International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:5, No:1, 2011

Cloud Computing Databases: Latest Trends and Architectural Concepts

Tarandeep Singh, Parvinder S. Sandhu

***Abstract*—**The Economic factors are leading to the rise of infrastructures provides software and computing facilities as a service, known as cloud services or cloud computing. Cloud services can provide efficiencies for application providers, both by limiting up-front capital expenses, and by reducing the cost of ownership over time. Such services are made available in a data center, using shared commodity hardware for computation and storage. There is a varied set of cloud services available today, including application services (salesforce.com), storage services (Amazon S3), compute services (Google App Engine, Amazon EC2) and data services (Amazon SimpleDB, Microsoft SQL Server Data Services, Google's Data store). These services represent a variety of reformations of data management architectures, and more are on the horizon.

***Keywords*—**Data Management in Cloud, AWS, EC2, S3, SQS, TQG.

I. Introduction

CLOUD Storage provides whatever amount of storage you require, on an immediate basis. It is persistent. It can be accessed in a variety of ways, both in the data center where the cloud is housed, as well as via the Internet. If you obtain this from an external provider, it is purchased on a pay as you go basis. You do not manage it, you use it, and the service provider manages it."

Cloud systems should be geographically dispersed to reduce their vulnerability due to earthquakes and other catastrophes, which increase technical challenge on a great level of distributed data interpretability and mobility. Data interoperability is even more essential in the future as one component of a multi-faceted approach to many applications; many open challenges still remain such as cloud data security and the efficiency of query processing in the cloud. [1][2][6].

II. Amazon Web Services (AWS)

The functionality and properties in terms of performance and consistency of three services of the Amazon Web Services (AWS): S3, SQS, and EC2. Recently, Simple DB was added to the AWS family of services; unfortunately, too late to be studied as part of this work. AWS is currently the most prominent provider of utility computing. AWS is used in the remainder of this study as a basis for studying the development of Web-based applications on utility computing.

Dr. Parvinder S. Sandhu is working as Professor in Computer Science & Engineering department at Rayat & Bahra Institute of Engineering and Bio-Technology, Mohali, Punjab, INDIA. Email: parvinder.sandhu@gmail.com

Tarandeep Singh is doing Doctorate from Punjab Technical University, Punjab, India.

Other providers such as Adobe Share are beginning to appear on the market place. The results of this work are applicable to all utility services which provide a read/write interface in order to persist data in a distributed system.

A. Elastic Computing Cloud (EC2)

EC2 stands for Elastic Computing Cloud. EC2 is a service which allows clients to rent machines (CPU + disks) for a client-specified period of time. Technically, the client gets a virtual machine which is hosted on one of the Amazon servers. The cost is USD 0.10 per hour (i.e., USD 72 per month), regardless of how heavily the machine is used. One interesting aspect of EC2 is that all requests from EC2 to S3 and SQS are free. From a performance perspective, it is attractive to run applications on EC2 if the data is hosted on S3 because that way the computation is moved to the data (i.e., query shipping and stored procedures). EC2 is also attractive to implement a distributed infrastructure such as a global transaction counter [1].

B. Simple Storage System (S3)

S3 is Amazon's Simple Storage System. Conceptually, it is an infinite store for objects of variable size (minimum 1 Byte, maximum 5 GB). An object is simply a byte container which is identified by a URI. Clients can read and update S3 objects remotely using a SOAP or REST-based interface; e.g., get (uri) returns an object and put (uri, byte stream) writes a new version of the object. A special get-if-modified-since (uri, timestamp) method allows to retrieve the new version of an object only if the object has changed since the specified timestamp. This feature is useful in order to implement caching based on a TTL protocol (Section 3.3). Furthermore, user defined metadata (maximum 4 KB) can be associated to an object and can be read and updated independently of the rest of the object. This feature is useful, for instance, to record a timestamp of the last change (Section 4.5). In S3, each object is associated to a bucket. That is, when a user creates a new object, the user specifies into which bucket the new object should be placed. S3 provides several ways to scan through objects of a bucket. For instance, a user can retrieve all objects of a bucket or only those objects whose URIs match a specified prefix. Furthermore, the bucket can be the unit of security: Users can grant read and write authorization to other users for entire buckets. Alternatively, access privileges can be given on individual objects.

