# CloudDB: A Data Store for All Sizes in the Cloud

Hakan Hacıgümüş, Jun'ichi Tatemura, Yun Chi, Wang-Pin Hsiung, Hojjat Jafarpour, Hyun Jin Moon, Oliver Po

*Data Management Research*
*NEC Labs America*

## ABSTRACT

We present a vision for a comprehensive data management platform in the cloud. The envisioned system, called *CloudDB*, would provide data management capabilities as a service to transparently and efficiently support diverse application workloads with identifiable SLA guarantees and end-to-end system management functions. The system will be able to employ heterogonous underlying storage models to effectively meet applications' query and scalability requirements. We propose the achievement of data independence for all underlying specific storage models as the key guiding principle for the system. If the system is able to achieve data independence, the application logic is decoupled from the data processing logic and allows applications to enjoy the benefits of individual storage models, which are optimized for particular purposes, without worrying about the specifics of data processing. In this paper, we describe the vision, and our rationale for why such a system is needed and desirable. We also highlight some significant challenges to the realization of such a system.

## 1. INTRODUCTION

Traditional relational database systems have matured after decades of research and development and very successfully created a large market and solutions for businesses. However, ever increasing need for scalability, new workload types and applications create further challenges for relational database systems. Different business needs and workloads may be satisfied with diverse set of data management capabilities.

Not surprisingly, 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 [1]. As the value of data and converting data into information become more evident, new players are coming to the market with products and services that are specialized for specific data managements needs, such as [2][3][4][5]. Those products and services can deliver significantly higher performance by exploiting the specific characteristics of the data and the workloads. Although emergence of those products creates valuable alternatives for the users, it also makes the selection of right products for data management needs even more daunting task for the users. It is simply not feasible for all organizations to test, invest in, and maintain all possible data management products to find the best fit. The development and deployment of these new technologies is still very difficult due to various obstacles, such as lack of standardization, documentation, well-defined architectures and components, etc [6]. In addition, there are clear benefits and desire to use different data management products side-by-side. For example some companies choose to use new-breed database products geared towards scalability, while keeping their more critical transactions in relational databases [7]. Similarly cloud providers are offering relational database services along with non-traditional database offerings [8]. More importantly the business needs also evolve over the time by making existing implementations obsolete, which results in constant re-architecting and code-patching.

The cloud computing model offers an opportunity here as a promising step in the evolution of information technology to address the challenges that organizations are facing in today's fast paced and fiercely competitive economic environment [9]. The database community has also shown great interest in exploiting the cloud computing model for data management services [10][11][12][13][14]. In cloud computing model the users do not have to make binding decisions on specific data management products or technologies 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 up front what size fits them but they will have a store – the cloud database provider – that offer all sizes as their needs and requirements evolve. We believe that such a data management platform in the cloud would afford a significant value proposition for the users and appealing business model for the service providers.

We envisioned such a platform, called *CloudDB*, and have been developing it at NEC Labs. In the remainder of the paper, we further describe the vision, and our rationale for why such a system is needed and desirable. We also highlight some significant challenges to the realization of such a system.

## 2. DESIGNING CloudDB

### 2.1 Design Goals

The design considers both the client and the service provider sides. The following is the design goals from those standpoints:

**For the clients:**

- Cost efficiency
- Standard language API (SQL)
- Identifiable and verifiable Service Level Agreements (throughput, latency, availability, scalability, security, etc.)

- Common DBMS maintenance tasks, (e.g. backup, versioning, patching etc.)
- Enablement of value-add services, such as business analytics, information sharing, collaboration etc.

