Renaissance in Data Management Systems: SQL, NoSQL, and NewSQL

Venkat N Gudivada¹, Dhana Rao¹ and Vijay Raghavan²
¹Marshall University, Huntington, WV 25755
²University of Louisiana at Lafayette, Lafayette, LA

Abstract

The recent emergence of a new class of systems for data management has challenged the well-entrenched relational databases. These systems provide several choices for data management under the umbrella term NoSQL. Making a right choice is critical to building applications that meet business needs. Performance, scalability and cost are the principal business drivers for these new systems. By design, they do not provide all of the relational database features such as transactions, data integrity enforcement, and declarative query language. Instead, they primarily focus on providing near real-time reads and writes in the order of billions and millions, respectively. This paper provides a unified perspective, strengths and limitations, and use case scenarios for the new data management systems.

Keywords: NoSQL, SQL, NewSQL, Data Management

1 Introduction

Until recently, Relational Database Management Systems (RDBMS) were the mainstay for managing data. Underlying the RDBMS is the relational model for structuring data and SQL query language for data manipulation and retrieval. Both the relational data model and SQL have formal mathematical basis. The declarative nature of SQL made RDBMS popular as it enabled users with no technical background to query and manipulate data.

RDBMS also provide built-in support for data integrity enforcement, user authentication and role-based access control, data backup and recovery, and ACID support for transactions (see sidebar). RDBMS are the lifeline for many businesses and drive millions of websites. According to an IDC report [1], the RDBMS market revenue in 2011 was $26 billion and estimated to reach $41 billion by 2016.

Though RDBMS are a perfect fit for many applications, they maybe less suitable or an expensive fit for certain applications. An array of new systems for data management have emerged in recent years to address the needs of such applications [2, 3]. For example, Facebook uses an in-memory caching solution for constructing a distributed key-value system to support the world’s largest social network [4]. This key-value system services billions of requests per second by delivering data from a database which stores trillions of items.

*To appear in IEEE Computer.
By design, these new database systems do not provide all the RDBMS features. They principally focus on providing near real-time reads and writes in the order of billions and millions, respectively. Their features vary widely and there are no standards yet. They use different data models, some do not provide database transactions, while others do not use SQL. They are referred to by various names including NoSQL, NewSQL, and non-RDBMS. To avoid the misconception that NoSQL systems eschew SQL, they are also referred to as Not only SQL.

Currently there are over 220 systems for data management and new ones are introduced routinely [5]. In this paper, we use the umbrella term Modern Database Systems to refer to them. Table 1 shows a taxonomy of these systems. Among the systems corresponding to the NoSQL subclasses, there is no clear distinction as functional features rapidly advance and overlap. The trend towards all encompassing, cross-cutting functionality is perceived as a means to market dominance and this is accelerating the blurring of lines between the subclasses.

Market Research Media estimates that NoSQL products will generate $14 billion in revenues over the period 2013 – 2018 [6]. NoSQL systems created both opportunities for innovation as well as challenges to the well established RDBMS vendors. The latter are introducing new features into their products to mitigate the risk of losing customer base. On the other hand, NoSQL vendors are introducing features in an ad hoc manner and rushing their products to the market.

Because of the rivalry between the established RDBMS and new database vendors, there is confusion and hype in the marketplace. There is also disagreement on terminology, taxonomy, and classification of new data management systems. This is further exacerbated by the fast evolving product landscape and vested vendor interests.

The primary goal of this paper is to provide an overview of modern database systems and help the reader determine candidate systems that match the needs of an application.

2 Business Drivers for the NoSQL Systems

Performance, scalability and cost are the principal business drivers. Special needs of Big Data, Web 2.0 and mobile computing applications have driven the proliferation of NoSQL systems. These applications require functionality to support tasks such as real-time (a) predictive analytics, (b) personalization, (c) dynamic pricing, (d) superior customer service, (e) fraud and anomaly detection, (f) order status through supply chain visibility, and (g) Web server access log analysis. They exhibit a preponderance of insert and retrieval operations on a very large scale. This entails efficiently supporting insert and read operations to the exclusion of update and delete. Other functionality that is important for some applications includes:

