A Hybrid Shared-nothing/Shared-data Storage Scheme for Large-scale Data Processing

Huaiming Song [†], Xian-He Sun [†], Yong Chen [‡]

[†]*Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616, USA*
[‡]*Department of Computer Science, Texas Tech University, Lubbock, TX 79409, USA*
huaiming.song@iit.edu, sun@iit.edu, yong.chen@ttu.edu

***Abstract*—Shared-nothing and shared-disk are the two most common storage architectures of parallel databases in the past two decades. Both two types of systems have their own merits for different applications. However, there are no much efforts in investigating the integration of these two architectures and exploiting their merits together. In this paper, we propose a novel hybrid storage architecture for large-scale data processing, to leverage the benefits of both shared-nothing and shared-disk architectures. In the proposed hybrid system, we adopt a shared-nothing architecture as the hardware layer and leverage a parallel file system as the storage layer to combine the scattered disks on all database nodes. We present an overall design of the new scheme, including data and storage organization, data access modes, and query processing methods. The proposed hybrid scheme can achieve both high I/O performance as a shared-nothing system, and high-speed data sharing across all server nodes as a share-disk system. Preliminary experimental results demonstrate that the hybrid scheme is promising and more appropriate for large-scale and data-intensive applications than each of the two individual types of systems.**

***Keywords*-storage architecture, shared-nothing, shared-data, parallel databases, parallel file systems**

I. Introduction

Shared-nothing and shared-disk [1][2][3] are the two widely-used storage architectures in parallel databases. Both two architectures have their own positive and negative features, but neither of them has the edge on the other in all aspects. Figure 1 shows a sketch of the two storage architectures. In a shared-nothing system, the data set is usually partitioned [1][4][5] into several subsets and each node keeps one subset in its native disks. Generally, the shared-nothing systems provide a high degree of parallelism for both I/O and computing [1]. Nevertheless, the shared-nothing systems also have multiple nodes transaction [6][7], data shipping [5][8], and data skew issues [4][5][8]. In a shared-disk system, data is stored in a large centralized storage, which is accessible by all database nodes. Since every node has direct access to all disks, there is no need of data partitioning according to the number of nodes, which eliminates the data skew problem. The main drawbacks of shared-disk systems, however, are low I/O bandwidth and poor scalability.

In an early stage of parallel database, the bandwidth and latency of network is worse than accessing data to local disks, hence it is natural to avoid accessing data from a remote disk through network. For that reason, the shared-nothing and the shared-disk architectures were presented separately. However, network transmission bandwidth grows rapidly during the past years. Nowadays, reading data from remote disks is no longer limited by network in a cluster environment. The tradeoffs between shared-disk and shared-nothing storage architectures need to be reevaluated. On the other hand, in the domain of data-intensive scientific computing, parallel file systems, such as Lustre[9], PVFS2[10], and GPFS[11], are widely used for high I/O performance. Compared with parallel databases, parallel file systems are shared-nothing structured (as shown in Figure 1), but they also unite all disks on multiple nodes and provide a single namespace.

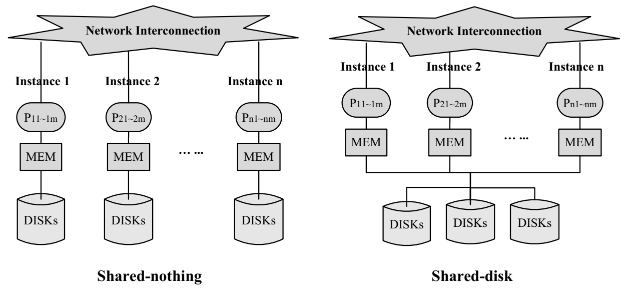


Fig. 1. Shared-nothing and Shared-disk Parallel Databases

Inspired by the design of parallel file systems, we propose a novel hybrid storage architecture for large-scale parallel databases, by integrating parallel database with parallel file system techniques together. It adopts shared-nothing for the upper layer database instances, but also provides data sharing through a lower-layer parallel file system. Therefore, the proposed hybrid system can work both as a shared-nothing system if each database node only accesses data on its own disks, and as a shared-disk system if the database nodes access data from the global namespace of the parallel file system.

