CloudDB: One Size Fits All Revived

Hakan Hacigumus, Junichi Tatemura, Wang-Pin Hsiung, Hyun J. Moon, Oliver Po, Arsany Sawires, Yun Chi, Hojjat Jafarpour
NEC Labs America

1. INTRODUCTION
We are witnessing the emergence of Cloud Computing as a promising step in the evolution of information technology to address the challenges many organizations are facing in today’s fast paced and fiercely competitive economic environment [1]. It is vital to be able to quickly react to market opportunities and threats in this environment. One challenge is to enable applications to scale to the “right size” for the current workload. The compute clouds look ideal for those applications to scale out by adding server resources as much as needed. In reality, however, software architecture for scale-out clouds is still a challenging issue due to the common need for re-architecting the applications for further scalability.

The limitation of natural parallelization comes from data management. Traditional relational database systems have matured after decades of research and development and very successfully created a large market and solutions to businesses. However, ever increasing need for scalability and new data and applications create new challenges for relational database systems. Different business needs and workloads may be satisfied with diverse set of data management capabilities. However, it is simply not feasible for all organizations to test, invest in, and maintain all possible data management solutions to find the best fit. More importantly the business needs also evolve over the time by making existing implementations obsolete.

Cloud computing model offers an opportunity here. In cloud computing model the users do not have to make a binding decision on a specific data management product or technology to invest in. In contrast, the cloud service provider can maintain a portfolio of products and offer them as services through common unified APIs. The service provider can afford this kind of offering as larger number of diverse clients would make the investment profitable. In this case, essentially, the clients do not have to decide what size fits them but they will have a store – the cloud database provider - that offers all sizes as their needs and requirements evolve.

We have been developing a data management platform, called CloudDB, to address these challenges and also exploit the opportunities introduced by the cloud computing model.

What should CloudDB offer for the applications? To answer this, let us first recall a key value the RDBMS has offered to traditional applications: data independence. The success of the RDBMS is made possible by the notion of data independence, which is a type of data transparency that decouples applications’ view of the data and physical organization of the actual data: the layout of data can be modified without any modification to applications. This feature enabled sophisticated optimization of data processing as well as consistency management of concurrent data accesses. We claim that data independence is missing in the current platforms on the scale-out clouds.

2. DATA INDEPENDENCE
The guiding principle of CloudDB design is establishing data independence for the applications that need to use diverse underlying data stores that are optimized for varying workload needs and characteristics. The applications should not have to be aware of the physical organization of the data and how the data is accessed. Ideally, an application only needs a logical specification of the data access layer and the data access requests are handled in a declarative way.

There has been an ongoing discussion in research and industrial communities questioning the use of traditional RDBMSs to solve all data management needs of today’s businesses [2]. CloudDB aims at removing the burden of choosing best-fit database products and data models from the application owners. Rather, CloudDB hosts variety of specialized databases that deliver high performance, scalability, and cost efficiency for varying application needs. CloudDB’s API layer is designed in such a way to give data independence to the higher level applications. The goal is to let the clients use just a simple, standard, and uniform language API to access data management functions as a service.

We choose SQL as the query language to develop the system API. Despite the known limitations of SQL, it is by far the most widely used language by the tools and applications for database access. Another important issue to observe is the confusion between the query language, SQL, and the data model, relational data model. It is important to recognize that SQL is essentially the language level support and does not define how the transactions are defined and handled, or queries are processed. Therefore SQL still manifests itself as a valuable option on top of different data models and query processing semantics as a query language and API, as an example of [3].

3. CloudDB System Architecture
The architecture for the CloudDB system is shown in Figure 1. Due to the space limitations, we only give an overview of most relevant major components of the system.

3.1 Data Stores
CloudDB maintains three different types of data stores to efficiently handle different types of workloads. The client data may be replicated or partitioned among multiple stores.

Relational Store: Relational store consists of traditional RDBMS nodes on commodity machines that are used to handle transactional workloads. CloudDB uses replication and partitioning to respond to various workload characteristics. Relational store also provides a preferable alternative for the clients who do not have very demanding query and data scalability requirements. In that case multiple clients can be collocated in RDBMS nodes and replication support can afford reasonable scalability. As the data/workload of a client grows
3.2 Intelligent CloudDB Coordinator (ICDC)

ICDC is responsible for all functions and decisions regarding resource management, capacity planning, workload management, and system data collection. The “profit” is the main metric of decisions in ICDC. This is in line with the dual goal of the system, which is meeting the SLA requirements of the clients while optimizing the operational cost of the cloud service delivery. The former constitutes the revenue and the latter is the cost incurred by the cloud service provider and the difference is essentially the profit.

System Monitor Database (SMDB): ICDC continuously monitors the system and keeps track of crucial information, such as performance and health of the storage engines, workload/data characteristics and statistics, performance of certain system modules (e.g., schedulers, dispatcher etc.). All this information is stored in SMDB.

The Workload Analysis and Optimization: component is responsible for analyzing the characteristics of the client workloads to find the optimal backend database node(s) to execute the workload.

Design Optimizer: makes decisions to figure out 1) optimal logical partitions and 2) optimal physical placement of the partitions. Alternative partitioning schemes and different placements of those partitions can have a significant impact on the performance of the system and operational costs.

Automated data and workload migration: The workload of applications will evolve over the time. A cloud DBMS should continuously monitor the workload and performance and switch data organization scheme when needed. Data re-organization requires migration of data, which should be done in a graceful manner to minimize the possibility of service disruption.

Capacity Planner: component periodically examines the information gathered in System Monitor Database along with operational cost and client SLA metrics to revise system resources. Scaling in/out decisions are implemented by the Cluster Controller component.

Multi-tenancy Manager (MTM): Deciding how the data management resources should be shared among multiple clients and how the clients should be grouped together to share those resources at varying granularity levels are crucial decisions that need to be made in the system. An important criterion for the Multi-tenancy manager is the isolation of SLAs of co-located tenants.

4. REFERENCES


over the time, the system migrates all or part of the data to dedicated RDMBS clusters or other data stores based on the SLA requirements and profit considerations.

Key/Value Store: Key-Value Store is maintained to achieve high levels of scalability for read/write intensive workloads. CloudDB achieves data independence for key-value stores by automating the software architects’ effort on physical data organization to fit the data in distributed data store (Auto Sharding). Given a logical design, CloudDB’s Design Optimizer de-normalizes entity relations and distributes them on key-value stores. Inconsistency due to duplication is automatically managed. SQL queries from an application are automatically transformed so that the application does not have to be modified no matter how physical organization of data is changed. The involved details of data independence support for key-values stores in CloudDB is beyond the scope of this presentation.

Columnar Store: Columnar Store is a read-optimized, throughput oriented data store to efficiently handle analytical (OLAP) type of workloads. Although the Design Optimizer may make use of materialized views and data partitioning for improved performance over relational store, the overall system design favors native support for columnar store to take full advantage of specific features of column stores, such as compression and specialized joins.

Data Store Selection: CloudDB makes the data store selection decisions based on various factors such as application context, workload characteristics, SLA requirements, etc. Data load and data access requests (i.e., queries) are two main areas where the selection of a data store is important.

Workload manager: performs fundamental functions, such as query dispatching and query scheduling. These functions are performed outside of the data nodes. Once a specific job is scheduled for a specific data node then the data node is responsible for local processing and optimization.

The dispatcher: receives queries from the Distributed Query Processor and makes the decision to submit the query to one of the valid replicas. After dispatching, the query is received by the local scheduler of the server. CloudDB employs advanced scheduling techniques to assign scheduling priority to the queries jobs in the local queues in front of the servers.