replaced by web service container. The implementation of Client Library can be replaced by SOAP engine. So, any programming languages and frameworks those able to call web service are able to access DPVDB.

The web service implementation of Data Management is vital for DPVDB. It eliminates the main limitation that prevents PVDB from operating on distributed environment by replacing product specific protocols with web service standard. The Data Management web service is the wrapper service for DBMS on Database Nodes. The wrapper service virtualizes the DBMS operations so that each Database Node able to use different DBMS products.

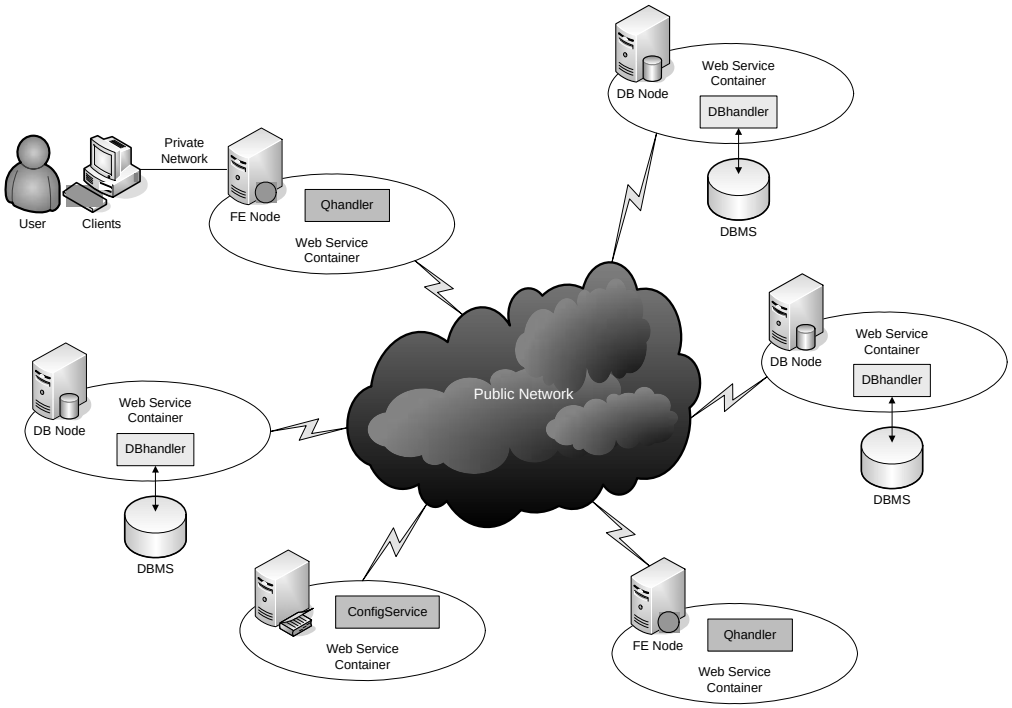


Figure 18 DPVDB software organization

Figure 18 demonstrates the software organization of a DPVDB system. Each Qhandler web service is hosted on a Front-end Node and uses web service container as its Session Management. ConfigSevice web service is hosted on a dedicate system. The DBhandler web services are hosted on Database Nodes to manage DBMS.

It is quite difficult to setup a real DPVDB system. A simulation system of DPVDB is necessary to evaluate the DPVDB design. The performance model of DPVDB system needs to be established in order to develop the simulation. The development process of such model is described in the following sections.

DPVDB Performance Model

The first step to establish performance model of DPVDB is identifying time consumed components those involve in query execution process. DPVDB has the same query execution process and PVDB which describe in previous section. The time consumed components are shown in figure 19.

In DPVDB, the query is sent from Clients to Front-end node over the Private Network. The Front-end Node preprocesses the query by using Qparser and Qanalyzer and passes control to Qexec to execute the query. Qexec coordinate with Database Nodes to prepare for the execution then send sub-queries to them over the Public Network. The Database Nodes then process the sub-query against its local data and return intermediate result back to Front-end Node over the Public Network. The Front-end Node summarizes all intermediate results and sends the final result back to the Clients over the Private Network.

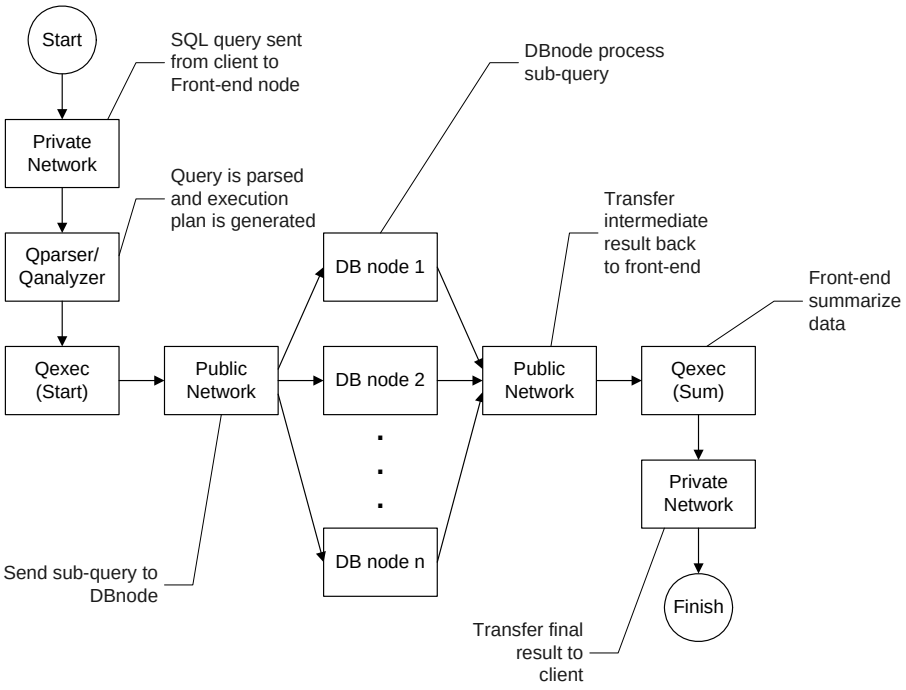


Figure 19 Time consumed components in execution process

However, the time used to transfer query from Clients to Front-end Node and from Front-end Node to Database Nodes are small relative to time used to process the query and transfer results. So, the time used to transfer query from Clients to Front-end Node is included into preprocessing time at the Front-end Node and the time used to transfer query to Database Nodes is included in parallel query starting time. The shortened execution process is depicted in figure 20.