The contribution of this study is three-fold. First, we argue that combining shared-nothing and shared-data in parallel database systems, would be greatly beneficial for many emerging data-intensive applications with the rapid advance of interconnection technology and swift development of parallel file systems. Second, we propose an innovative hybrid storage architecture for parallel databases to carry the combination and to have the merits of both two architectures. We introduce the design of the proposed storage scheme, including data access modes, data organization, and query processing methods. Finally, we have conducted extensive experimentation to verify the proposed storage design. The results demonstrate that the hybrid system is promising in leveraging the advantages of both shared-nothing and shared-data architectures and has a real potential in parallel database architectures.

The remainder of this paper is organized as follows. Section II briefly reviews the related work in parallel database architectures and parallel file systems. Section III describes the system design of the proposed hybrid shared-nothing/shared-data storage scheme. In Section IV we describe query and transaction processing methods. Verification experiments of the hybrid system are presented in Section V. Finally, Section VI concludes this study and discusses future work.

II. Related Work

A. Shared-nothing and Shared-disk

In shared-nothing systems, data is partitioned and distributed across all the database nodes [1][4][5]. DeWitt and Gray [1] introduced three well known partitioning methods: round-robin, range and hash. Besides, there are some combined strategies on multiple columns. When a query comes, it will be routed to the appropriate database nodes according to the data partitioning scheme. Complex queries, e.g. 'multi-join' or 'nested', are often executed as recursive computing and re-distributing phases [1][12] on multiple database nodes. It is unlikely to find a universal partitioning strategy suitable for all query types. As dataset and query pattern vary with the time, a good partitioning scheme in the past may become sub-optimal, resulting in load imbalance and performance degradation. Dynamic or adaptive algorithms [4][5][13] were introduced to deal with the load balance problems. Nowadays, there are numerous database clusters using the shared-nothing architecture, such as IBM DB2 UDB[14], Mysql Cluster [15], and Teradata products[16], etc.

In shared-disk systems, data is stored in shared disks, and every node has full access to the entire data [1]. There is no need to partition data across multiple database nodes, which eliminates the data skew problems. During query processing, the collaboration of different nodes relies on inter-nodal message and shared data access. Data sharing between memories on different nodes can deliver a superior performance in shared-disk system [3][17]. Cache fusion is a memory-sharing technique applied by Oracle RAC version [18], which can largely improve the query performance, because the high transmission speed and low latency of network make it much faster than data write back to shared disk and re-read courses. Currently, the shared-disk systems usually adopt storage area network (SAN) to provide high I/O performance, and it is costly due to dedicated hardware.

B. Parallel and Distributed File Systems

Parallel file systems, such as Lustre[9], PVFS2[10], and GPFS[11], are widely used in large-scale and data-intensive applications, to provide high I/O capabilities to high-performance computing (HPC) clusters. Normally, parallel file systems provide high I/O performance by striping data files over multiple storage nodes, and accessing these data strips in parallel. I/O clients can access files by logic address as in a single namespace, without the knowledge of physical layout. The transparency of data block placement is a convenient feature for users. Google file system (GFS)[19] and Hadoop distributed file system (HDFS)[20] are scalable distributed file systems for large distributed web search engine applications. Both GFS and HDFS are designed with big file chunks (chunk size is 64MB) and MapReduce [21] programming model. The write-once-read-many data access manner means no data modifications, in addition, MapReduce programming model is not designed for low latency. For those reasons, GFS and HDFS are not suitable for general purpose parallel databases.

There are some research efforts in integrating database and parallel file system technologies. Some scientific computing systems [22] employ databases for metadata management and parallel file systems for data storage respectively. Hive[23] and HBase[24] are built on HDFS and based on MapReduce model. They are designed for special purpose, and the query types and query processing manners are different from traditional databases. In addition, they are not suitable for low-latency applications.