**For the cloud service provider:**

- Satisfying clients' SLAs to sustain revenue
- Great cost efficiency via high level of automation and resource sharing to ensure profitability
- Maintaining an extendable platform for value-add services

CloudDB aims at transparently supporting specialized storage models and engines that are optimized for specific application workloads. For example, a columnar store could be preferable for an analytical (OLAP) type workload whereas a row store could give a better performance for a transactional (OLTP) type of workload. The maintenance of those diverse storage engines should be hidden from the clients. Also, CloudDB makes necessary decisions to direct a given workload to optimal storage engine considering various factors, such as data availability, SLA requirements, costs etc. To achieve these design goals the CloudDB architecture can employ specific functional components and advanced methods that are discussed in the following sections.

## 2.2 Design Principals

Similar to the design goals, we try to identify our design principals both from the users' and the service providers' perspectives.

From the users' perspective, the key question we asked is: 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 Codd's 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.

We extend the notion of data independence to *system independence*. The idea is that the platform can use any system to store and process the users' data. However, the applications do not have to know what systems are used and how. Achieving this goal requires a middleware layer between the applications and the underlying data stores. The middleware layer would be responsible for making all the decisions regarding the choice of data stores, processing the queries, and end-to-end system optimization both for the users' and the service providers' goals.

However, while the middleware can abstract away the underlying storage systems, it should explicitly express certain essential aspects of the system, such as consistency levels and scalability of transactions. The design and implementation of such a middleware is the central activity in the CloudDB project.

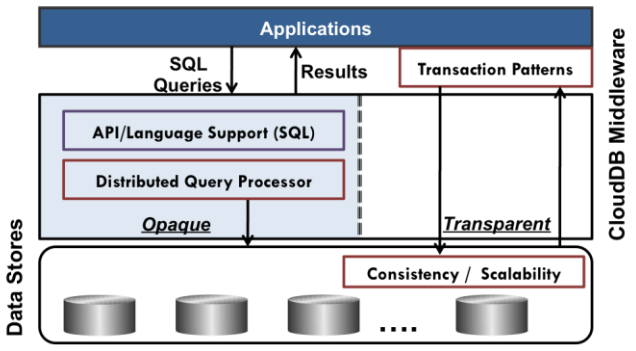

**Figure 1.** Hidden (Opaque) vs. Visible (Transparent) Aspects of the System

We show this concept in Figure 1. An application can submit the workload in SQL to the middleware and the middleware decides how the requests should be handled by hiding the details of the executions from the application, i.e. this part is opaque to the applications. However, the application can see how the transaction patterns would be handled from the consistency and scalability perspectives in the system. In essence this part is visible to the applications. One interesting questions is the right separation between the hidden and visible aspects of the system, on which we are working in the project.

## 2.3 Scalability Challenges

The scalability requirements of new applications and workloads are arguably the most important factor that accelerated the development of database technologies that are proposed as alternatives to the relational databases. There are two types of scalability CloudDB has to address as depicted in Figure 2.

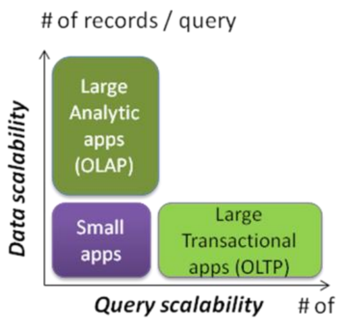

**Figure 2.** Scalability Dimensions

- **Query scalability**: the system must be scalable against increasing number of requests per second. A typical scenario is web applications / SaaS.
- **Data scalability**: the system must be scalable against increasing amount of data that needs to be processed together for one request. A typical scenario is analytics / data warehousing.