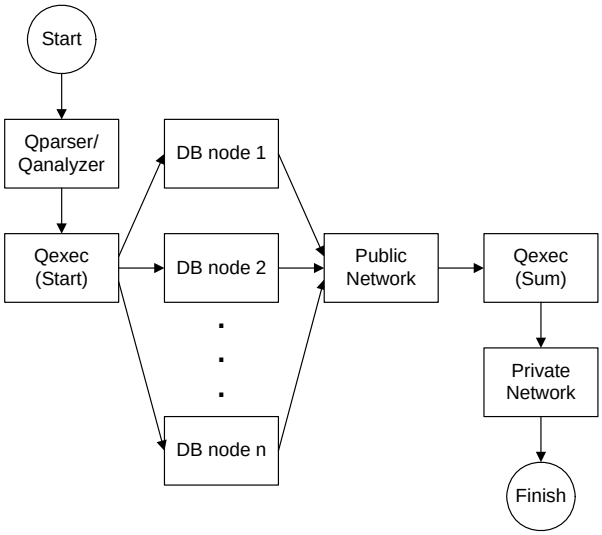


Figure 20 Shortened execution process

The notations those used in this modeling and their descriptions are listed in Table 1.

Table 1 Description of notations

Notation	Description
Q	A query that submitted to DPVDB
db_i	The Database Node i^{th}
fe_j	The Front-End Node j^{th} that handle the query Q
c_k	The Client k^{th} that submit the query Q
Q_i	The sub-query for database node i^{th}
result(Q)	The result of query Q
$T_{exec}(Q)$	Time used to execute query Q on DPVDB
$T_{exec}(Q,n)$	Time used to execute query Q on node n
$T_{analyze}(Q,n)$	Time used to analyze the query Q by node n
$T_{start}(n)$	Time used to start query execution on node n
$T_{pubnet}(D,src,dst)$	Time used to transfer data D from source src to destination dst over the public network
$T_{prinet}(D,src,dst)$	Time used to transfer data D from source src to destination dst over the private network
$T_{summarize}(D,n)$	Time used to summarize data D on node n
dbsize(n)	Size of data store on node n in records
procrate(n)	Data processing rate of node n in records per second
size(D)	Size of data D in bytes
bw(n)	Bandwidth of node n in bytes per second

The cost model of the shortened execution process in figure 20 can be expressed as shown in figure 21.

$$
\begin{aligned}
T_{exec}(Q) \;=\; & T_{analyze}(Q,\ fe_j) \\
& + \underset{i=1}{\overset{m}{MAX}} \left(T_{start}(db_i) + T_{exec}(Q_i, db_i) + T_{pubnet}\left(result(Q_i), db_i,\ fe_j\right)\right) \\
& + T_{summarize}\left(result(Q),\ fe_j\right) + T_{prinet}\left(result(Q),\ fe_j, c_k\right), \quad where\ 1 \le m \le n
\end{aligned}
$$

Figure 21 Cost model of DPVDB execution process

The total time used to process a query Q in DPVDB equals to summation of time to analyze the query, maximum query processing time of m selected Database Nodes from all n Database Nodes, time to summarize result and time to transfer result back to client. The query processing time on each database node is the summation of startup time, execution time, and intermediate result transfer time. The reasons for uses maximum query processing time is that selected database nodes independently process their query and front-end node waiting for the last intermediate result return before starting result summarization process.

The query analysis time and startup time are almost constant for each node and independent to current condition on the node because they are considered as lightweight process.

In contrast, the query execution time is varied which resulted from many parameters. Figure 22 expresses the cost model of query execution on a Database Node.

$$T_{exec}(Q,n) = \frac{dbsize(n)}{procrate(n)}$$

Figure 22 Cost model of query execution on a node

According to the model, the query execution time on a node equals to database size of that node divided by node processing rate. This model reflects behavior that node always scans all of its local data to when process a query. The varying parameter is the processing rate because sometime the node is heavily loaded so the process rate is shared among running queries. However, characteristics of the query do not take into account for current model.

The result summarization time has many factors like the query execution time but it has different data to process. Figure 23 express the cost model of result summarization process.

$$T_{summarize}(D,n) = \frac{size(D)}{procrate(n)}$$

Figure 23 Cost model of result summarization on a node

The model demonstrates that time to summarize data on a node equals to size of the data divided by node processing rate. This model also reflects the same behavior in data processing as query execution model. Node summarizes data by process always process all records.

The data transfer time on both Public Network and Private uses the same cost model as express in figure 24.

$$T_{network}(D, src, dst) = \frac{size(D) \times record\_size}{MIN(bw(src), bw(dst))}$$

Figure 24 Cost model of data transfer over network

The time used to transfer data over a network equals to size of data in bytes divided by minimum bandwidth between source node and destination node. In DPVDB the unit of data is always record. To calculate transfer time, data size should be converted into byte by multiple with record_size. The record_size is a system-wide constant that represent average size in byte of records stored in DPVDB. The bandwidth function (bw) is depended on network that model applying to. If the bandwidth function applies on source or destination that does not exist on applying network, the model does not define.

DPVDB Queuing Model

After establish cost model of DPVDB, the queuing model also need to be established in order to successfully develop the simulation. Figure 25 illustrates the queuing model of DPVDB

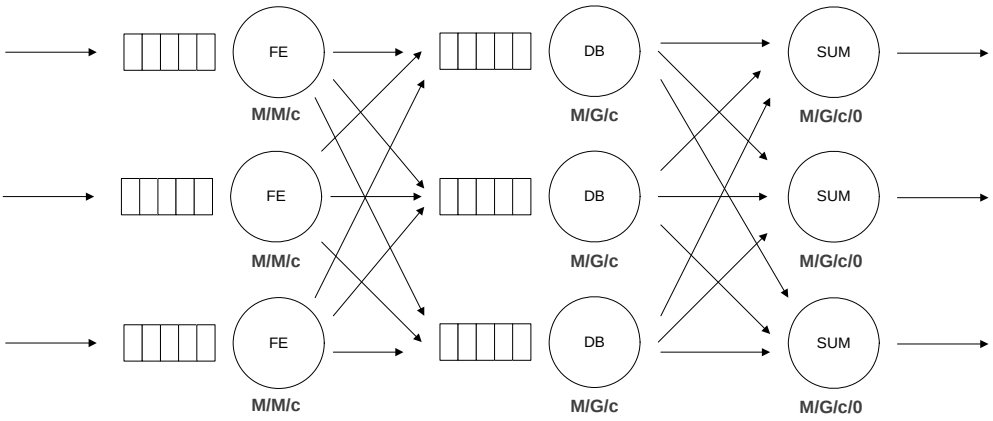


Figure 25 Queuing model of DPVDB

Queuing model of DPVDB is a network of queue that consists of three types of services: FE, DB and SUM. The FE is the service that customers arrive. It queuing notation is M/M/c because the customer interarrival time and its service time are memoryless. Memoryless means that service time for each customer independent from the others. Each FE service support maximum for c customers in the same time. A system can have unlimited number of FE service.

