International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-1, Issue-3, June 2015
ISSN: 2395-3470
www.ijseas.com

EFFICIENT DETECTION OF INCONSISTENCIES OF DISTRIBUTED DATA IN CLOUD USING INCREMENTAL VIOLATION DETECTION

S. Anantha Arulmary[1] and N.S. Usha[2]
[1]Student, Department of Computer Science, Sir Issac Netwon College of Engineering and Technology, Pappakoil, Nagapattinam, India
[2] Assistant Professor, Department of Computer Science, Sir Issac Netwon College of Engineering and Technology, Pappakoil, Nagapattinam, India

ABSTRACT

This paper investigates the problem of incremental detection of errors in distributed data. In many distributed environments, the primary function is to detect the violation in data using various constrains, that is, instances data does not follow the constrain. Existing approaches for detecting such inconsistent data state at all times, even during normal operation, and thus incur wasteful resource overhead. In this project, we propose efficient schemes for the inconsistencies detection problem, which we model as one of detecting the violation of global constraints defined over distributed system variables. Our approach eliminates the need to continuously track the data by composing multiple constraints into single constraint using new mechanism called shard query that can be checked efficiently at each data. Only in the occasional event that a local constraint is violated, do we resort to more expensive global constraint checking. We formulate the problem of selecting constraints as an optimization problem that takes into account and objective is to minimize communication costs.

Keywords— Shared Query Technique, Horizontal and vertical partition, Elastic Compute Cloud(EC2), Conditional functional dependencies, Distributed Data.

I. INTRODUCTION

Real life data is often dirty. To clean the data, efficient algorithms for detecting errors have to be in place. Errors in the data are typically detected as violations of constraints (data quality rules), such as functional dependencies (FDs), denial constraints, and conditional functional dependencies (CFDs). When the data is in a centralized database, it is known that two SQL queries suffice to detect its violations of a set of CFDs. It is increasingly common to find data partitioned vertically or horizontally, and distributed across different sites. This is highlighted by the recent interests in SaaS and Cloud computing, MapReduce and columnar DBMS. In the distributed settings, however, it is much harder to detect errors in the data.

In Existing model, No conditional functional dependencies and no Error detection in distributed data. SQL Queries are used to detect its violations of CFDs. Applicable only for centralized Database, Cannot be used for Distributed Database in cloud.

1.1 Heuristic algorithm

These algorithms, usually find a solution close to the best one and they find it fast and easily. Sometimes these algorithms can be accurate, that is they actually find the best solution, but the

algorithm is still called heuristic until this best solution is proven to be the best. The method used from a heuristic algorithm is one of the known methods, such as greediness, but in order to be easy and fast the algorithm ignores or even suppresses some of the problem's demands.

1.2 Hashing Algorithm (MD5)

The MD5 message-digest algorithm is a widely used cryptographic hash function producing a 128-bit (16-byte) hash value typically expressed in text format as a 32 digit hexa decimal number. MD5 has been utilized in a wide variety of cryptographic applications, and is also commonly used to verify data integrity

MD5 was designed by Ron Rivest in 1991 to replace an earlier hash function,MD4.

In 1996 a flaw was found in the design of MD5. While it was not deemed a fatal weakness at the time, cryptographers began recommending the use of other algorithms, such as SHA-1 which has since been found to be vulnerable as well. In 2004 it was shown that MD5 is not collision resistant. As such, MD5 is not suitable for applications like SSL certificates or digital signatures that rely on this property for digital security. Also in 2004 more serious flaws were discovered in MD5, making further use of the algorithm for security purposes questionable; specifically, a group of researchers described how to create a pair of files that share the same MD5 checksum. Further advances were made in breaking MD5 in 2005, 2006, and 2007. In December 2008, a group of researchers used this technique to fake SSL certificate validity, and CMU Software Engineering Institute now says that MD5 "should be considered cryptographically broken and unsuitable for further use", and most U.S. government applications now require the SHA-2 family of hash functions.

1.3 Map Reducing Algorithm

A. MapReduce is a software framework that allows developers to write programs that process massive amounts of unstructured data in parallel across a distributed cluster of processors or stand-alone computers. MapReduce is a software framework that allows developers to write programs that process massive amounts of unstructured data in parallel across a distributed cluster of processors or stand-alone computers. It was developed at Google for indexing Web pages and replaced their original indexing algorithms and heuristics in 2004.