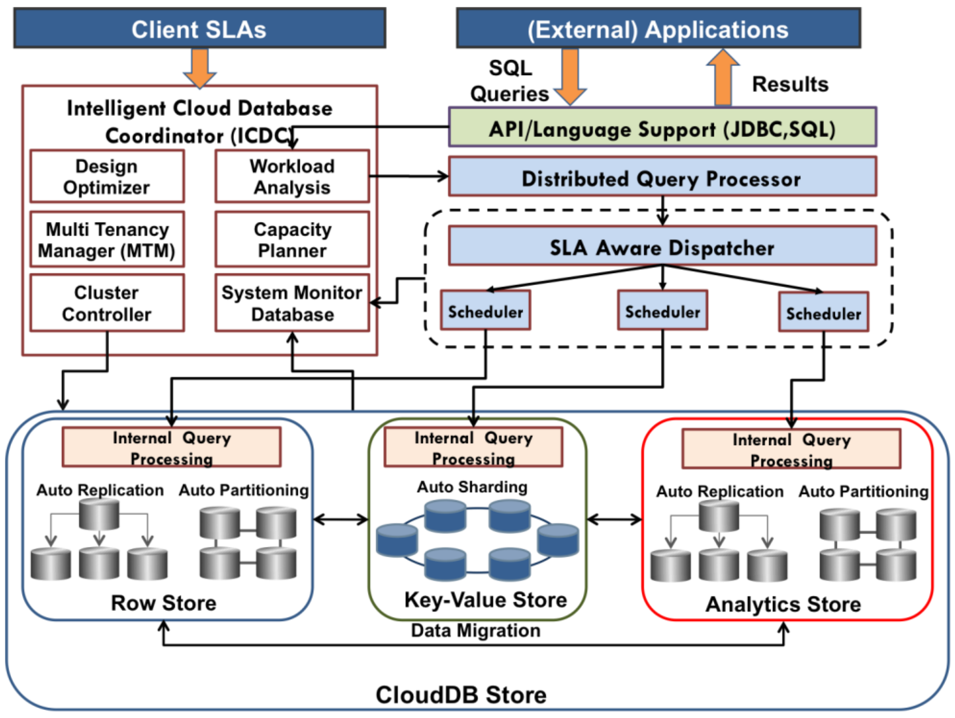

**Figure 3.**Strawman CloudDB System Architecture

It is obvious that we need a data management platform that supports both query and data scalability for diverse set of workloads. Typically the users purchase, deploy, administer, and maintain separate data management solutions and products to achieve the scalability in those two dimensions. Our goal is to remove that burden from the clients by providing all necessary data management capabilities to support data and query scalability and let the client's use just a simple, standard, and uniform language API to access data management functions as a service.

CloudDB should be able to leverage elasticity of the cloud computing resource, that is, the system should be able to handle evolving workloads by adding and removing cloud resources without re-engineering applications. In addition to row store (RDBMS), CloudDB employs specialized data stores to address scalability challenges along those two dimensions, namely; key-value store for query scalability and analytics store for data scalability dimension.

Note that strict consistency cannot be achieved for scale-out clouds [15]. In order to achieve data independence despite of this limitation, we introduce declarative specification of data quality requirements (such as consistency and accuracy), which is given from the application developer, in accompany with logical data/query description. This can be seen as a type of SLA. The traditional SLA would specify response time and availability. In addition to that, the CloudDB let the user specify requirements on data quality.

One of our goals in our vision is the extensibility of the platform from the scalability perspective. For example, if there are future applications that would require the handling of empty parts of this picture; we should be able to integrate those seamlessly.

## 3. STRAWMAN CloudDB SYSTEM ARCHITECTURE

Here we outline a strawman CloudDB system architecture as shown in Figure 3.

### 3.1 Data Stores

We plan to maintain three different types of data stores; row store, key-value store, and analytics store, to efficiently handle different workload types in CloudDB. If other types of stores tailored for specific workloads are developed, they could also be added. The client data may be replicated or partitioned among multiple stores. CloudDB aims at removing the burden of performing data migrations among various stores and management of those stores that should be done manually by the users otherwise.

One of the stores is designated as the primary and others as secondary. A typical example is the relational store is primary and data is asynchronously replicated into the analytics store. In this case the workload manager can decide to send OLTP-like requests to the relational store and OLAP-type requests to the analytics store. Ability to use specialized stores together and join data from them is a desired feature for many applications. For example, some e-commerce applications require significant amount of analytics functionality, such as customer behavior tracking, fraud detection etc. This type of workload can be handled well in an analytics store. The same applications also require transactional workloads for money transactions etc., which can be efficiently handled in a relational database system. At the same time it is desired to be able to join data from those stores to create new value-add applications such as real-time analytics, which are enriched and individualized based on the transactional customer data.