- Big Data applications deal with large, heterogeneous and sparse data.
- From an economic standpoint, horizontal scaling using processors built from commodity hardware becomes a necessity.
- The ability of the network partitioned parts of a system to operate independently amid network failures is crucial.
<table>
<thead>
<tr>
<th>Database Class</th>
<th>Subclasses</th>
<th>Distinguishing Characteristic</th>
<th>Application Use-cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDBMS</td>
<td>Row-oriented RDBMS</td>
<td>Optimized for both reads and writes to support online transaction processing (OLTP).</td>
<td>Suitable for applications that require high data integrity, support for transactions, and fine-grained access control.</td>
</tr>
<tr>
<td></td>
<td>Column-oriented RDBMS</td>
<td>Efficient reads to support online analytical processing (OLAP).</td>
<td>Suitable for applications such as data warehousing, decision support, and business intelligence (BI).</td>
</tr>
<tr>
<td></td>
<td>NewSQL</td>
<td>Optimized for both reads and writes. Aim to provide same scalable performance as that of NoSQL systems.</td>
<td>NewSQL applications are characterized by short-lived transactions that touch a small subset of data, do not require large distributed joins, and prevalence of recurring parameterized queries.</td>
</tr>
<tr>
<td>NoSQL</td>
<td>Key-value</td>
<td>Storing key-value pairs in a way to guarantee fast retrieval. Query latency is independent of data size.</td>
<td>Key-value applications are characterized by the need for real-time processing of extremely large data, horizontal scalability, high reliability and availability. They treat data as monolithic blob and the primary query mechanism is key-based lookup.</td>
</tr>
<tr>
<td></td>
<td>Column-family</td>
<td>Efficiently storing sparse, non-transactional, heterogeneous data to support partial record access.</td>
<td>Column-family applications are characterized by the need for flexible and evolving database schema, horizontal scalability, and tolerance to both network failures and temporary data inconsistency.</td>
</tr>
<tr>
<td></td>
<td>Document model</td>
<td>A natural extension of key-value systems to manage semi-structured, arbitrarily nested hierarchical document data. The data is mostly organized in the form of key-value pairs in JSON format.</td>
<td>Applications are characterized by the need for flexible schema to accommodate high data variability from one record to another as in healthcare records and derivative securities.</td>
</tr>
<tr>
<td></td>
<td>Graph model</td>
<td>Efficiently store and query relationship-rich data.</td>
<td>Applications are characterized by queries that require graph traversals, and identifying subgraphs and cliques based on relationship types.</td>
</tr>
<tr>
<td></td>
<td>Native XML</td>
<td>Efficiently storing and retrieving hierarchically structured heterogeneous data with high variability from one record to another.</td>
<td>Applications are characterized by the need for standards-based technologies, increased development productivity, enabling nontechnical staff to build and maintain applications, and one data model across all application layers.</td>
</tr>
</tbody>
</table>

Table 1: Modern database management systems fall into two broad classes: RDBMS and NoSQL. All systems corresponding to the subclasses of the RDBMS class are based on the relational data model and the SQL query language. In contrast, systems corresponding to the NoSQL subclasses vary widely in terms of data models, query languages, support for database transactions, application programming interfaces, and granularity of security features. The third column describes distinguishing features of each subclass. Typical use-cases are indicated in the fourth column.
Non-overlapping, automatic data partitioning is necessary to address data volumes and scalability.

As data volumes mandate distribution of data across the nodes of a cluster, other issues such as distributed system coordination, failover, and resource management overshadow consistency and transaction features.

Simple data replication is mandatory to ensure high availability.

Built-in support for versioning and compression is imperative in some applications.

A complete database schema does not exist upfront and evolves over time.

Partial record updates are the norm in some cases.

ACID-compliant transactions are an overkill and only relaxed consistency is required.

Near real-time query execution is essential.

Multiple query methods ranging from simple REST API to complex and ad hoc queries are required.

Interactive processing of ad hoc queries mandates massive parallel computation using schemes such as MapReduce.

NoSQL systems meet the above needs to varying degrees. They offer scalability, schema flexibility, and query performance. They are scalable because they store and manage data distributed within the nodes of a cluster or nodes across clusters. Clusters can easily be scaled horizontally by adding more commodity computers. Some systems make use of massive parallelism of MapReduce natively to achieve performance. Many NoSQL systems do not require that the database schema be completely specified up front. The schema evolves to accommodate new data requirements. Most NoSQL systems are open source.