The proposed hybrid architecture is different from others work. It is designed for general parallel databases. It has a shared-nothing design in the hardware layer, and provides data sharing through the underlying parallel file system. It has the merits of both shared-nothing and shared-disk parallel databases.

III. Hybrid Database System

A. System Architecture

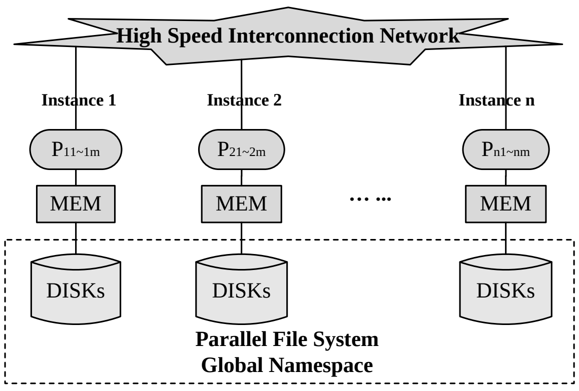


Fig. 2. Architecture of the Hybrid Parallel Database Storage Scheme

The proposed scheme adopts shared-nothing hardware interconnection manner, but uses a parallel file system to unite all scattered disks to provide data sharing capability. Figure 2 illustrates the system architecture of the proposed hybrid storage scheme, which can leverage the advantages of both the shared-nothing hardware structure and the shared-data facility

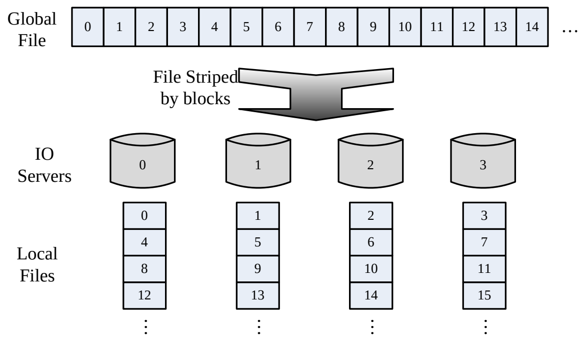


Fig. 3. Data Layout in Parallel File Systems

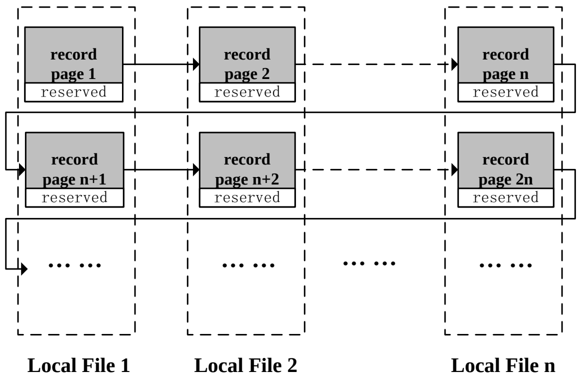


Fig. 4. File Striped by Record Pages in Hybrid Systems

of parallel file systems. In the proposed hybrid database, each node runs a database instance, and serves as both a parallel file system I/O server and an I/O client. Therefore, all database instances can access data in the parallel file system, namely data can be shared among different nodes conveniently. In a word, it is shared-nothing in hardware layer, but it is also a shared-data system.

Usually, in parallel file systems, data is striped across multiple I/O servers in a round-robin way, thus one file is divided into several sub-files on these servers. In the proposed hybrid system, one file in parallel file system is called global file, and a sub-file on one node is called local file. Thus, a global file is mapped into a set of local files, as shown in Figure 3. We introduce two data access modes in the proposed hybrid systems: local mode and global mode.

- **Local Mode**: each database node accesses data via a local file descriptor on its native disks.
- **Global Mode**: each database node accesses data via a global file descriptor in the global namespace.