#### *3.1.1 Row Store*

The row store will be implemented as traditional RDBMS nodes that are used to handle transactional workloads. Current design considers replication-based open source relational database clusters on commodity machines. The row store 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 RDMBS nodes and replication support can afford reasonable scalability. As the data/workload of a client grows over the time, the system could migrate all or part of the data to dedicated RDMBS clusters or other data stores based on the SLA requirements and profit considerations.

#### *3.1.2 Key-Value Store*

Key-Value Store is considered to achieve higher levels of scalability for read/write intensive workloads. Although they show great scalability characteristics, one major drawback of key-value stores is that it is very difficult to program full-scale applications only with a limited interface (i.e., put/get) of the key-value stores. For instance, it does not support join processing. The best practice among skilled engineers is to de-normalize data: when an application needs to join two types of entities together, such join is pre-computed, and the combined data is put into the key-value store. This eliminates the need of join which is expensive on key-value store environment. However, note that de-normalization causes duplication of original data. The application is now responsible to handling inconsistency due to duplication. Even if the engineer is skilled enough to develop customized maintenance of inconsistency, this can be very expensive to execute.

CloudDB could achieve data independence for key-value stores by automating the software architects' effort on physical data organization to fit the data in distributed data store. Given a logical design; CloudDB's Design Optimizer can de-normalize entity relations into nested ones, which are distributed on key-value stores. Inconsistency due to duplication is automatically managed. SQL queries from an application should also be automatically transformed so that the application does not have to be modified no matter how physical organization of data is changed.

#### *3.1.3 Analytics Store*

Analytics Store would be a read-optimized, throughput oriented data store to efficiently handle analytical (OLAP) type of workloads. Currently, CloudDB design considers open-source-based storage and data processing technologies that could be leveraged both for structured and unstructured data types. Although the Design Optimizer can make use of materialized views and data partitioning for improved performance over row store, the overall system design favors native support for analytics store to take full advantage of specific features that are only available in specialized storage models, such as compression and specialized joins [16].

#### *3.1.4 Data Store Selection*

CloudDB can make the data store selection decisions based on various factors such as application context, workload characteristics, SLA requirements, etc. Data load and data access requests are two main areas where the selection of a data store is important. The initial data load can be performed against any of the stores and the data can be replicated or migrated fully or partially to other stores as necessary.

#### *3.1.5 Data Organization*

Logically the clients' data are always partitioned in the system. The number of partitions is subject to optimization. For example, if the client data is small enough, it may not be beneficial to spread it to multiple partitions. In that case client's entire data can be stored in a single partition. Partitions are replicated for scalability and availability purposes. Each replica of a partition is assigned to a physical storage engine at the backend.

As example, consider a client who has a diverse data set and certain parts of the data are used for transactional queries while the other parts are used for off-line reporting purposes. The system may make a decision to partition the client's data into two partitions corresponding to those two sets. Then the first set can be stored in a relational database and the second set can be stored in the analytics store. Both of those can utilize replication to achieve higher scalability and availability

### 3.2 SYSTEM COMPONENTS

**Intelligent CloudDB Coordinator (ICDC)** is responsible for all functions and decisions regarding resource management, capacity planning, workload management, and system data collection. The key issue with ICDC's decision making is the identification of the metric on which the system optimizes. The metric should be relevant to database systems' characteristics and the cloud computing model. We choose to use *SLA-based profit* as the metric, which is the ultimate goal of service providers, rather than low-level system metrics, such as average query response time. SLA-based profit is identified by two parts: i) the revenues and ii) operational cost. The revenue is what service clients pay to the service provider based on the delivery of the services according to the SLAs in the contract between the service provider and the customer. The revenue is not fixed at it may change with potential reduction in payments or even penalties, depending on the service quality, e.g. too high query latency. Operational cost is the cost of resources used to run the service, e.g. the payment to the IaaS. Hence, the research question in ICDC is how to manage resources and workloads in the system to maximize SLA-based profit, which is SLA-based revenue minus operational cost. We believe applying SLA-based profit optimization to all system components opens up many interesting research challenges and opportunities.