C. Simple Queueing System (SQS)

SQS stands for Simple Queueing System. SQS allows users

to manage a (virtually) infinite number of queues with (virtually) infinite capacity. Each queue is referenced by a URI and supports sending and receiving messages via a HTTP or REST-based interface. As of November 2007, the maximum size of a message is 256 KB using the REST based interface and 8 KB for the HTTP interface. Any byte stream can be put into a message; there is no pre-defined schema. Each message is identified by a unique id. Based on that id, a message can be read, locked, and deleted from a queue. [1].

III. Data Management in the Cloud

Cloudy Despite the potential cost advantages, cloud-based implementations of the functionality found in traditional databases face significant new challenges, and it appears that traditional database architectures are poorly equipped to operate in a cloud environment. For example, a modern database system generally assumes that it has control over all hardware resources (so as to optimize queries) and all requests to data (so as to guarantee consistency). Unfortunately, this assumption limits scalability and flexibility, and does not correspond to the cloud model where hardware resources are allocated dynamically to applications based on current requirements. Furthermore, cloud computing mandates a loose coupling between functionality (such as data management) and machines. Cloudy is a vehicle for exploring design issues such as relaxed consistency models and the cost efficiency of running transactions in the cloud. One key idea is to employ a reservation pattern in which updates are reserved before they are actually committed – in some sense, a generalization of 2-phase commit in which the ability to commit is reserved before the actual commit itself.

This section revisits distributed database architectures as they are used in cloud-computing today. First, the classic multi-tier database application architecture is described as a starting point. Then, four variations of this architecture are described. These variations are based on simple principles of distributed databases such as replication, partitioning, and caching. The interesting aspect is how these concepts have been packaged and adopted by commercial cloud services.

A. Classical

Client → Web server + application server → DB server → SAN (storage) : - In the classical model the client is connected with web server and application server which implement business logic queries on DB server and db server stores the data on SAN.

B. Partitioning

Client → Web server + application server → DB server + storage: - In partitioning the db server and the storage are combined and the data is spread across different portions of the storage.

C. Replication

The ROWA (Read Once, Write All) is implemented to replicate all data or some partition of data if combined with partitioning.

D. Distributed control

Client → Web server + application server + DB server → Storage : - storage system is separated from the database servers and the database servers access concurrently and autonomously the shared data from the storage system.

E. Caching

Client → Web server + application server → DB server → Storage → Mem Cache

The results of database queries are stored by dedicated cache servers. Typically, these servers keep the query results in their main memory so that accessing the cache is as fast as possible. Correspondingly, the set of caching servers is typically referred to as MemCache [2].

The tools take only the database schema as input and generate the queries without looking at the underlying data. Therefore, they cannot guarantee generation of queries with certain kinds of properties. In particular, we are interested in generating queries that satisfy cardinality constraints on intermediate sub expressions.

The QAGen system introduces a complementary approach towards the targeted testing problem. Instead of generating a test query given the test database, the approach generates a test database given the test query. To do so, QAGen introduces symbolic query processing which necessitates the use of constraint solvers to generate the underlying database. The primary drawback of the approach is that it generates a different database instance for each test case. As a result, the storage overheads of applying this approach for large scale testing of a new feature may be unacceptable. In addition, QAGen suffers from the overheads of using an expensive constraint solver which make it unacceptably slow for large databases [5].

IV. Targeted Query Generation (TQG) problem.

Range predicates can be modified only by changing the constant in the expression. Thus, for example age < 50 can be modified to age < 70 or age < 30, but not to age > 20. Queries can be modified by altering the range predicates. This process of modifying queries is defined as query refinement.

1) Single Cardinality Constraint
2) Multiple Constraints
3) Space Bounding
4) Scoring Cells
5) Pruning
6) Sampling Scheme : Concept of histograms (cost base, rule base)
7) Evaluation Layer for Bounding.

V. Schema-Mapping Techniques

A. Basic Layout

The most basic technique for implementing multi-tenancy is

to add a tenant ID column (Tenant) to each table and share tables among tenants. This approach provides very good consolidation but no extensibility.

B. Private Table Layout

The most basic way to support extensibility is to give each tenant their own private tables. In this approach, the query-transformation layer needs only to rename tables and is very simple. Since the meta-data is entirely managed by the database, there is no overhead for meta-data in the data itself. However only moderate consolidation is provided since many tables are required. This approach is used by some larger services when a small number of tenants can produce sufficient load to fully utilize the host machine [3].