NoSQL systems functionality varies from one system to another significantly. Furthermore, they are evolving rapidly, making it difficult to state their functionality in an absolute sense. For example, (a) Riak is highly scalable and available, (b) MongoDB's defining characteristic is managing deeply nested structured documents and computing aggregates on them, (c) Neo4j excels at managing data that is rich in relationships, (d) Redis is known for its data structures and elegant query mechanisms, (e) HBase is rigid and less reliable, but offers extreme scalability, good at bulk loading data from MapReduce jobs, and leverages existing HDFS installations, and (f) HBase and MongoDB strive for consistency and partition tolerance, whereas CouchDB and Cassandra emphasize availability and partition tolerance.

NoSQL systems’ business drivers are also responsible for the emergence of column-oriented RDBMS and NewSQL systems under the RDBMS class (see Table 1). Column-oriented RDBMS use optimized storage engines to exclusively support efficient read operations required in OLAP applications. Some NewSQL systems are built from the ground up to operate in a distributed cluster of shared-nothing nodes. Others feature a sharding middleware layer to automatically split data across multiple nodes of a distributed cluster. Yet others have focused on highly optimized storage engines for SQL. For example, TokuDB uses fractal tree index data structure, whose time for search and sequential access is same as that of the B-tree, but time for insertions and deletions is asymptotically faster than a B-tree.
Applications Development Using NoSQL Systems

The NoSQL applications development process is somewhat at odds with that of the RDBMS. Relational databases view data as a strategic and shared corporate resource with well-established policies and procedures for data governance and quality control. In contrast, NoSQL systems tend to promote data silos, each geared towards meeting the performance and scalability requirements of just one or more applications. However, it should be noted that data silos are often dictated by organizational issues, not technological ones.

In RDBMS, data normalization refers to the process of dividing tables into smaller ones to minimize data redundancy. Denormalization is the opposite process. Data duplication and denormalization are accepted NoSQL practices. Duplication occurs due to denormalization and helps to simplify and optimize query processing through elimination of database table joins. Though it is not a standard practice, one can duplicate and denormalize data in an RDBMS just as easily in key-value systems.

Some NoSQL systems do not allow specification of constraints on data values. These practices run against the RDBMS approach to enterprise data management, redundancy control, and integrity constraints enforcement [7].

NoSQL databases typically require a developer-centric approach from the application inception to completion. Database schema is designed by the application architect or developer. The architect begins by identifying application queries and structures the data model to efficiently support these queries. There is a strong coupling between the database schema and application queries. Any changes to the queries will necessitate changes to the database schema. In the RDBMS approach, the database schema is designed by data architects and data modelers. Though the physical data model of an RDBMS maybe optimized for certain queries and transactions, the same logical data model works fine for any ad hoc queries. NoSQL data modeling is in stark contrast to the time-tested RDBMS principles of logical and physical data independence.

NoSQL systems model many-to-many-relationships with links and extracting these relationships requires joins, which are not typically supported. The relationships are synthesized by implementing the join operation in the application code. Also, the two-phase commit (2PC) required for implementing database transactions needs to be provided by the application code. However, it is not always possible to correctly implement joins or 2PC in the application as it will eliminate the scalability benefits of NoSQL systems. In contrast, RDBMS provides these as built-in features. Many NoSQL applications embrace eventual consistency to support extremely fast insert and read operations, whereas RDBMS typically provides strict consistency. Some RDBMS provide options for specifying desired consistency level.

In the RDBMS approach, design details such as data partitioning are transparent to the database client programs. RDBMSs' try to hide data layout details from the perspective of correctness, but not from the performance viewpoint. In comparison, client programs need to be aware of such details in the case of some NoSQL systems. Key-value systems prioritize performance over application-level correctness. Since not all NoSQL systems provide declarative query languages, often queries need to be implemented by writing programs.

Though NoSQL systems can be deployed on a single computer, typical enterprise scale deployments target distributed cluster computing platforms. Installing, configuring, and
maintaining NoSQL systems requires in-house personnel with advanced technical expertise. Deploying NoSQL applications on cloud platforms is a way to lessen this need.

4 Narrowing Down Database Management System Choices

NoSQL systems are like a double-edged sword. On the one hand, they provide many choices so that one may choose the right system that perfectly matches the application requirements and constraints. On the other hand, it is quite a challenge to determine which system is the right match. This problem is further exacerbated by rapidly evolving features. NoSQL and NewSQL systems are converging to provide multi-model systems. For example, MarkLogic NOSQL system features functionality to operate as a native XML database, document database, RDF store, and full-text search engine. FoundationDB, another NoSQL system, can be used as an RDBMS, a key-value store, and a document database.