Data access pattern can be either local mode or global mode, by calling read/write functions on a local file descriptor or a global file descriptor respectively. Which data access mode to use for query execution is determined by system query optimizer, based on the costs of query execution of the two modes. With high-speed interconnection techniques, the proposed hybrid scheme can obtain the advantages of both shared-nothing and shared-disk systems.

B. Data Organization

In relational databases, data files are composed of record pages. We design the stripe size in parallel file systems the same as the record page size in the hybrid system, thus database tables are naturally striped across all database nodes. Figure 4 shows the organization of the relational tables in the proposed hybrid system. Each table can be stored as a global file, which is striped by record pages. Hence each local file is a subset of original table, which could be regarded as a sub-table for each database node. Especially, record assigned across pages is not recommended, because it would lead to complex data access behaviors like data shipping and distributed locks. For that reason, we reserve a small area in each page, which is also useful for updating records with variable lengths. The percentage of reserved area in a record page is configurable, like that in current commercial databases. With the design of record page striping and reserved area, each node can be regarded as a standalone sub-database in local data access mode.

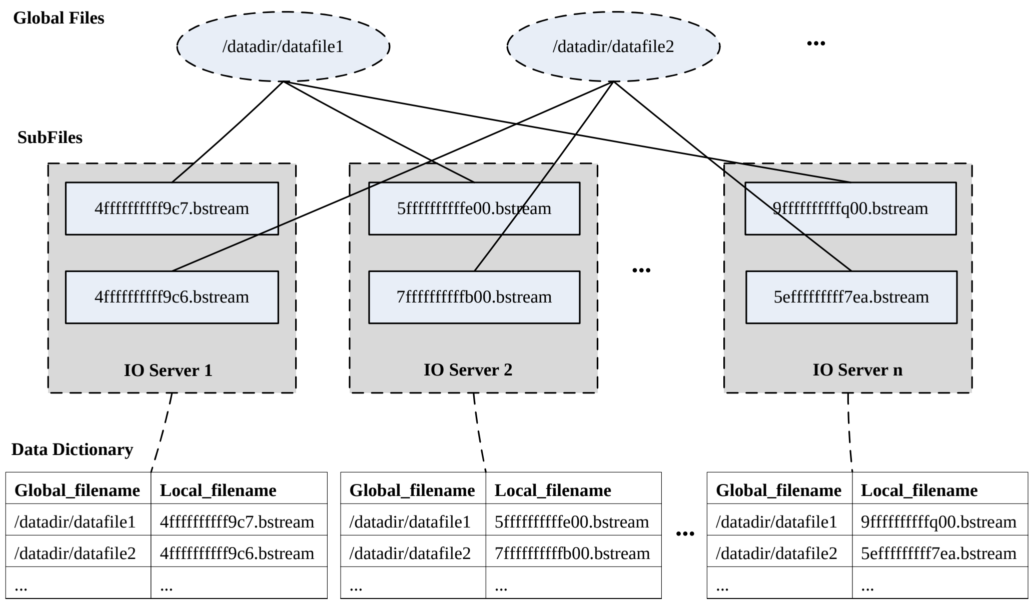
The data dictionary is modified to support two data access modes in the proposed scheme. For local data access, each node is a standalone sub-database, and it should keep all the metadata information in the local data dictionary. This design can eliminate the data dependency among different nodes. Furthermore, some additional information should be included on each node for global access. For example, file information in data dictionary on each node must contain a local file name and a global file name, as shown in Figure 5, to support for both two data access modes. This data organization is quite different from existing GFS and HDFS. In GFS and HDFS, storage nodes do not contain the global data set information, they have to turn to the dedicated metadata server for remote data access.

IV. Query Processing

A. Query Processing

In large-scale and data-intensive systems, query execution can be roughly classified as two categories: simple merging and recursive processing. Simple merging means a query is executed in parallel on multiple database nodes independently without data exchanging or inter-nodal message. The final result is a combination of intermediate results from the execution nodes, such as web search engine applications. Recursive processing means that query execution consists of multiple computing and re-distribution phases. Usually, queries that involve multi-join or nested-query are recursive processing types in a clustered database. The execution control mechanisms is much more complicated than simple merging queries.