The framework is divided into two parts:

- Map, a function that parcels out work to different nodes in the distributed cluster.
- Reduce, another function that collates the work and resolves the results into a single value.
- The MapReduce framework is fault-tolerant because each node in the cluster is expected to report back periodically with completed work and status updates. If a node remains silent for longer than the expected interval, a master node makes note and re-assigns the work to other nodes.

1.4 Incremental Algorithm

An incremental algorithm updates the solution to a problem after an incremental change is made on its input. In the application of an incremental algorithm, the initial run is conducted by an algorithm that performs the desired computation from scratch and the incremental algorithm is used in the subsequent runs (i) using information from earlier computations and (ii) to reflect the update on the network while avoiding re-computations as much as possible. The computation of between's centrality depends on the number of shortest paths in a network and the intermediate nodes on these paths. A network update such as an edge insertion or edge cost decrease might result in

creation of new shortest paths in the network. However, a considerable portion of the older paths might remain intact, especially in the unaffected parts of the network. Therefore, accurate Maintenance of the number of shortest paths and the

Predecessors on the shortest paths will suffice for accurately updating between's values in the case of dynamic network updates. This is the key observation we make in the design of our incremental betweenness centrality algorithm.

Ad Hoc Networks (MANETs) consists of a collection of mobile nodes which are not bounded in any infrastructure. Nodes in MANET can communicate with each other and can move anywhere without restriction. This non-restricted mobility and easy deployment characteristics of MANETs make them very popular and highly suitable for emergencies, natural disaster and military operations.

Nodes in MANET have limited battery power and these batteries cannot be replaced or recharged in complex scenarios. To prolong or maximize the network lifetime these batteries should be used efficiently. The energy consumption of each node varies according to its communication state: transmitting, receiving, listening or sleeping modes. Researchers and industries both are working on the mechanism to prolong the lifetime of the node's battery. But routing algorithms plays an important role in energy efficiency because routing algorithm will decide which node has to be selected for communication.

The main purpose of energy efficient algorithm is to maximize the network lifetime. These algorithms are not just related to maximize the total energy consumption of the route but also to maximize the life time of each node in the network to increase the network lifetime. Energy efficient algorithms can be based on the two metrics: i) Minimizing total transmission energy ii) maximizing network lifetime. The first metric focuses on the total transmission energy used to send the packets from source to destination by selecting the large number of hops criteria. Second metric focuses on the residual batter energy level of entire network or individual battery energy of a node [1].

II. LITERATURE SURVEY

In Algorithms for computing provably near-optimal (in terms of the number of messages) local constraints. Experimental results with real-life network traffic data sets demonstrate that our technique can reduce message communication overhead by as much as 70% compared to existing data distribution-agnostic approaches. [1]

The incremental algorithm over vertical partitions to reduce data shipment. They are verify experimentally, using real-life data on Amazon Elastic Compute Cloud (EC2), that our algorithms substantially outperform their batch counterparts.[2]

Computing an optimal global evaluation plan is shown to be NP-hard. Finally, we present an implementation of our algorithms, along with experiments that illustrate their potential not only for the optimization of exploratory queries, but also for the multi-query optimization of large batches of standard queries. [3]

Along with experiments that illustrate their potential not only for the optimization of exploratory queries, but also for the multi-query optimization of large batches of standard queries. [4]

Addresses the problem of finding efficient complete local tests for an important class of constraints that are very common in practice: constraints expressible as conjunctive queries with negated sub goals. For constraints where the predicates for the remote relations do not occur more than once, we present complete local tests under insertions and deletions to the local relations. These tests can be expressed as safe, no recursive Data log queries against the local relations. These results also apply to other constraints with negation that are not conjunctive. [5]

The MapReduce is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real-world tasks. Users specify the computation in terms of a map and a reduce function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks. Programmers find the system easy to use: more than ten thousand distinct MapReduce programs have been implemented internally at Google over the past four years, and an average of one hundred thousand MapReduce jobs are executed on Google's clusters every day, processing a total of more than twenty pet bytes of data per day. [6]