DB services receive customers after the FE services. A customer at a FE service may produce many customers to DB services. The queuing notation of DB service is M/G/c. The service time of DB service is general because DB services do not service each customer independently. A system can have unlimited number of DB service and do not necessary to equal to number of FE service.

SUM service receive customer from DB service. The number of SUM service equals to number of FE service and each FE service has its corresponding SUM service. All customers generated by a FE service must return to its corresponding SUM service after receive service from DB services. The queuing notation of SUM service is M/G/c/0. The SUM services also do not process the customers independently. They also have no queue space due to the number of customer is already controlled by FE service.

DPVDB Event Graph Diagram

The last thing that needed to establish before developing the simulator is designing of event graph diagram. The event graph diagram represents events those happened in the system, the event occurrence sequence and time between their occurrences. The event graph diagram for DPVDB is illustrated in figure 26.

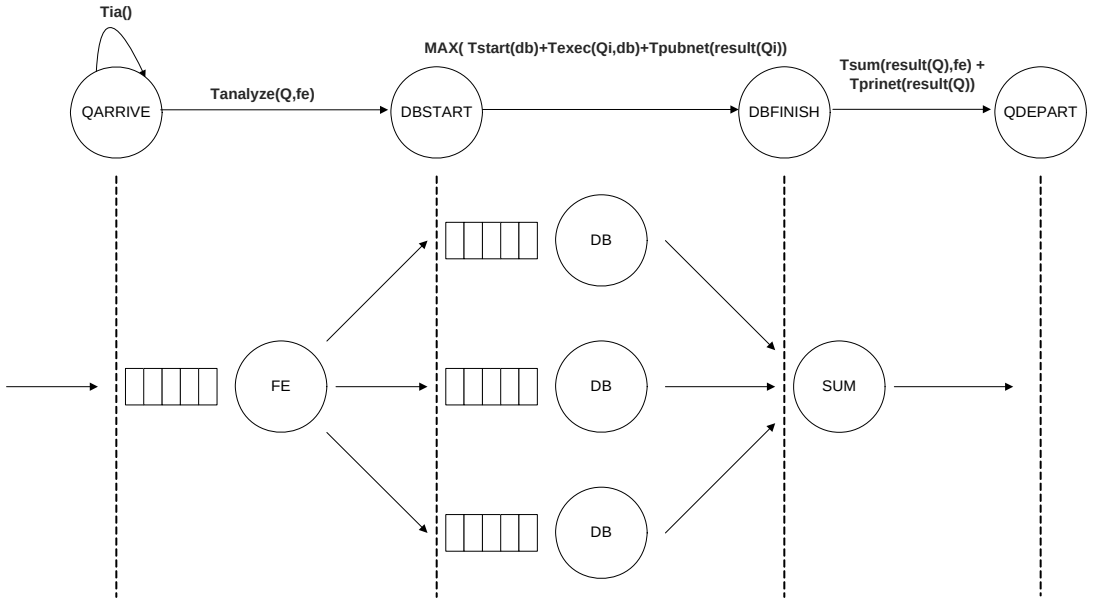


Figure 26 Event graph diagram of DPVDB

There are four events occurred in DPVDB system: QARRIVE, DBSTART, DBFINISH and QDEPART.

The QARRIVE is the event that a query arrives at a FE service. DBSTART is the event that the query arrives at each DB service. The time between QARRIVE event and DBSTART event is the time that query spent in FE service which equals to queue time plus analysis time. DBFINISH is the event that the last query generated by FE service reach its corresponding SUM service. The time between DBSTART and DBFINISH event equals to maximum time that the query spent in DB services. This means the queue time plus query execution time as described in previous section. The last event is QDEPART. It represent the event that query depart from the system. It

occurs after the DBFINISH in the time equal to result summarization time plus the transfer time.

DPVDB Simulation Program

A simulation program is developed to simulate behavior of a DPVDB system. The simulation program receives a set of parameters and produces results to output files. User can adjust some parameters and observe new results. The simulation program mainly uses to study performance behavior of DPVDB without need to setup a real system. Table 2 lists parameters those accepted by the simulation program along with their description.

Table 2 DPVDB simulation program parameters

Parameter Name	Description
Seed	Random seed for random number generator
log	Event logging level 0 – 4
num-fe	Number of front-end node
fe-prate	Processing rate of front-end node in records/sec
fe-max	Maximum concurrent running queries on front-end node
fe-bw	Bandwidth of front-end node on public network in bytes/sec
fe-lbw	Bandwidth on private network in bytes/sec
num-db	Number of database node
db-prate	Processing rate of database node in records/sec
db-max	Maximum concurrent running queries on database node
db-bw	Bandwidth of database node on public network in bytes/sec
db-st	Execution startup time in seconds
num-fgmt	Number of fragment
fm-size	Size of fragment in records
rec-size	Size of a record in bytes
ia-time	Interarrival time in seconds
qry-len	Size of a query in fragments
num-qry	Number of queries to observe results

The simulation program is developed under the framework of a simulation library called HyperSim (Phatanapherom et al., 2003). HyperSim is a library that uses to develop discrete event system simulation program. It requires the developer

completes the design of event graph diagram of the interesting system to successfully develop a simulation program. Basic facilities those HyperSim provides to developer are event scheduling and processing system, random number generator and data collector. The event scheduling and processing system functions are provided by EventListener class and its child classes. Random number generator function is provided by Distribution class and its child classes. The data collector function is provided by Monitor class. It also provides some of ready to use model for developer such as Grid, Cluster, Host, Utilization Queue, Capacity Queue, etc. Further information are available on HyperSim reference manual.

The DPVDB simulation program developed using Python scripting language. HyperSim is also available for Python. The implementation of the simulation program is illustrated in figure 27.

