International Journal on Wireless, Networking & Mobile Communication Innovations
Vol. 5, Issue 1 – 2021
ISSN: 2581-5113

An Optimized System for Data Sharing in Corporate Networks

Dr. Y. Dasaratha Rami Reddy[1], Mr. G. Sreenivasa Reddy[2]

[1]*Associate Professor, BVSR Engineering College.*
[2]*Associate Professor, CBIT Engineering College.*

Abstract

Most of the companies utilize the corporate networks for share the information among the companies and providing the communication between them based on common interest. It can help to reduce the computational cost and ensures the profits. Despite of its advantages it poses unique security risks, network ability and efficiency to such information sharing system. To address these problems we present a novel system called ComPeer, which provides the flexible information sharing services in cloud environments based peer to peer technology. By combining the database, peer to peer technology and cloud computing, this system provides cost-efficient, resilient and scalable network platform for corporate network applications and distribute data sharing services to participants based on the accepted pay-as-go-model.

Keywords: P2P Technology, Cloud Computing, Data Sharing, Query Processing.

Introduction

Companies of the same industry sector are often connected into a corporate network for collaboration purposes. Each company maintains its own site and selectively shares a portion of its business data with the others. From a technical perspective, the key for the success of a corporate network is choosing the right data sharing platform, a system which enables the shared data network-wide visible and supports efficient analytical queries over those data.

Existing System

Traditionally, data sharing is achieved by building a centralized data warehouse, which periodically extracts data from the internal production systems (e.g., ERP) of each company for subsequent querying. Unfortunately, such a warehousing solution has some deficiencies in real deployment. In current system data sharing is enabled by constructing a centralized data warehouse, which periodically extracts data from the internal production systems of each company for subsequent querying. But thissolution has many drawbacks in real deployment. To ensure the usability of conventional P2P networks, database community have proposed a series

of Peer-to-PeerDatabase Management System (PDBMS) by combining thestate-of-art database techniques in the P2P systems. First, the corporate network needs to scale up to support thousands of participants, while the installation of a large- scale centralized data warehouse system entails nontrivial costs including huge hardware/software investments and high maintenance cost. Second, companies want to fully customize the access control policy to determine which business partnerscan see which part of their shared data. Finally, to maximize the revenues, companies often dynamically adjust theirbusiness process and may change their business partners.

Proposed System

To address the aforementioned problems, this paper presents ComPeer, a cloud enabled data sharing platform designed for corporate network applications. By integrating cloud computing, database, and peer-to-peer (P2P) technologies, this system achieves its query processing efficiency and is a promising approach for corporate network applications. This system is also quite different from the systems based on the MapReduce and Hadoop frameworks. This ComPeer employs hybrid design to achieve the query processing Efficiency. This will also extends role-based access controls and employs P2P technology to retrieve the data between business patterns.

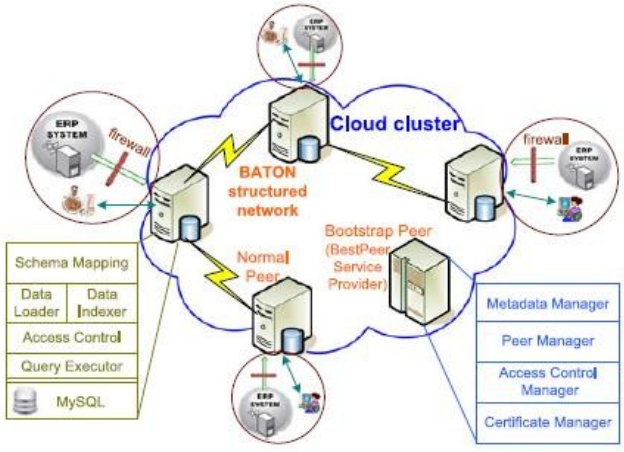

Figure 1.Overview of the System

Overview of Optimized System

In this section, we first describe the evolution of thisoptimized system platform from its early stage as anunstructured P2P query processing system, and an elastic data sharing services in the cloud. We then present the design and overall architecture of system as shown in Fg.1. While traditional P2P network has not been designed for enterprise applications, the ultimate goal of ComPeer is to bring the state-of-art database techniques into P2P systems. In particular, ComPeer provides efficient distributed search services with a balanced tree structured overlay network and partial indexing scheme for reducing the index size. A cloud enabled evolution of

ComPeer. Now in the last stage of its evolution, ComPeer is enhanced with distributed access control, multiple types of indexes, and pay-as-you- go query processing for delivering elastic data sharing services in the cloud. The software components of ComPeer are separated into two parts: core and adapter. The core contains all the data sharing functionalities and is designed to be platform independent. The adapter contains one abstract adapter which defines the elastic infrastructure service interface and a set of concrete adapter components which implement such an interface through APIs provided by specific cloud service providers.