The proposed hybrid scheme can significantly simplify data migration between different nodes during query processing, by hiding data migration details to the underlying parallel file system. For example, data collecting from multiple databases can be converted to data reading in global modes. In addition, some recursive processing queries in shared-nothing systems can be converted to simple merging queries in the hybrid



Global_filename	Local_filename
/datadir/datafile1	4fffffffff9c7.bstream
/datadir/datafile2	4fffffffff9c6.bstream
...	...

Global_filename	Local_filename
/datadir/datafile1	5ffffffffffe00.bstream
/datadir/datafile2	7ffffffffffb00.bstream
...	...

Global_filename	Local_filename
/datadir/datafile1	9ffffffffffq00.bstream
/datadir/datafile2	5efffffffff7ea.bstream
...	...

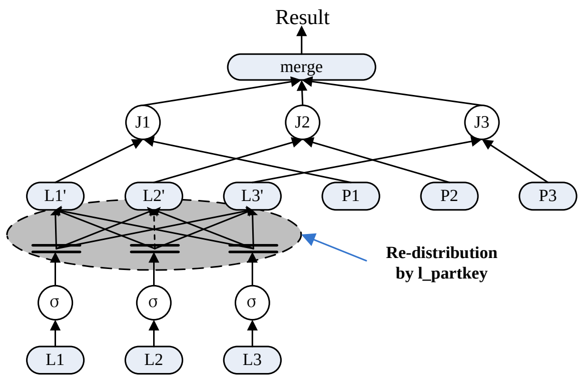
Fig. 5. File Information in Database Dictionary

system. Taking table *lineitem* and table *partsupp* in TPC-H benchmark [25] for example, the query is as follows.

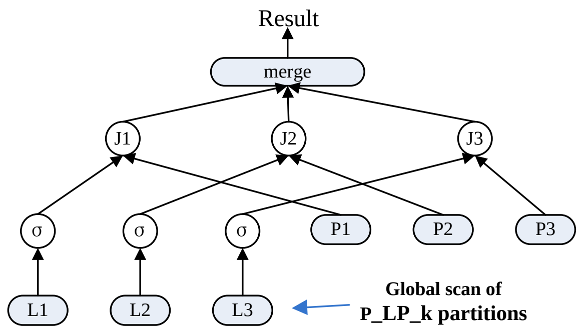
```
select   l_orderkey, l_partkey, l_suppkey,
         l_extendedprice * ( 1 - l_discount ) -
         ps_supplycost * l_quantity   profit
from     lineitem, partsupp
where    l_partkey = ps_partkey
         and l_suppkey = ps_suppkey
         and l_commitdate between(t1,t2);
```

This query lists the *profit* of each item in a certain period. We compare the query execution plan with shared-nothing systems to illustrate how this query is processed in a hybrid system. In a shared-nothing system, table *lineitem* is primarily partitioned by *l_orderkey* across database nodes and secondarily partitioned by *l_partkey* on each node, and table *partsupp* is primarily partitioned by *ps_partkey* and secondarily partitioned by *ps_suppkey*. There must be a re-distribution phase of *lineitem* before join, as shown in Figure 6(a). While in a hybrid system, the table *lineitem* is also partitioned by *l_orderkey* and *l_partkey*, but each database node can directly access data from other database nodes in global mode. Thus it eliminates the re-distribution phase in execution plan, as shown in Figure 6(b).

The proposed hybrid system simplifies the data migration during query processing, and provides more performance optimization choices. For the same reason, the hybrid system can also simplify the control mechanism for transaction processing.