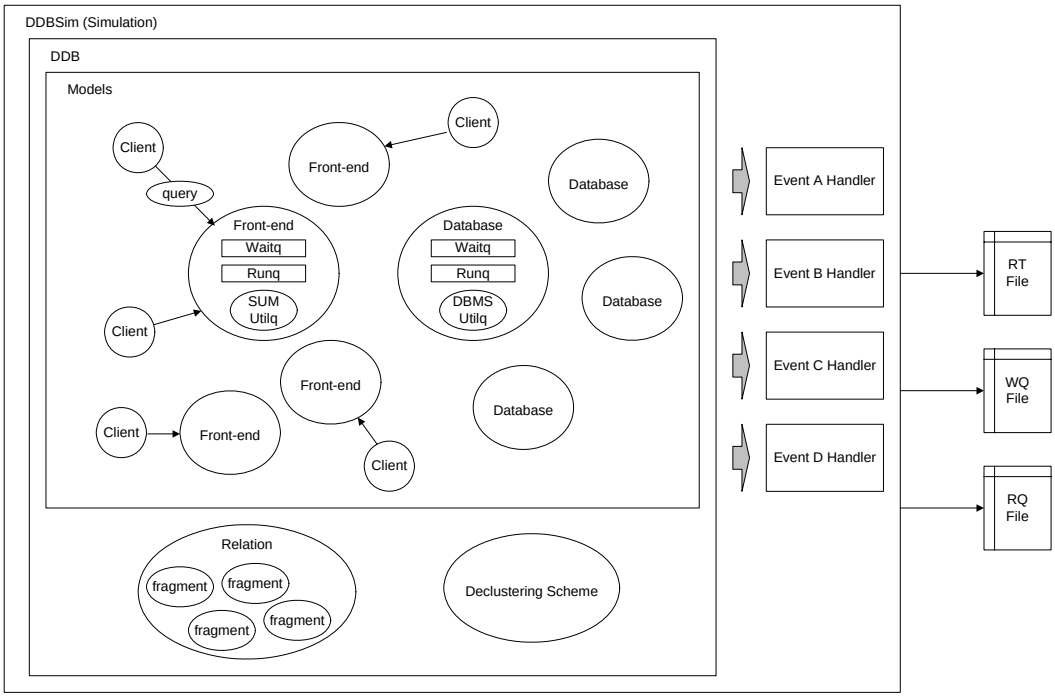


Figure 27 DPVDB simulation program implementation

The main component of the implementation is DDB. DDB is the container for collection of model object, relation object and declustering scheme object. Model objects are components those implement logics of DPVDB system components which are FrontendNode, DatabaseNode and Client. The FrontendNode implements logics of FE service and SUM service in the queuing model. The DatabaseNode implements logics of DB service in the queuing model. The Client implements the query generation logics. These components control the event scheduling for the simulation. The Relation is the component that keeps information about data stored in DPVDB. The Declustering Scheme object assigns placement for each fragment, selects DatabaseNode object to be involved in the query execution and perform other decision making tasks for the simulation. DDBSim is the component that observes

statistic information of the simulation by listen for events those occurred within the DDB and record static data on to the output files.

All FrontendNode and DatabaseNode objects contain three important objects: running queue, waiting queue and UtilizationQueue model. The running queue is a vector use to keep queries those being processed by UtilizationQueue object. The waiting queue is a vector use to keep queries those waiting to process by UtilizationQueue. The UtilizationQueue object is a HyperSim ready to use model that implement time shared service. Time shared service is the service that the servers in the service share their processing time among servicing customers. In FrontendNode object, the UtilizationQueue implements general service time for the SUM service. In DatabaseNode object, the UtilizationQueue implements general service time for DB service.

There are three output file from the DPVDB simulation program: RT file, RQ file and WQ file. All of these files are written in comma separated value (CSV) format. The RT file records response time of each queries. RQ file records length of running queue on all nodes in the simulation. WQ file records length of waiting queue on all nodes in the simulation.

RESULTS AND DISCUSSION

PVDB Implementation

The performance evaluation of the proof of concept PVDB implementation performs by using a special client program called PVDBBench. PVDBBench is a client program that able to simultaneously send multiple queries to PVDB server. In this evaluation, PVDBBench is configured to send 100 of test queries to server by using 10 threads. This evaluation is performed against a PVDB server system that configured to have one database node to five nodes. The time that uses to process all of 100 queries, is recorded for each number of database nodes in PVDB system. The recorded processing times are used to calculate speedup ratio compared with processing for standalone database system.

The PVDB system use in this evaluation is a six nodes PVDB system. Each node has dual Pentium4 1.8 GHz processors with 1 Gigabyte memory, 80 Gigabytes hard disk and a Fast Ethernet adapter. One of node is assigned to be the front-end node. All remaining nodes are assigned to be database nodes. All database nodes use MySQL version 4.0 as their DBMS and are loaded with a database table with 10,000,000 data records. The front-end node and all database nodes are connected together using a 16 ports Fast Ethernet switch.

One of database node is used as standalone database system. This system is used as baseline to calculate speedup ratio with PVDB.

The client workstation use in this evaluation is a personal computer with Pentium4 1.8 GHz processor with 1 Gigabyte memory, 80 Gigabytes hard disk and a Fast Ethernet adapter. The client workstation and front-end node are connected using another 16 ports Fast Ethernet switch.

The test results are listed in table 3.

Table 3 Total processing time for each system configuration

	Elapsed Time (seconds)	Query Rate (query per second)
Standalone System	376.09	0.2659
PVDB with 1 database node	373.37	0.2678
PVDB with 2 database nodes	194.13	0.5151
PVDB with 3 database nodes	133.92	0.7523
PVDB with 4 database nodes	106.38	0.9400
PVDB with 5 database nodes	97.24	1.0284

The plot of elapsed times in table 3 is illustrated in figure 28.

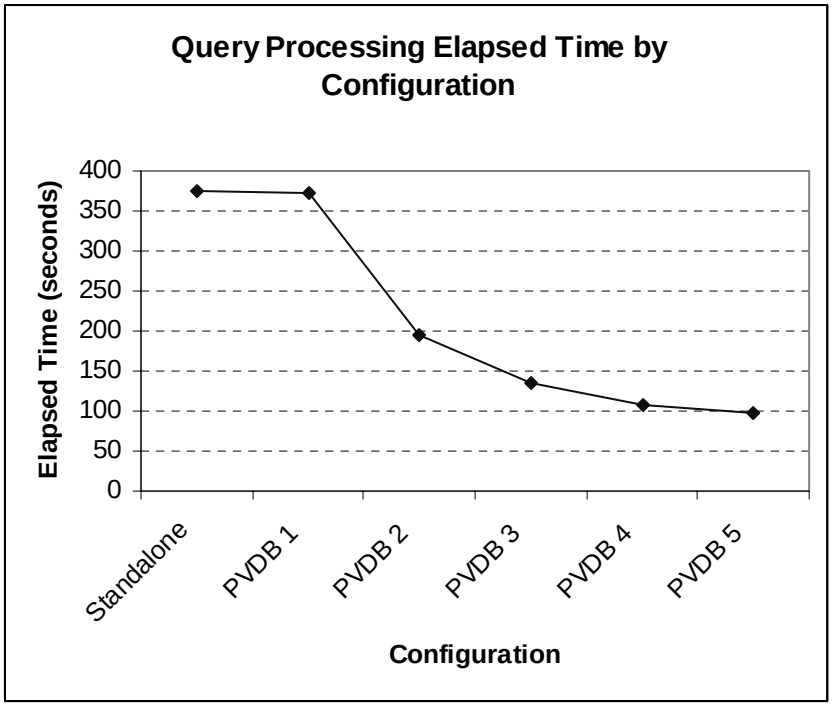


Figure 28 Elapsed time for each system configuration plots

The figure 28 shows that adding more database nodes to the system will decrease elapsed time. The system is able to process the same problem size using shorter amount of time. This means that system performance is increased as increasing number of database nodes.