In the class of integrity constraints for relational databases, referred to as conditional functional dependencies (CFDs), and study their applications in data cleaning. In contrast to traditional functional dependencies (FDs) that were developed mainly for schema design, CFDs aim at capturing the consistency of data by enforcing bindings of semantically related values. For static analysis of CFDs we investigate the consistency problem which is determine whether or not, there exists a nonempty database satisfying a given set of CFDs, and the implication problem, which is to decide whether or not a set of CFDs entails another CFD. We show that while any set of transitional FDs is trivially consistent, the consistency problem is NP-complete for CFDs, but it is in PTIME when either the database schema is predefined or no attributes involved in the CFDs have a finite domain. For the implication analysis of CFDs, we provide an inference system analogous to Armstrong's axioms for FDs, and show that the implication problem is coNP-complete for CFDs in contrast to the linear-time complexity for their traditional counterpart. We also present an algorithm for computing a minimal cover of a set of CFDs. Since CFDs allow data bindings, in some cases CFDs may be physically large, complicating the detection of constraint violations. We develop techniques for detecting CFD violations in SQL as well as novel techniques for checking multiple constraints by a single query. We also provide incremental methods for checking CFDs in response to changes to the database. We experimentally verify the effectiveness of our CFD-based methods for inconsistency detection. This work not only yields a constraint theory for CFDs but is also a step toward a practical constraint-based method for improving data quality [7].

In top-down join enumeration algorithm that is optimal with respect to the join graph. We present performance results demonstrating that a combination of optimal enumeration with search strategies such as branch-and-bound yields an algorithm significantly faster than those previously described in the literature. Although our algorithm enumerates the search space top-down, it does not rely on transformations and thus retains much of the architecture of traditional dynamic programming. As such, this work provides a migration path for existing bottom-up optimizers to exploit top-down search without drastically changing to the transformational paradigm. [8]

III. PROPOSED METHOD

In our proposed system, to reduce data shipment, e.g., counters pointer and tags in base relations. While these could be incorporated into our solution, they do not yield bounded/optimal incremental detection algorithms. There has also been a host of work on query processing and multi-query optimization for distributed data. The former typically aims to generate distributed query plans, to reduce data shipment or response time and Error Deduction.

International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-1, Issue-3, June 2015
ISSN: 2395-3470
www.ijseas.com

A. Description of the Proposed Method:

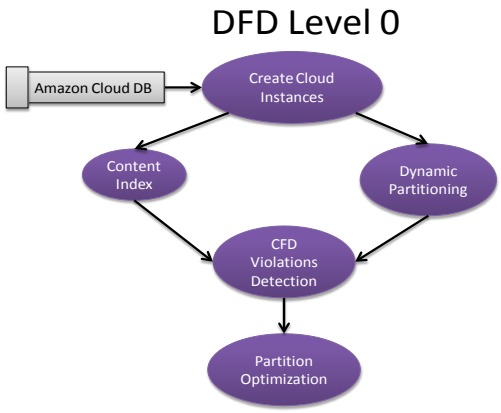


Fig 1. Data flow diagram for Proposed Model.

The steps involved in our approach are as follows:

Step 1: Data fragmentation

In relations D of schema R that are partitioned into fragments, either vertically or horizontally. In some application one wants to partition D into (D1, . . . ,Dn) horizontal partition and in some case it may be vertical. This process is mainly due to reduce the communication cost.

Step 2: CFD Violations Detection

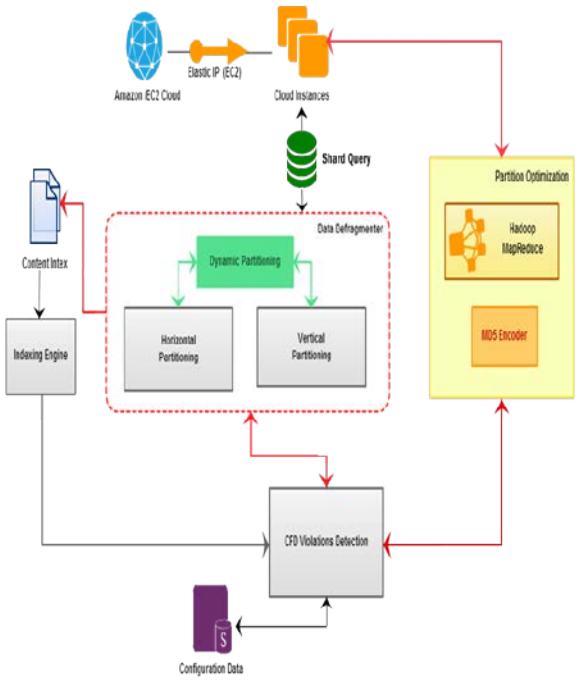
An algorithm for detecting violations of CFDs for vertical and horizontal partitions. Leveraging the index structures, an incremental algorithm is used to detect violations in vertical partitions. At first it considers a single update for a single CFD. then extend the algorithm to multiple CFDs and batch updates.