(a) Query Execution Tree in a Shared-nothing System



(b) Query Execution Tree in a Hybrid System

Fig. 6. Query Execution Tree in Different Systems

V. Preliminary Experiments

A. Environment

We conducted a set of experiments to verify data file striping by record pages, the performance of local and global data access modes, and the performance of query processing. The experiments were conducted on a 64-node Sun Fire Linux-based cluster. Each node was a Sun Fire X2200 server, with dual 2.3GHz Opteron quad-core processors, 8GB memory, and a 250GB 7200RPM SATA hard drive. All 64 nodes were connected with Gigabit Ethernet. In addition, there were 16 nodes connected with 4X InfiniBand network. All these nodes

TABLE I
DATA ORGANIZATIONS OF TPC-H TABLES

Table Name	Data Size	Stripe Size	Reserved(%)
Customer	2.43 GB	64 KB	5
Lineitem	72.42 GB	8 MB	0
Nation	2224 Bytes	–	–
Orders	16.36 GB	8 MB	0
Part	2.42 GB	64 KB	5
Partsupp	11.32 GB	8 MB	0
Region	389 Bytes	–	–
Supplier	142 MB	64 KB	10

Note: reserved percentage 0 means no reserved area in record pages, but we also considered boundary alignment, adding some null characters in the end of a page if necessary in order to guarantee no record aligned across pages.

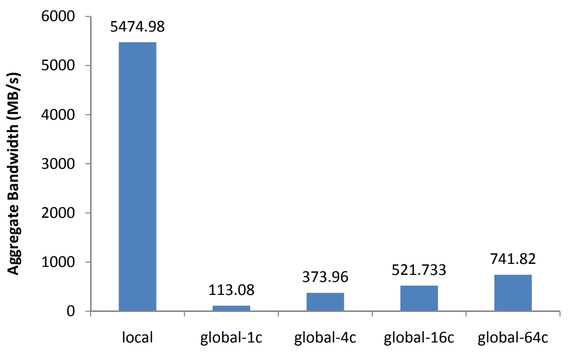
ran Ubuntu 9.04 (Linux kernel 2.6.28.10) operating system.

We employed PVFS2 as the underlying parallel file system. Global data access refers to data reading or writing from PVFS2 mount point. While local file refers to a sub-file on each storage node (file name like ***.stream). We verified the performance potential of the proposed hybrid scheme with TPC-H benchmarks.

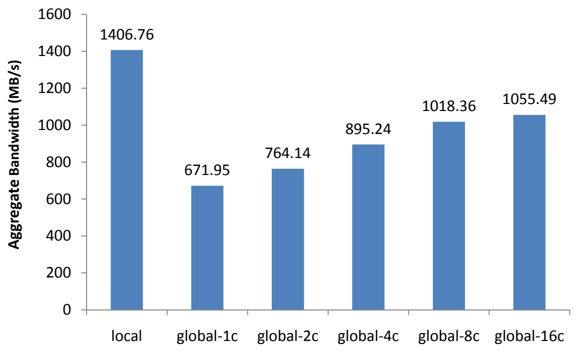
B. Results and Analysis

Firstly, we designed experiments to verify striping tables in parallel file systems by record pages. In this set of experiments, TPC-H population scale is 100, meaning that total data size is 100GB. We used dbgen(a data generating tool in TPC-H) to generate the original testing data, and then re-organized the data into record pages. After that we wrote these record pages into PVFS2 files. We adopted different stripe sizes and reserved percentage for different tables. Table I shows the data stripe sizes and reserved percentages. For each table, the record page size was equal to the stripe size. Table *nation* and *region* were not striped because the data size was too small.

Next we conducted experiments to testing the I/O performance in global and local data access modes. Figure 7 shows the aggregate bandwidth of full table scan in hybrid systems. We chose the biggest table *lineitem*, and the data size is shown in Table I. We measured the data reading performance via both Ethernet and InfiniBand interconnections. In Ethernet testing, we used all 64 nodes. The number of clients were 1, 4, 16, and 64, respectively. Each client read only one portion of the whole table in global access mode. In InfiniBand testing, we used 16 server nodes. The number of clients were 1, 2, 4, 8, and 16, respectively. Label 'local' refers to local data access mode, and 'global-1c' refers to 1 client in global data access mode, and other labels are similarly defined. From the results we can observe that, in both Gigabit Ethernet and InfiniBand testing, local data access mode obtained the highest aggregate bandwidth compared to all global data accesses, because each database node read the native files independently. However, the performance difference between local and global data access was much larger in Gigabit Ethernet interconnection than that in InfiniBand environment. This set of experiments verified that, with high-speed interconnection, the hybrid architecture