The speedup ratio for each PVDB configuration is calculated by dividing elapse time of each configuration with elapse time of standalone system. The calculate results are shown in table 4.

Table 4 Speedup ratio for each system configuration

PVDB Configuration	Speedup
1 database node	1.01
2 database nodes	1.94
3 database nodes	2.81
4 database nodes	3.54
5 database nodes	3.87

The plot of elapsed time table 4 is illustrated figure 29.

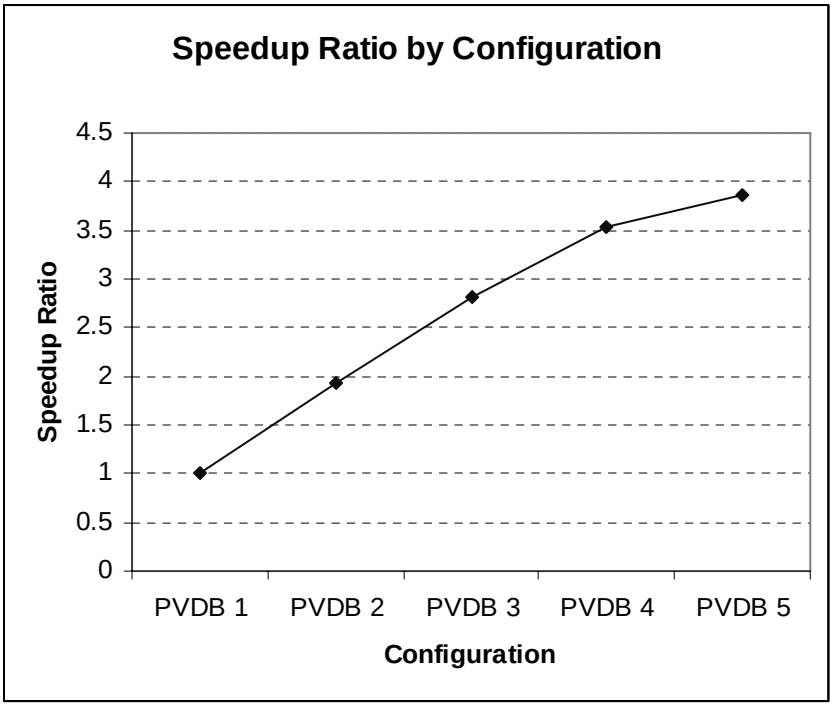


Figure 29 Speedup for each system configuration plots

In figure 29, the system gains more speedup as increasing database node to the system. However, the speedup is increased at fewer rates as increasing number of database nodes. This happens due to overhead in parallel query execution that depends on number of database node in the system.

DPVDB Design

In this section, the design of DPVDB is evaluated by using a simulation program. The simulation program simulates behavior of DPVDB design as described in previous sections. There are two studies in this evaluation. The first is the study of DVPDB design behavior when in heavily loaded condition. The second is the study of scalability of the design with different number of database node.

The evaluating DPVDB system that uses for both studies is illustrated in Figure 30. It has one Front-end Node and a number of Database Node. The number of database node is the first study is ten but it varies from 1 to 500 in the second study. All nodes are have the same processing rate and maximum capacity. A client sends queries to the Front-end Node. Both Public Network and Private Network are Fast Ethernet which has bandwidth of 100 Mbps.

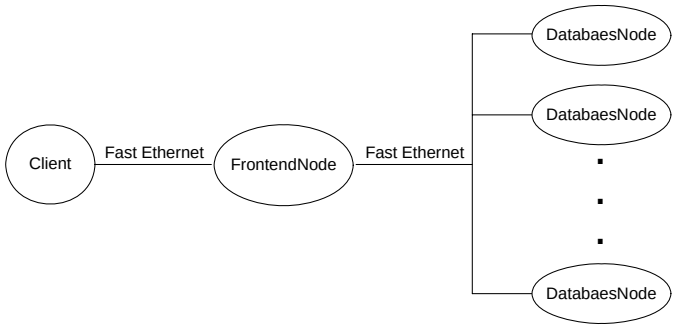


Figure 30 Evaluating DPVDB system

Design Behavior Study

For the first study, a set of parameter called base case is defined to load the evaluating system as the base line. Then the system is loaded by the heavy load test case which has shortened interarrival time of 60 seconds. All simulation parameters of both test cases are listed in table 5.

Table 5 Behavior study simulation settings

Parameter Name	Base Case	Heavy Load Case
seed	2549	2549
num-fe	1	1
Fe-prate	c(100000)	c(100000)
fe-max	100	100
fe-bw	e(12800000)	e(12800000)
fe-lbw	e(12800000)	e(12800000)
num-db	10	10
db-prate	c(100000)	c(100000)
db-max	100	100
db-bw	e(12800000)	e(12800000)
db-st	e(5.0)	e(5.0)
num-fgmt	1000	1000
fm-size	c(100000)	c(100000)
rec-size	1024	1024
ia-time	**e(120)**	**e(60)**
qry-len	c(10)	c(10)
num-qry	10000	10000

The simulation results of the base case are illustrated in figure 31 and figure 32. Figure 31 shows the response time. Figure 32 also shows running queue length on the Front-end Node.