C. Extension Table Layout

The above two layouts can be combined by splitting off the extensions into separate tables. Because multiple tenants may use the same extensions, the extension tables as well as the base tables should be given a Tenant column. A Row column must also be added so the logical source tables can be reconstructed. At run-time, reconstructing the logical source tables carries the overhead of additional joins as well as additional I/O if the row fragments are not clustered together. On the other hand, if a query does not reference one of the tables, then there is no need to read it in, which can improve performance. This approach provides better consolidation than the Private Table Layout, however the number of tables will still grow in proportion to the number of tenants since more tenants will have a wider variety of basic requirements [3].

D. Universal Table Layout

Generic structures allow the creation of an arbitrary number of tables with arbitrary shapes. A Universal Table is a generic structure with a Tenant column, a Table column, and a large number of generic data columns. The data columns have a flexible type, such as VARCHAR, into which other types can be converted. The nth column of each logical source table for each tenant is mapped into the nth data column of the Universal Table. As a result, different tenants can extend the same table in different ways. By keeping all of the values for a row together, this approach obviates the need to reconstruct the logical source tables. However it has the obvious disadvantage that the rows need to be very wide, even for narrow source tables, and the database has to handle many null values. While commercial relational databases handle nulls fairly efficiently, they nevertheless use some additional memory. Perhaps more significantly, fine-grained support for indexing is not possible: either all tenants get an index on a column or none of them do. As a result of these issues, additional structures must be added to this approach to make it feasible [3].

E. Pivot Table Layout

A Pivot Table is an alternative generic structure in which each field of each row in a logical source table is given its own row. In addition to Tenant, Table, and Row columns as described above, a Pivot Table has a Col column that specifies which source field a row represents and a single data-bearing column for the value of that field. The data column can be given a flexible type, such as VARCHAR, into which other types are converted, in which case the Pivot Table becomes a Universal Table for the Decomposed Storage Model. A better approach however, in that it does not circumvent typing, is to have multiple Pivot Tables with different types for the data column. To efficiently support indexing, two Pivot Tables can be created for each type: one with indexes and one without. Each value is placed in exactly one of these tables depending on whether it needs to be indexed. This approach eliminates the need to handle many null values. However it has more columns of meta-data than actual data and reconstructing an n-column logical source table requires $(n - 1)$ aligning joins along the Row column. This leads to a much higher runtime overhead for interpret-ing the meta-data than the relatively small number of joins needed in the Extension Table Layout. Of course, like the Decomposed Storage Model, the performance can benefit from selectively reading in a small number of columns [3].

F. Chunk Table Layout

The third generic structure, called a Chunk Table, that is particularly effective when the base data can be partitioned into well known dense subsets. A Chunk Table is like a Pivot Table except that it has a set of data columns of various types, with and without indexes, and the Col column is replaced by a Chunk column. A logical source table is partitioned into groups of columns, each of which is assigned a chunk ID and mapped into an appropriate Chunk Table. In comparison to Pivot Tables, this approach reduces the ratio of stored metadata to actual data as well as the overhead for reconstructing the logical source tables. In comparison to Universal Tables, this approach provides a well-defined way of adding indexes, breaking up overly-wide columns, and supporting typing. By varying the width of the Chunk Tables, it is possible to find a middle ground between these extremes. On the other hand, this flexibility comes at the price of a more complex query-transformation layer [3].

G. Chunk Folding

As the technique called Chunk Folding where the logical source tables are vertically partitioned into chunks that are folded together into different physical multi-tenant tables and joined as needed. The database's "meta-data budget" is divided between application-specific conventional tables and a large fixed set of Chunk Tables [3].

VI. Keyword Query Specification

NUITS supports several advanced keyword queries as well as simple keyword queries. Keyword Specification followed by the discussions on advanced keyword queries. [4]

A. Simple keyword: A simple keyword is just a keyword, for example database.

B. Typed keyword: Users do not need to know the underneath relational database schema when they issue keyword queries. But, because a keyword may appear in any attributes and in any relations, the results may be large and include many users do not need. In order to restrict the search space, typed keywords are introduced in NUITS which allows users to specify a keyword with a type. Here a type can be either relation-name or attribute-name. For example, Paper: database means that a keyword of database appearing in the Paper relation. In addition, we introduce a wildcard * for any possible keyword. For instance, if a user is interested in any authors who wrote a paper on database, he/she can issue a 2 keyword query with Author:* and database. Since casual users may not know the exact relation or attribute name, NUITS supports aliases. The same query in the above example can also be written as Writer:* and database, as long as the alias "Writer" has been configured in advance by system administrators.