Step 3: Partition Optimization

To reduce data shipment for error detection in vertical partitions. The idea is to identify and maximally share indices among CFDs such that when multiple CFDs demand the shipment of the same tuples, only a single copy of the data is shipped. The problem for building optimal indices is NP-complete, but provides an efficient heuristic algorithm. It also provides an incremental detection algorithm for horizontal partitions. The algorithm is also optimal, as for its vertical counterpart. A tuple may be large. To reduce its shipping cost, a natural idea is to encode the whole tuple, and then send the coding of the tuple instead of the tuple.

Step 4:Shared Query Technique

Shard-Query is a high performance MySQL query engine for which offers increased parallelism compared to stand-alone MySQL. This increased parallelism is achieved by taking advantage of MySQL partitioning, MySQL sharding, common MySQL query clauses like BETWEEN and IN like etc,. The shard Query Technique is used, to enable parallel query processing to improve the query engine performance and to reduce the query processing timing.



In this above figure shows the overall architecture of Distributed data in cloud. MD5 (Message-Digest algorithm 5) is a widely used cryptographic hash function with a 128-bit hash value. We use MD5 in our implementation to

further reduce the communication cost, by sending a 128-bitMD5 code instead of an entire tuple.

IV. CONCLUSION

We studied the problem of detecting CFD violations in distributed cloud databases. The novelty of our work consists in (1) a formulation of CFD violation detection as optimization problems to minimize data shipment or response time, (2) the NP-completeness of these optimization problems when the data is partitioned either vertically or horizontally, (3) algorithms to detect CFD violations in horizontally partitioned data, aiming to minimize either data shipment or response time, (4) a characterization of locally checkable CFDs for vertically partitioned data in terms of dependency preservation, and the intractability of minimally refining a vertical partition to make it dependency preserving. As verified by our experimental results, the algorithms scale well with respect to the size of data, the number of fragments, and the complexity of CFDs, and hence provide effective methods for catching inconsistencies in distributed data. A more interesting topic is to develop techniques for detecting errors in distributed databases that are both vertically and horizontally partitioned (a.k.a. hybrid fragmentation). In the distributed setting it is also common to find replicated data . It is more interesting yet more challenging to develop detection algorithms that capitalize on data replication to increase parallelism and reduce response time. Furthermore, load balancing has proved effective for reducing the response time of distributed query processing. While our detection algorithms distribute detecting processes to distinct sites to balance the workload and explore parallel executions, this issue deserves a full treatment for violation detection in distributed databases.

REFERENCES

[1] S. Agrawal, S. Deb, K. V. M. Naidu, and R. Rastogi, “Efficient detection of distributed constraint violations,” in *Proc. ICDE*, Istanbul, Turkey, 2007.

[2] J. Bailey, G. Dong, M. Mohania, and X. S.Wang, “Incremental view maintenance by base relation tagging in distributed databases,” *Distrib. Parall. Databases*, vol. 6, no. 3, pp. 287–309, Jul. 1998.

[3] L. F. Mackert and G. M. Lohman, “R* optimizer validation and performance evaluation for distributed queries,” in *Proc. VLDB* Kyoto, Japan, 1986.

[4] Kementsietsidis, F. Neven, D. Craen, and S. Vansummeren, “Scalable multi-query optimization for exploratory queries over federated scientific databases,” in *Proc. VLDB*, Auckland, New Zealand, 2008.

[5] N. Huyn, “Maintaining global integrity constraints in distributed databases,” *Constraints*, vol. 2, no. ¾, , pp. 377–399, 1997.

[6] J.Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” in *Proc. OSDI*, 2004.

[7] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, “Conditional functional dependencies for capturing data inconsistencies,” *ACM Trans. Database Syst.*, vol. 33, no. 2, Article 6, Jun. 2008.

[8] D. DeHaan and F. W. Tompa, “Optimal top-down join enumeration,” in *Proc. ACM SIGMOD*, New York, NY, USA, 2007.