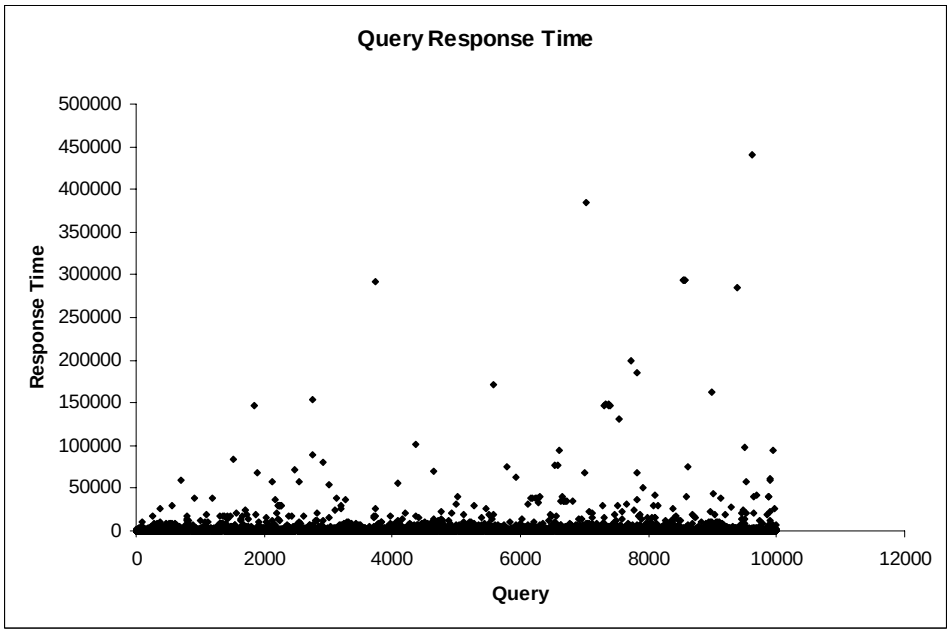


Figure 31 Response time plots of base case

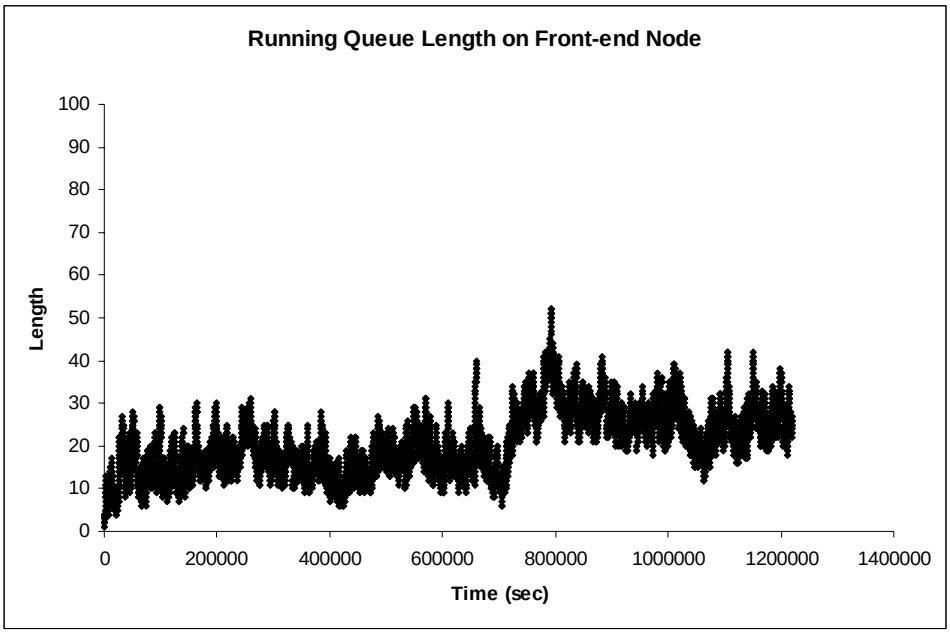


Figure 32 Running queue length on Front-end Node plots of base case

In base case, the response times of 10000 queries are quite equal. The average response time of base case is 2147.38 seconds. The number of running queries on the Front-end node is moving within range of maximum capacity of 100. The number of running query at Front-end Node represents the number of query running in the system because there is only one Front-end Node.

The simulation results of heavy load case are illustrated in figure 33 and figure 34. Figure 33 shows the plots of queries response time. Figure 34 shows the plots of running queue length on Front-end Node.

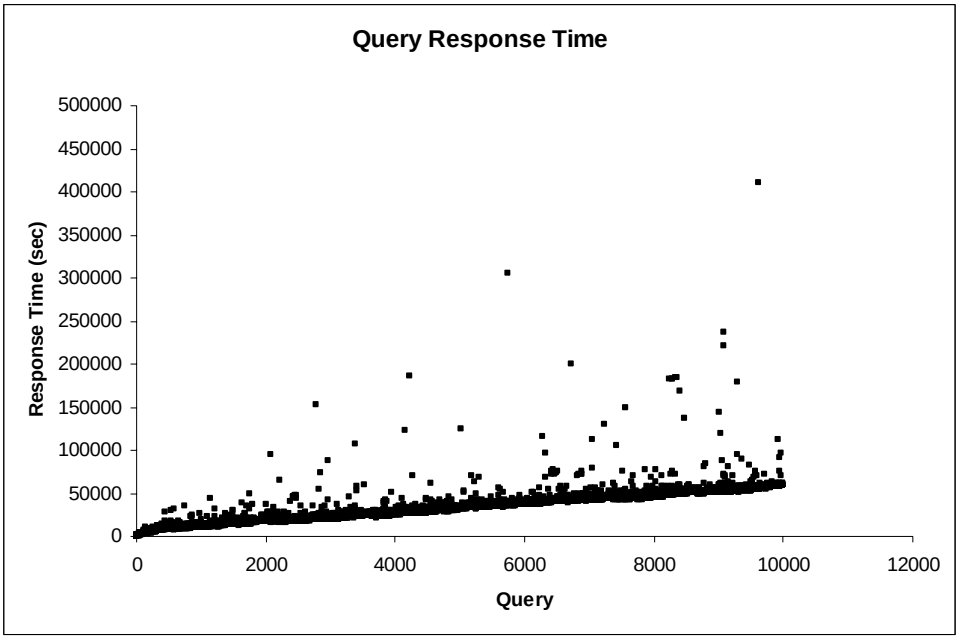


Figure 33 Queries response time plots of heavy load case

In the heavy loaded case, the query response times are increasing through simulation period. It increases exceed 50000 seconds at the end of simulation. The average query response time is 33609.46 seconds that is more than 9 hours which is unacceptable response time for most users. This happened due to effects of queuing. As depicted in figure 34, the number of running queries on Front-end Node reaches maximum capacity at very beginning of the simulation. In this situation, all incoming queries have to wait in waiting queue until all previously queued queries are executed. So, the response time of queued queries equal to sum of waiting time and execution time.

Let take a look at maximum running queue length on all Database Nodes in figure 35. The number of running query on Database Nodes is never exceeding the maximum capacity. All queries those reach Database Nodes are immediately executed. This implies that the Front-end node is bottleneck when the system is heavily loaded. Adding additional Front-end Nodes to the system will solve this problem.