(a) Gigabit Ethernet Test on 64 Nodes



(b) InfiniBand Test on 16 Nodes

Fig. 7. Table Scan Performance in Different Access Modes

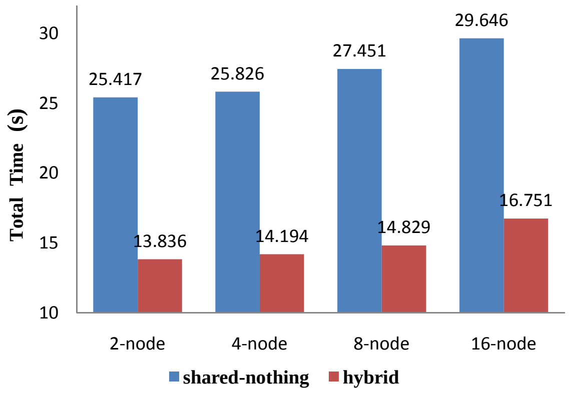


Fig. 8. Execution Time before 'JOIN' in Fig.6

can also obtain high I/O bandwidth when table scanning in global data access mode.

We then conduct experiments to test query execution time. Figure 8 shows the execution time before '*JOIN*' (SQL is listed in Section IV and execution plan is shown in Figure 6). The results were tested in the InfiniBand environment. We scaled the total data size according to the nodes number, so each database node had the same data size on average (table *lineitem* had 12 million rows and table *partsupp* had 1.6 million rows for each node on average). In both shared-nothing and hybrid systems, table *lineitem* was partitioned on two columns *l_orderkey* and *l_partkey*. As shown in Figure 6, in the shared-nothing architecture, the execution time includes local data scan, data filtering and re-distribution. In the hybrid architecture, the execution time includes global data scan and filtering. The results demonstrate that the proposed hybrid system can achieve much higher performance in all system scales, about 82% higher on average. Since the query executor can access tables in both global and local modes, it is much more appropriate for applications with varieties of query types.

VI. Conclusions and future work

In a large-scale and data-intensive system, there are various of query patterns. Some queries are proper to be processed in a shared-nothing architecture, while others are more proper to be processed in a shared-disk architecture. In this study, we propose a novel hybrid shared-nothing/shared-data scheme for large-scale and data-intensive applications, to leverage the benefits of both shared-nothing and shared-disk architectures. We adopt a shared-nothing architecture as the hardware layer and leverage a parallel file system as the storage layer to combine the scattered disks on all database nodes. With the idea of the new hybrid architecture, we first introduce two data access modes: global and local data access modes, which enable the new system can work as both a shared-nothing system and a shared-disk system. Second, we present the methodology of organizing data files in the underlying parallel file system so that each node can run as a stand-dalone sub-database. Third, we present query processing in the new storage architecture. Analytical and experimental results demonstrate that the proposed hybrid scheme can achieve both high I/O performance as a shared-nothing system, and high-speed data sharing across all server nodes as a share-disk system.

While we have proposed and verified our design and thought, this study has revealed more research issues than it has solved. In the future, we plan to study cost estimation of global and local access modes to optimize query execution. In addition, we plan to further investigate data organization and layout optimization in underlying parallel file systems.

Acknowledgment

The authors are thankful to Dr. Rajeev Thakur and Samuel Lang of Argonne National Laboratory for their help and suggestions with PVFS2. The authors are also grateful to anonymous reviewers for their coming comments and suggestions. This research was supported in part by National Science Foundation under NSF grant CCF-0621435 and CCF-0937877.

References

[1] D. DeWitt and J. Gray, "Parallel Database Systems: the Future of High Performance Database Systems," *Commun. ACM*, vol. 35, no. 6, pp. 85–98, 1992.