**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.

ICDC continuously monitors the system and keeps track of crucial information, such performance and health of the storage engines, workload/data characteristics and statistics, performance of certain system modules. All this information is store in a separate store, called **System Monitor Database (SMDB)**. The system implements a feedback loop to relevant ICDC modules to continuously optimize and tune the system modules and methods. SMDB provides valuable information to enable that feedback loop.

**The Workload Analysis and Optimization:** component is responsible for identifying the characteristics of the client workloads. The goal of this analysis is to find the optimal backend database node(s) to execute the workload by considering SLA and profit objectives. The workload analyzer is closely tied with the dispatcher component. The dispatcher receives queries (or jobs) from the **Distributed Query Processor** makes the decision to submit the query to one of the valid replicas. The validity of a replica is defined by data availability. Even a replica has the required data answer the query. It may not be the optimal resource, compared to the other valid replicas.

**The dispatcher:** considers the current status of the valid replicas by evaluating certain metrics, such as performance characteristics, current workload assignments etc. After dispatcher dispatches the query to a server, the query is received by the local **scheduler** of the server. CloudDB employs advanced scheduling techniques to assign scheduling priority to the jobs in the local queues in front of the servers. Those techniques consider both SLA constraints and also profit maximization for the service providers. The details of those advanced scheduling techniques and profit models are beyond the scope of this presentation.

**Internal query processing and optimization** component is responsible for making decisions at the individual data store level. One important problem is to identify feasible physical design and query plans (deployment time and run time) that satisfy the specification. Another problem is, given an objective function (e.g., minimize resource budget) in addition to the specification, to identify optimal physical design and query plans.

**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 a have a significant impact on the performance of the system and operational costs. Consider an example were two data partitions that both are used by similar type workloads and demanding SLAs in terms of query throughput. Co-locating those two partitions in the same database may result significantly worse SLA/cost performance than separating them into different database servers. The specific consistency requirement from the clients is another factor that is considered the Design Optimizer. Given the very dynamic nature of the cloud environment and the potential diversity of the clients, achieving design optimization is a highly-non-trivial problem.

**Automated data and workload migration:** among and within the data stores is one of the key capabilities in CloudDB. CloudDB uses both partitioning and replication based techniques to achieve scalability and availability. The workload of applications will evolve over the time. The 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. The decisions to migrate data and workloads are made based on various metrics, including performance, profit optimization, client SLA satisfaction.

**Capacity Planner:** The system should be able to scale in and out the resources with the changing workload characteristics. ICDC includes a capacity planner component that 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.

**Multitenancy Manager (MTM):**

Controlling the operational costs is a crucial goal for the service provider. Sharing the system resources among multiple tenants is an effective way to achieve that goal. A tenant is an entity that consumes the services provided by the system. The provider-consumer relationship is defined via SLAs. Traditional DBMSs do not provide effective solutions to support massive Multitenancy. Deciding how the data management resources should be shared among multiple tenants and how the tenants should be grouped together to share those resources at varying granularity levels are crucial decisions that need to be made in the system.