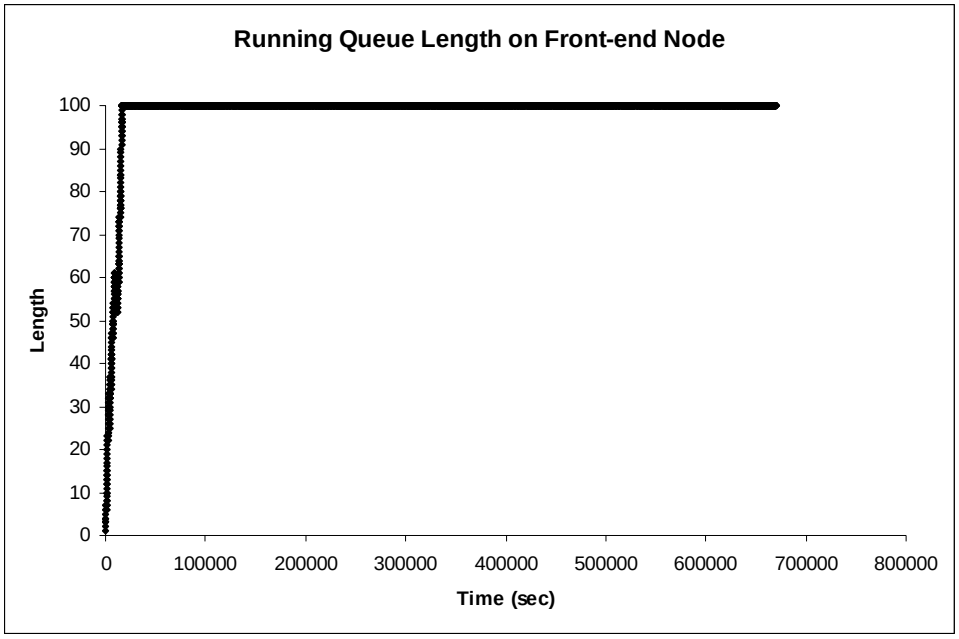


Figure 34 Running queue length on Front-end Node plots of heavy load case

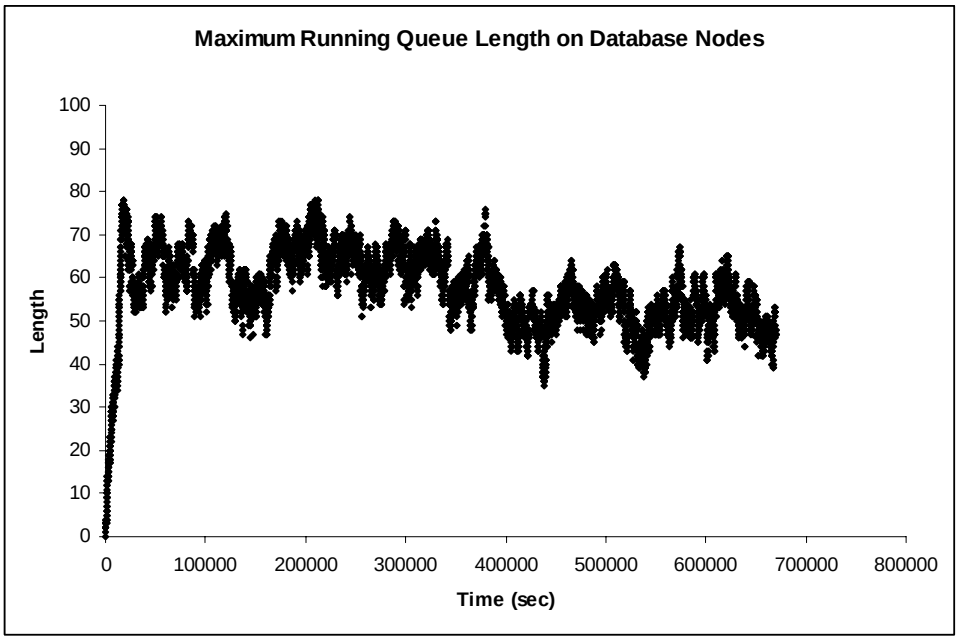


Figure 35 Maximum running queue length on all Database Nodes plots

Design Scalability Study

In the second study, the evaluating system with different number of Database Node uses to execute 10000 queries. The numbers of Database Node use in this study are 1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, and 500 nodes

respectively. All other settings are the same as base case of the first study. The plots of average response time results are illustrate in figure 36. Average query response time number of database node

Table 6 Average query response time by number of Database Node

Number of Database Node	Average Response Time (seconds)
1	4450725.51
2	1922000.50
3	1056308.14
4	587345.42
5	300194.91
10	2147.38
20	1580.78
30	1603.11
40	1638.36
50	1637.14
60	1687.14
70	1528.21
80	1495.59
90	1544.84
100	1533.85
200	1471.04
300	1480.89
400	1459.22
500	1446.98

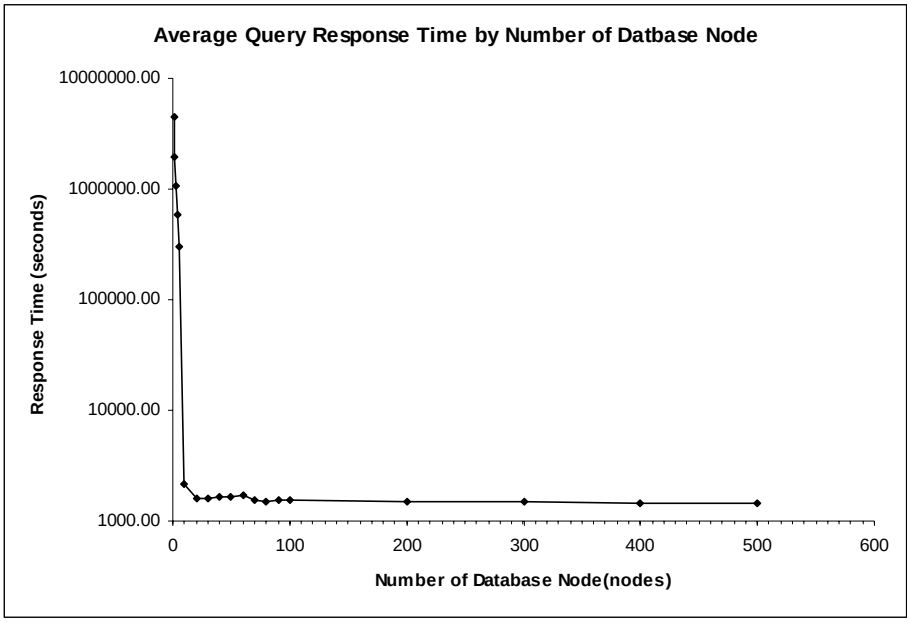


Figure 36 Average query response time by number of Database Node plots

As depicted in figure 36, the average query response time of a DPVDB system is decreased as the number of Database Node increasing. The average response time of system with one Database Node is 4450725.51 seconds which definitely unacceptable time (about 24 days). When a new database node added, it significantly decreases average response time to 1922000.50 seconds which less than the half of one Database Node system. This can conclude that adding new nodes to a DPVDB system will increase its performance.