[2] M. G. Norman, T. Zurek, and P. Thanisch, "Much Ado About Shared-nothing," *SIGMOD Rec.*, vol. 25, no. 3, pp. 16–21, 1996.

[3] E. Rahm, "Parallel Query Processing in Shared Disk Database Systems," *SIGMOD Rec.*, vol. 22, no. 4, pp. 32–37, 1993.

[4] M. Mehta and D. J. DeWitt, "Data Placement in Shared-nothing Parallel Database Systems," *The VLDB Journal*, vol. 6, no. 1, pp. 53–72, 1997.

[5] D. D. Chamberlin and F. B. Schmuck, "Dynamic Data Distribution (D3) in a Shared-nothing Multiprocessor Data Store," in *VLDB '92: Proceedings of the 18th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992, pp. 163–174.

[6] R. Marek and E. Rahm, "Performance Evaluation of Parallel Transaction Processing in Shared Nothing Database Systems," in *PARLE '92: Proceedings of the 4th International PARLE Conference on Parallel Architectures and Languages Europe*. London, UK: Springer-Verlag, 1992, pp. 295–310.

[7] B. C. Jenq, B. C. Twichell, and T. W. Keller, "Locking Performance in a Shared Nothing Parallel Database Machine," *IEEE Trans. on Knowl. and Data Eng.*, vol. 1, no. 4, pp. 530–543, 1989.

[8] C. Lee and Z.-A. Chang, "Workload Balance and Page Access Scheduling for Parallel JOINs in Shared-nothing Systems," in *Proceedings of the Ninth International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 1993, pp. 411–418.

[9] "High-performance Storage Architecture and Scalable Cluster File System," Lustre File System White Paper, December 2007.

[10] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur, "PVFS: A Parallel File System for Linux Clusters," in *In Proceedings of the 4th Annual Linux Showcase and Conference*. USENIX Association, 2000, pp. 317–327.

[11] F. Schmuck and R. Haskin, "GPFS: A Shared-disk File System for Large Computing Clusters," in *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2002, p. 19.

[12] L. Brunie and H. Kosch, "Control Strategies for Complex Relational Query Processing in Shared Nothing Systems," *SIGMOD Rec.*, vol. 25, no. 3, pp. 34–39, 1996.

[13] A. Shatdal and J. F. Naughton, "Adaptive Parallel Aggregation Algorithms," in *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 1995, pp. 104–114.

[14] D. C. Zilio, J. Rao, S. Lightstone, G. Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden, "DB2 Design Advisor: Integrated Automatic Physical Database Design," in *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*. VLDB Endowment, 2004, pp. 1087–1097.

[15] M. Ronstrom and L. Thalmann, "Mysql Cluster Architecture Overview, High Availability Features of Mysql Cluster," MySQL Technical White Paper, April 2004.

[16] Teradata Website. [Online]. Available: http://www.teradata.com/t/white-papers/

[17] E. Rahm and T. Stöhr, "Analysis of Parallel Scan Processing in Shared Disk Database Systems," in *Euro-Par '95: Proceedings of the First International Euro-Par Conference on Parallel Processing*. London, UK: Springer-Verlag, 1995, pp. 485–500.

[18] "Oracle real application cluster 10g," An Oracle Technical White Paper, May 2005.

[19] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, 2003.

[20] D. Borthakur, "The Hadoop Distributed File System: Architecture and Design," Hadoop Project Website, 2007.

[21] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10.

[22] J. No, R. Thakur, and A. Choudhary, "Integrating Parallel File I/O and Database Support for High-performance Scientific Data Management," in *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*. Washington, DC, USA: IEEE Computer Society, 2000, p. 57.

[23] Hadoop Hive Project Website. [Online]. Available: http://hadoop.apache.org/hive/

[24] Hadoop Hbase Project Website. [Online]. Available: http://hadoop.apache.org/hbase/

[25] Transaction Processing Performance Council (TPC) Website. [Online]. Available: http://www.tpc.org/tpch/default.asp