C. Conditional keyword: NUITS allows users to specify conditions associated with a keyword. For exam database year>2000 specifies a condition associated with the keyword database. The condition means that, if a tuple containing the keyword database has an attribute called year, its value must be greater than 2000. Instead of >, the other comparators such as <, ≤, =, ≥ and _= can also be used. Note: a keyword can be associated with multiple conditions. In addition, NUITS provides a special operator ∼ for approximation keyword. For example, database year ∼ 2000 means that the tuple-connection-trees with nodes (tuples) containing a numerical value of year, which is closer to year 2000, will be given a smaller cost [4] .

VI. Tree Clustering

The search engine will report the top-k minimal cost tuple connection trees. However, a potential problem is how to select such a parameter k. When k is small, a user may not be able to find the expected tuple-connection-trees. When k is large, a user may find it difficult because there are too many such trees. In order to assist users to find the needed tuple-connection-trees, in NUITS, we propose to cluster the similar trees into clusters. Two trees, ti and tj , are in the same cluster, if ti and tj are isomorphic to each other. Here, we consider trees as labeled trees at the schema level. Then, ti and tj are isomorphic to each other, if there is a one-to one mapping from nodes of ti to nodes of tj. [4]

a) Structural-Level Clustering: The structural-level clustering is to cluster trees using the tree isomorphic
b) Content-Level Clustering: The content-level clustering further clusters tuple-connection-trees if the size of the cluster is larger than a user given threshold, after structural-level clustering. The content-level clustering is based on keyword frequencies and content similarity.

VII. Conclusion

Cloud based applications need high scalability and availability at low and controlled cost. In the 1960s, most of the database applications were used to maintain cash flow i.e. simple debit and credit transactions. For any organization it was easy to spend a large portion of the IT budged on database software and administration. In the meantime, applications have been changed and there is tremendous growth in data and databases only solve a relatively small fraction of problem. As of today, utility computing is not limited only to the single database system for support and high performance, it requires many interactive applications. The purpose of this paper is to show some latest database architecture concepts. The paper tried to do that for Web-based applications such as, e.g. an online Mobile recharge system, Using Mobile as Wallet (Mollet) for shopping, the result was a new problem statement, and not surprisingly a new architecture and a different packaging of database functionality. In this scenario there must be link between banking application, Mobile application and web application so that the transaction can took place from Web to mobile , mobile to mobile and mobile to bank for the cash flow. It will make easy the track sale, purchase and tax calculation. This is possible when all the database applications are linked and this can be possible via Cloud.

References

[1] Matthias Brantner, "Building a Database on S3," Proceeding of Systems Group, ETH Zurich, Pages 251-263, June 9–12, 2008.
[2] Donald Kossmann Tim Kraska Simon Loesing, "An Evaluation of Alternative Architectures for Transaction," Proceeding of Systems Group, Department of Computer Science, ETH Zurich, Switzerland Pages 579 - 590, June 6–11, 2010
[3] Chaitanya Mishra,Nick Koudas, Calisto Zuzarte, "Generating Targeted Queries for Database Testing," SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada Pages 499 - 510,.
[4] Shan Wang, Zhaohui Peng, Jun Zhang Lu Qin, Sheng Wang , Jeffrey Xu Yu, Bolin Ding , "NUITS: A Novel User Interface for Efficient Keyword Search over Databases," Proceeding of VLDB '06, September 12-15, 2006, Seoul, Korea. Pages 1143 - 1146.
[5] Gustavo Alonso Donald Kossmann Timothy Roscoe Nesime Tatbul,Andrew Baumann Carsten Binnig Peter Fischer Oriana Riva Jens Teubner, "The ETH Zurich Systems Group and Enterprise Computing Center," In preceding of ETH Zurich, Zurich 8092, Switzerland Pages 94 - 99 SIGMOD Record, December 2008
[6] Rakesh Agrawal, Anastasia Ailamaki, Philip A. Bernstein, Eric A. Brewer, Michael J. Carey, "The Claremont Report on Database Research," Proceeding of Claremont Resort in Berkeley Pages 9 - 19 , September 2008.