The ComPeer core contains all platform-independent logic, including query processing and P2P overlay. It runs on top of the Cloud adapter and consists of two software components: bootstrap peer and normal peer. The bootstrap peer is run by the ComPeer service provider, and its main functionality is to manage the ComPeer network. The normal peer software consists of five components: schema mapping, data loader, data indexer, access control, and query executor.

Managing Peers Join or Depature

In addition to managing peer join and peer departure, the bootstrap peer spends most of its running-time on monitoring the healthy of normal peers and scheduling fail- over and auto-scaling events. Algorithm 1 shows how the daemon service of the bootstrap works.

Algorithm 1: Auto Fail Over and Auto Scaling BootStrapDaemon ()

Step 1: while true do
Step 2: identify the network status by calling invokeCloudWatch () function Status
S:= invokeCloudWatch()
Step 3: Declare ArrayList for peerList and newPeer
ArrayList PeerList:= BootStrap.getAllPeer ()
ArrayList newPeer:= new ArrayList ()
Step 4: for i:= 0 to peerList.size () then
Step 5: if peerList.get (i).fails () then
Peer peer:= new Peer ()
peer.loadMySQLBackUpFromRDS
(peerList.get(i))
newPeer.add(peer)
BootStrap.setBlackList(peerList.get(i))
Step 6: else
if peerList.get (i).overloaded() then
Peer peer := new Peer()
peer.upScale(peerList.get(i))
peer.clone(peerList.get(i).getDB())
BootStrap.setBlackList(peerList.get(i))
newPeer.add(peer)
Step 7: BootStrap.removeAllPeersInBlackList()

Step 8: BootStrap.addAllNewPeers (newPeer)
Step 9: BootStrap.broadcastNetworkStatus()
Step 10: sleep T seconds.

Adaptive Query Processing

ComPeer provides two services for the participants: the storage service and search service, both of which are charged in a pay-as-you-go model. This section presents the pay-as- you-go query processing module which offers an optimal performance within the user's budget. ComPeer employs two query processing approaches: basic processing and adaptive processing. The basic query processing strategy is similar to the one adopted in the distributed databases domain. Overall, the query submitted to a normal peer is evaluated in two steps:fetching and processing. In the fetching step, the query is decomposed into a set of subqueries which are then sent to the remote normal peers that host the data involved in the query. In the processing step, the normal peer first collects all the required data from the other participating normal peers. To reduce I/O, the peer P creates a set of MemTables to hold the data retrieved from other peers and bulk inserts these data intothe local MySQL when the MemTable is full. After receiving all the necessary data, the peer finally evaluates the submitted query. One problem of the basic approach is the inefficiency of query processing. The performance is bounded only one node is used. We can easily address this problem by employing more nodes to process the query in parallel.

A. A P2P Parallel Processing Approach

The idea of parallel processing for each join, instead of forwarding all tuples into a single processing node, we disseminate them into a set of nodes, which will process the join in parallel. We adopt the conventional replicated joinapproach. Namely, the small table will be replicated to all processing nodes and joined with a partition ofthe large table. When a query involves multiple joins and group by, the query plan can be expressed as a processing graph.

Processing Graph: Given a Query, the processing Graph is generated as follows

- For each node we assign a level id to each node
- Root node represents the peer that accepts the query, which is responsible for collecting the results for the user.
- Suppose query involves x joins and y "Group By" attributes, the mixmum level of the Graph satisfies

  $\leq x + f(y)$
  $(f(y) = 1, if\ y \geq 1. Otherwise\ f(y) = 0)$.
- Except for the root node, allother nodes only process one join operator or the "Group By" operator.
- Nodes of level accept inut data from the ComPeer storage system. After completing its processing, nodes sends its data to the nodes in the above level.
- All of operators that are not evaluated in the non-rootnode are processed by the root.

In the replicated join, we trade off the network cost for the parallelism. The benefit may be neutralized when a large number of tuples are re-partitioned in the P2P network. Therefore, we propose a model to estimate the cost. This parallel processing in the cost model, that the I/O and the CPU time dominate the overall cost.