Answers to the following questions is a starting point to narrow down the data management system choices for a given application:

- Type of data to be managed - structured, semi-structured, unstructured, or a mix of these types? Native XML and document databases are better at dealing with structured and semi-structured data.
- Can the application data be stored with one data model or multiple models? Graph databases are suitable for data that can be modeled as a graph and support only this data model. Systems such as ArangoDB and Aerospike natively support multiple data models.
- Is the data sparse? For sparse data, column-family systems such as Cassandra or a document database such as MongoDB are a better fit than traditional RDBMS.
- Type of query language desired - declarative, procedural, or both? Some systems provide a broad range of query languages from low-level programmatic access to high-level declarative querying.
- Are ad hoc queries so prevalent on semi- and unstructured data that a native MapReduce framework is required? Some systems require writing map and reduce functions in languages such as JavaScript to execute ad hoc queries.
- Are the users geographically distributed and data volume is very high to warrant data partitioning? Systems that support horizontal scaling are essential for this scenario.
- Are the read or write throughput levels too high for a conventional RDBMS to handle? For high throughput reads and writes, key-value, document, and column-family systems provide more options.
- Can you afford to lose data now and then? Different approaches to consistency has implications such as lost writes, flip-flopping between new and old versions of data.
- What is the minimal level of consistency essential for reads and writes? Depending on the choices, different systems exhibit varying behaviors.
What are the consequences of eventual consistency? Applications maybe forced to provide merge and reconciliation functions when replicated copies of data diverge.

Is automated failover with no service interruption imperative? When availability is lost, a system may switch to read-only mode for a subset of users/data. In the worst case, the system becomes completely unreachable. In other cases, read or write availability maybe lost while still maintaining consistency.

What type of replication is needed? Some systems provision for specifying the number of replicas and the replication mode (synchronous vs asynchronous).

Is an algorithmic approach needed for data partitioning with automated redistribution to avoid hotspots and performance bottlenecks? Some systems provide this functionality out of the box whereas others require specifying data sharding policies through application code.

Does the application load fluctuate unpredictably to require provisioning resources dynamically without service interruption? Cloud-hosted systems are a better fit for this scenario and for situations where local expertise is limited or unavailable for system management.

What type and level of user authentication and authorization are required? Some systems assume trusted environment for their operation whereas others such as MarkLogic Server provide fine-grained access control.

Is versioning and compression of data required? Not all systems provide this functionality.

Consistency (all copies of the same data are the same at the same time), availability (every service request receives a response), and partition tolerance (system continues to operate despite partial system failure or arbitrary message loss) are the three primary concerns that determine which data management system is suitable for a given application. The CAP theorem states that it is impossible for a distributed computer system to achieve all these three features at the same time [8, 9]. For example, MongoDB and Redis provide consistency and partition tolerance, and do not guarantee availability. Others such as DynamoDB and Cassandra provide availability and partition tolerance, and provide no data consistency guarantees. RDBMS provide and prioritize consistency and availability over partition tolerance.

The CAP theorem should not be viewed as selecting any two from the triad consistency, availability, and partition tolerance – 2 of the 3 view. The degree to which these three features are available in a system should be viewed as spanning a spectrum rather than as binary values. Furthermore, the choice between consistency and availability can occur many times within the same system at fine granular levels.

Use the answers to the questions above to come up with a small list of candidate choices. Also, consider the CAP theorem implications and further narrow down the choices.
5 Defining Characteristics of Modern Database Systems

According to DB-Engines [5], there are over 220 systems for data management grouped into 13 categories. However, just seven classes (Relational, Key-value, Document, Column-family, Graph Data Model, NewSQL, and Native XML) represent almost the entire database market.

5.1 Relational Systems

Relational Database Management Systems (RDBMS) were developed over the last four decades and have withstood the test of time for online transaction processing applications (OLTP). RDBMS provide several standard features for data management (a) a declarative query language (SQL) for data definition, loading, retrieval and manipulation, (b) transaction management, (c) authorization and access control, and (d) backup and recovery.

RDBMS work well with fixed database schema and preserve data integrity through constraints and triggers. They are well-known for ACID-compliant transactions. Though database stored procedures help minimize data movement by executing application logic within the database, they are not a substitute for native MapReduce.

High availability in relational systems is achieved through replication and data partitioning on disk systems. Likewise, enhanced performance can be attained through vertical scaling. However, these approaches are expensive and also run into limits. Schema rigidity and unacceptable query latency are often cited as relational database limitations for Web 2.0 applications. Nonetheless, their revenue will reach $41 billion in 2016 and this represents 93% of the entire database market [10].