However, when the number of Database Node is 10, adding more Database Node to the system does not significantly increase the system performance. The simulation results show that average response time of the system with 500 Database Nodes is less than one with 10 Database Nodes only about 600 seconds. This happened because benefits of decreasing execution time from adding more Database Nodes offset by overhead of parallel query execution.

The simulation program can be used as scalability planning tool. It helps system engineer to find economy of scale number of Database Node as in the second study experiment.

CONCLUSION AND FUTURE WORK

This thesis proposes design of a new parallel database system called Parallel Virtual Database or PVDB. PVDB design based on shared-nothing architecture which no shared resources among processors in the system. A proof-of-concept implementation of PVDB is developed. The implementation utilizes database management system as its data management layer and implements only parallel execution layer. The implementation is evaluated by using a special client program to send queries to a PVDB system with different number of Database Node. The implementation achieves speedup ratio of 3.8 for 5 Database Nodes.

The improved design of PVDB that concerns distributed environment in its design called Distributed Parallel Virtual Database or DPVDB. The design of DVPDB is evaluated by a simulation program that developed using HyperSim simulation library. A performance model and queuing model of the design of DPVDB are defined in order to develop the simulation program. The simulation program is used to evaluate the design in two studies: system behavior under heavily loaded and system scalability. The simulation results tell that the Front-end Node is the bottleneck when system is heavily loaded and adding more Database Node to the system will increase system performance but at limited scale. The simulation program can be used to predict the economy of scale number of Database Node. It can also use to predict system behavior for certain configuration before real implementation.

We can also conclude that this kind of technology can be applied effectively to handle massive amount of data. With the ever decreasing cost of commodity hardware and open source software, this kind of architecture is very useful for modern enterprise computing environment.

LITERATURE CITED

Bhatia, R., R.K. Sinha and C. Chen. 2000. Declustering using golden ratio sequences. pp. 271-280. **Proceedings of the 16th International Conference on Data Engineering.** San Diego, California, USA

Chen, C., R. Bhatia and R.K. Sinha. 2003. Multidimensional Declustering Schemes Using Golden Ratio and Kronecker Sequences. pp. 659-670. **IEEE Transactions on Knowledge and Data Engineering.** vol. 15(3). IEEE Computer Society Digital Library

Copeland, G.P., W. Alexander, E.E. Boughter and T.W. Keller. 1988. Data Placement In Bubba. **Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data.** Chicago, Illinois, USA

DeWitt, D.J. and J. Gray. 1992. Parallel Database Systems: The Future of High Performance Database Systems. pp. 85-98. **Communications of the ACM.** vol. 35(6). Association for Computing Machinery, Inc.

Du, H. and J. S. Sobolewski. 1982. Disk Allocation for Cartesian Product Files on Multiple-Disk Systems. pp. 82-101. **ACM Transactions of Database Systems.** vol. 7(1). Association for Computing Machinery, Inc.

Faloutsos, C. and P. Bhagwat. 1993. Declustering Using Fractals. pp. 18-25. **Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems (PDIS 1993), Issues, Architectures, and Algorithms.** San Diego, CA, USA

Feelifl, H. and M. Kitsuregawa. 2000. Heat Balancing for Btree Indexed Database over Ring Configured Shared Nothing Machines. **DEWS2000.** Japan

_____, _____ and B.Ooi. 2000. A fast Convergence Technique for Online Heat balancing of Btree Indexed Database over Shared-nothing Parallel Systems. pp. 846-858. **Proceedings of the 11th International Conference on Database and Expert Systems Applications.** London/Greenwich, UK

Ghandeharizadeh, S. and D.J. DeWitt. 1990. A Multiuser Performance Analysis of Alternative Declustering Strategies. pp. 466-475. **Proceedings of the Sixth International Conference on Data Engineering.** Los Angeles, California, USA

_____ and _____. 1990. Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines. pp. 481-492. **Proceedings of 16th International Conference on Very Large Data Bases.** Brisbane, Queensland, Australia

Kim, M. and S. Pramanik. 1988. Optimal File Distribution For Partial Match Retrieval. pp. 173-182. **Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data.** Chicago, Illinois, USA

Lo, Y., K.A. Hua and H.C. Young. 2001. GeMDA: A Multidimensional Data Partitioning Technique for Multiprocessor Database Systems. pp. 211-236. **Distributed and Parallel Databases.** vol. 9(3). Springer Science+Business Media B.V.

Mehta, M. and D.J. DeWitt. 1997. Data Placement in Shared-Nothing Parallel Database Systems. pp. 53-72. **The VLDB Journal.** vol. 6(1). Very Large Data Base Endowment Inc

Phatanapherom, S., P. Uthayopas and V. Kachitvichyanukul. 2003. Fast Simulation Model for Grid Scheduling using HyperSim. pp. 1494-1500. **Proceeding of the Winter Simulation Conferrence.** New Orleans, Louisiana, USA

Prabhakar, S., K.A.S. Abdel-Ghaffar, D. Agrawal and A.E. Abbadi. 1998. Cyclic Allocation of Two-Dimensional Data. pp. 94-101. **Proceedings of the Fourteenth International Conference on Data Engineering.** Orlando, Florida, USA

Scheuermann, P., G. Weikum and P. Zabback. 1998. Data Partitioning and Load Balancing in Parallel Disk Systems. pp. 48-66. **The VLDB Journal.** vol. 7(1). Very Large Data Base Endowment Inc

Zhou, Y., S. Shekhar and M. Coyle. 1994. Disk Allocation Methods for Parallelizing Grid Files. pp. 243-252. **Proceedings of the Tenth International Conference on Data Engineering.** Houston, Texas, USA

CURICULUM VITAE

NAME:	Mr. Suphachan Phakhawirotkul		
BIRTH DATE:	April 14, 1978		
BIRTH PLACE:	Bangkok, Thailand		
EDUCATION:	YEAR	INSTITUTION	DEGREE/DIPLOMA
	1999	Thammasat Univ.	B. Sc. (Computer Science)
POSITION/TITLE:	Lab Manager		
WORK PLACE:	High Performance Computing and Networking Center, Faculty of Engineering, Kasetsart University		