## 4. 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.

CloudDB's API layer is being designed in such a way to give data independence to the higher level applications. 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 in the market. An important observation to make is the confusion between the query language, SQL, and the data model - relational data model. In essence, SQL is a data access and manipulation language. There has been a recent movement in so called NoSQL direction, e.g., [17][18]. SQL does not define how the transactions, query processing, and execution engines are defined and handled. Therefore SQL 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 [19][20]. We also experienced in numerous real business cases that the users want to exploit the advantages of new data stores, such as key-value stores, by integrating them to their existing infrastructure. However, they desire to minimize the amount of application re-modeling and re-writing, as SQL and relational models are used in vast majority of the cases. Thus, a capability to continue relational modeling and SQL language constructs, even in a limited fashion, was received as a valuable option.

In addition to API level support, CloudDB will provide design and query optimization capabilities that decide which and how data objects and entities should be stored thereby completing the data independence over heterogeneous storage engines. By providing this level of data independence, CloudDB aims at removing the burden of making a decision on the "best" storage engine or data management products from the application owners.

CloudDB plans to achieve data independence as follows:

- The application owner builds logical data models (e.g. entity relationships), without worrying about physical data organization.

- In addition to logical data models, the application owner provides SLAs, such as response time, consistency, availability etc.

- The logical data model is automatically mapped to its physical organization on top of the cloud computing infrastructure.

- The application's queries are automatically transformed to execution plans that manipulate physically organized data.

- Due to physical constraints in scale-out clouds, data inconsistency may be inevitable [7]. Such inconsistency is automatically handled to meet data quality requirements.

- Since the application's workload evolves over time, the physical data is adaptively re-organized to ensure the application is scalable.

An important point to note here is the separation between the hidden and visible aspects of the system from the applications view point. For example, the applications can provide information about their transaction patterns. Then, the system can guarantee varying levels of consistency levels. This information is visible to the applications, while the query processing and data storage over various stores are abstracted away from the applications.

## 5. ACKNOWLEDGMENTS

We thank Michael Carey, Hector Garcia-Molina, and Jeffrey Naughton for their contributions.

## 6. REFERENCES

[1] Michael Stonebraker, Ugur Cetintemel, *One Size Fits All: An Idea Whose Time Has Come and Gone*, In Proc. of IEEE ICDE, 2005

[2] Netezza Corporation, http://www.netezza.com/

[3] Greenplum, http://www.greenplum.com/

[4] Aster Data, http://www.asterdata.com/

[5] Google BigQuery. http://code.google.com/apis/bigquery/

[6] MangoDB: http://www.mongodb.org/display/DOCS/MongoDB+Data+Modeling+and+Rails

[7] Davit Mytton. Choosing a non-relational database; why we migrated from MySQL to MongoDB, 2009

[8] Amazon Relational Database Service. http://aws.amazon.com/rds/

[9] Gartner, http://www.gartner.com/it/page.jsp?id=1210613, 2009

[10] D. G. Campbell, G. Kakivaya, and N. Ellis. Extreme scale with full sql language support in microsoft sql azure. In SIGMOD, 2010

[11] M. Brantner, D. Florescu, D. A. Graf, D. Kossmann, and T. Kraska.

Building a database on s3. In SIGMOD, 2008

[12] C. Curino, E. Jones, Y. Zhang, E. Wu, and S. Madden. Relational cloud: The case for a database service. MIT CSAIL Technical Report 2010

[13] D. Kossmann, T. Kraska, and S. Loesing. An evaluation of alternative architectures for transaction processing in the cloud. In SIGMOD, 2010

[14] H. Hacıgümüş, S. Mehrotra, and B. R. Iyer, "Providing database as a service," in Proc. of ICDE, 2002

[15] E. A. Brewer, *Towards Robust Distributed Systems*, Principles of Distributed Computing, Invited Talk, 2000

[16] Daniel J. Abadi, Samuel R. Madden, Nabil Hachem, *Column-Stores vs. Row-Stores: How Different Are They Really*, In Proc. of ACM SIGMOD, 2008

[17] MangoDB, http://www.mongodb.org/

[18] CouchDB, http://couchdb.apache.org/

[19] Hive, http://hadoop.apache.org/hive

[20] Google bigquery, http://code.google.com/apis/bigquery /