Typically RDBMS storage engines are optimized for row-level processing (row-oriented RDBMS subclass in Table 1). Even to change one value in a row, the entire row is fetched into main memory from disk. Though this storage model suits OLTP applications, computing aggregates on table columns in real-time becomes difficult as the data size grows. Column-oriented RDBMS use a storage model that is optimized for efficient computation of column aggregates to meet the requirements of Online Analytical Processing (OLAP) applications. MonetDB, MonetDB/X100, C-Store, and InfiniDB are examples of column-oriented RDBMS.

There are over 80 open source and commercial RDBMS [5]. Oracle, IBM DB2, Microsoft SQL Server, SAP HANA, Teradata, PostgreSQL, and MySQL are most widely used. They are suitable for applications that require data integrity preservation, authentication and fine granular access control, declarative query languages and transactions.

5.2 Key-value Systems

Data is stored as key-value pairs. The value can be anything and is treated as opaque binary data. The key is transformed into an index using a hash function. Partitioning schemes other than hashing are also often used, especially in systems that store values in sorted order on disk. These methods of storing data entails fast retrieval and the query latency is independent of data size.

The defining characteristics of key-value databases include real-time processing of Big Data, horizontal scalability, reliability and high availability. There are over 40 key-value systems [5]. Redis, Memcached, Riak, DynamoDB, and Ehcache are more popular. Key-
value systems differ widely in functionality and performance. They vary in architecture used (master-master, master-slave, and client-server), in-memory database feature, disk persistence, data structures, transactions, sharding, replication, native MapReduce, and client access methods.

Use cases for key-value systems include applications where response time is in the order of milliseconds and the primary query mechanism is key-based lookup. Such applications include (a) session management for Web applications, (b) configuration management; distributed locks, (c) messaging, (d) personalization of user experience and providing engaging user interactions in social media, mobile platforms, and Internet gaming, (e) real-time bidding and online trading (f) Ad servers, (g) recommendation engines, and (h) multi-channel retailing and eCommerce.

5.3 Column-family Systems

Column-family databases are column-centric. Conceptually, a column-family database is like an RDBMS with an index on every column. However, column-family databases do not incur the overhead of traditional RDBMS for column-centric processing. It is also useful to think of column-family databases as nested key-value systems. Column-family databases are also called wide-column databases.

A set of related columns is called a column family. A column family is similar to the column concept in RDBMS. These databases require predefining column families, but not columns within a family. A column family may contain any number of columns of any type of data, as long as the latter can be persisted as byte arrays. Columns in a family are logically related to each other and are physically stored together. Performance gain is achieved by grouping columns with similar access characteristics into the same family. Database schema evolution is achieved by adding new columns to column families. When a value changes, it is stored as a new version differentiated from other versions by timestamp values. Partial record access contributes to dramatic performance improvements for certain applications. Column-family databases perform aggregate operations such as computing maxima and minima on large datasets with extreme efficiency.

Cassandra and HBase are the most widely used column-family databases. Other prominent systems in this category include Bigtable, Apache Accumulo, Hypertable, and Sqrrl. All these systems are based on the master-master, shared-nothing architecture. They differ in support for transactions, sharding, replication, native MapReduce, and client access methods.

Column-family databases are appropriate for applications which are characterized by (a) flexible and evolving database schema, (b) data heterogeneity, (c) sparse and non-transactional data, (d) tolerance to temporary inconsistency, (e) versioning, (f) native MapReduce processing, (g) partial record access, and (h) high speed insert and read operations. Cassandra’s use for time series data management in financial services industry exemplifies a typical use case.
5.4 Document Database Systems

Document databases are used for managing semi-structured data mostly in the form of key-value pairs structured as JSON documents [11]. Each document is a record with potentially varied attributes. Fields capture semi-structuredness and variability in documents. These systems allow efficient search on JSON fields through appropriate indexing.

There are over 30 document databases [5]. MongoDB, CouchDB, Couchbase, RavenDB, GemFire, RethinkDB, FoundationDB, and OrientDB are representative systems in this category. They differ in the type of system architecture employed, and support for transactions, sharding, replication, native MapReduce, and client access methods.

Document databases are similar to key-value systems. For example, Couchbase is both a key-value and document database. In the key-value mode, values are stored as opaque objects. In the document-mode, they are stored as JSON documents.