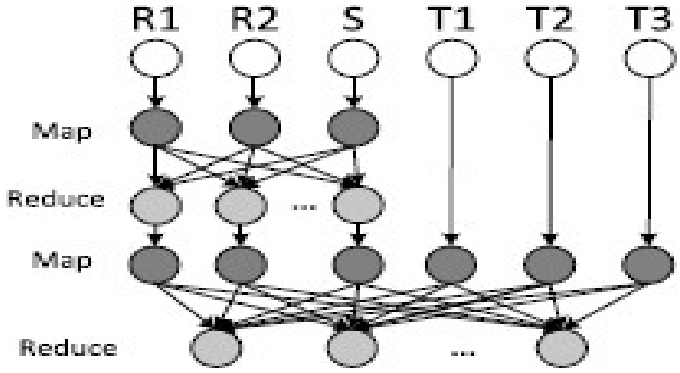

Figure 2.MapReduce Processing

B. MapReduce for COMPEER

Besides its native processing strategy, we also implement a MapReduce-style engine for ComPeer. To facilitate MapReduce processing, a Hadoop distributed file system (HDFS) is mounted at system start time to serve as the temporal storage media for MapReduce jobs. The main difference between MapReduce method and native P2P method comes from the join processing. The Fig. 2, in MapReduce method, instead of doing replicate joins, the symmetric-hash join approach is adopted. Each mapper reads in its local data and shuffles the intermediate tuple according to the hash value of the join key. Therefore, each tuple only needs to be shuffled once on each level. Note that the configuration and launch of a MapReduce job also incurs certain overhead, which, can be measured in the runtime, is a constant value.

C. Adaptive Query Processing

Based on the above-named price models, we propose our adaptive query process approach. When a query is submitted, the query planner retrieves connected histogram and index info from the bootstrap node, analyzes the question and constructs a process graph for the query. Then the cost of each the P2P engine and MapReduce engine area unit foretold supported the histograms and runtime parameters of the value models. The query planner compares the prices between 2 strategies and executes the one with lower price. The careful algorithmic rule description is shown in algorithm 2.

Adaptive Query Processing

Input: Query

Output: Query configuration on a specific query engine


```
TableSet S ← TableParser(Q);
Cost C_min ← MAX_VALUE ;
QueryPlan arget ← null ;
QueryPlanSet S ← ∅ ;
foreach Table T ∈ S do
    GraphSet GS = GraphGen(T);
foreach Graph G ∈ GS do
    QueryPlan P_1 = P2PPlanGen(G) ;
    QueryPlan P_2 = MapredPlanGen(G);
    QS = QS ∩ {P_1};

        QS = QS ∩ {P_2};
foreach QueryPlan P ∈ QS do
      if COSTEst(P) < C_min then
          C_min = CostEst(p);
          Target = P;
return Target;
```

Comparing between two cost models, we can observe that table size and query complexity are the key factors that affect the query planner's decision. With more levels of join, and larger size of tables, the query planner tends to choose the MapReduce method, while on the contrary, simple queries involving smaller data size and fewer joins are taken care ofby the P2P method.

Conclusion

We have discussed the unique challenges posed by sharing and processing data in an inter-businesses environment and proposed ComPeer, a system which delivers elastic data sharing services, by integrating cloud computing, database, and peer-to-peer technologies. Our system can efficiently handle typical workloads in a corporate network and can deliver near linear query throughput as the number of normal peers grows. Therefore, ComPeer is a promising solution for efficient data sharing within corporate networks.

References

1. Gang Chen, Tianlei Hu, Dawei Jiang, Peng Lu, Kian-Lee Tan, Hoang Tam Vo, and Sai Wu, "BestPeer++: A Peer-to- Peer Based Large-Scale Data Processing Platform" Vol. 26,No. 6, June 2014.
2. A. Abouzeid, K. Bajda-Pawlikowski, D.J. Abadi, A. Rasin, and A. Silberschatz, "HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads," Proc. VLDB Endowment, vol. 2, no. 1, pp. 922-933, 2009.
3. C. Batini, M. Lenzerini, and S. Navathe, "A Comparative Analysis of Methodologies for

Database Schema Integration,"ACM Computing Surveys, vol. 18, no. 4, pp. 323-364, 1986.

4. D. Bermbach and S. Tai, "Eventual Consistency: How Soon is Eventual? An Evaluation of Amazon s3's Consistency Behavior," in Proc. 6th Workshop Middleware Serv. Oriented Comput. (MW4SOC '11), pp. 1:1-1:6, NY, USA, 2011.
5. B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," Proc. First ACM Symp. Cloud Computing, pp. 143- 154, 2010.