Document databases often integrate with full-text search libraries and services such as Solr, Lucene, and ElasticSearch. For example, ElasticSearch integrates with MongoDB and provides real-time response to document queries in JSON format.

Document databases are ideal for applications that (a) require flexible schema, (b) the number and the type of record instances vary widely from each other as in derivative securities and healthcare records, (c) data types and the number of values for a field vary from one record instance to another, (d) documents contain deeply nested fields, and (e) computing aggregates across document collections is a major query type.

5.5 Graph Data Model Systems

Graph databases are not so much about data volumes and availability, but more so about managing rich relationships in the data. A graph data model is at the heart of graph databases [12]. A graph consists of vertices and edges. Vertices represent entities and the relationships between entities are modeled as edges. Both vertices and edges can have attributes. Queries in graph databases include the shortest path between two vertices, finding clusters, and community detection.

There are over 30 graph database systems. Neo4j, Titan, Sparksee, Graph, and Allegro-Graph are representative systems in this category. They differ in the type of system architecture employed, and support for transactions, sharding, replication, native MapReduce, and client access methods.

Graph database applications include geospatial processing, recommendation engines, social networks analysis, metabolic and protein-protein interaction networks in systems biology. Graph databases are also used in other industries including airlines, freight companies, healthcare, retail, gaming, oil and gas. They are also popular for implementing access control and authorization subsystems for applications that serve millions of users.

5.6 NewSQL Systems

NewSQL systems have evolved from the traditional RDBMS and influenced by NoSQL systems. They are also known as multi-model systems as they support more than one data model. PostgreSQL-SC, VoltDB, VoltCache, ArangoDB, Aerospike, FoundationDB, and Spanner are representative systems in this category. They differ in system architecture (master-
master and multi-level masters), and support for transactions, sharding, replication, native MapReduce, and client access methods.

Applications that critically depend on RDBMS functions, but also require some NoSQL features are suitable candidates for using NewSQL.

5.7 Native XML Database Systems

Native XML databases are the oldest and most mature class of NoSQL systems. They can store and query a wider range of data types than any other NoSQL database. They excel at managing structured, semi-structured, and unstructured documents – mixed content. They feature an expressive and extensible data model which can be validated against an XML Schema or DTD. Complex data validation that is beyond the scope of XML Schema can be specified using Schematron, an ISO/IEC standard. Schematron is a rule-based validation language for making assertions about the presence or absence of patterns in XML documents.

The term native XML databases is used to distinguish them from the add-on approaches to XML data management provided by several RDBMS including IBM DB2, Oracle, PostgreSQL, and Microsoft SQL Server. The latter feature a new datatype called XML which enables storing XML in its natural hierarchical form. Queries can be specified using XQuery, SQL, or a combination of both.

XPath and XQuery provide declarative means for querying and updating XML databases. An extension to the XQuery/XPath language specifies how full-text search queries be specified as XQuery functions. XProc is a declarative language for pipeline processing of XML documents. XForms is another declarative language for building client applications that use a model-view-controller architecture. XSL-FO is a document formatting standard for specifying print formats for XML documents. Finally, XSLT is a declarative language used to specify transformation of an XML document into another XML document.

XML databases integrate well with the Web standards of the World Wide Web consortium (W3C). XML, XML Schema, XPath, XQuery, XQuery/XPath Full-text Search, XProc, XForms, and XSL-FO are W3C standards. XSLT 3.0 has W3C Last Call Working Draft status.

XML databases reduce applications development time through the use of standards, elimination of data mappings between application layers by using the same data model, and enabling nontechnical staff to perform development and maintenance work. Application portability is enhanced by using standards-based technologies. Native XML databases also enable fine-grained specification of security policies and access control.

MarkLogic, eXist-db, BaseX, Tera Text Database System, Snapbridge FDX Cross Media Server, and Sedna are representative systems in this category. They are suitable for applications that require mixed content management based on mature and standards-driven technologies. Applications that involve very large number of documents, long-running transactions as in workflow management, rapidly evolving database schema, or querying hierarchical data are ideal candidates for native XML databases. Other application areas include single source and multi-channel publishing, genetics, healthcare, insurance, data integration, messaging, and data-driven Websites.
6 Data Management Trends

Modern database management systems coupled with increased adoption of cloud hosting is contributing to innovative approaches to data management. As traditional RDBMS vendors continue to add more NoSQL features, NoSQL vendors are striving to bring more stability, robustness and transactions to their products. Rapid evolution and cross-over functionality between NoSQL and RDBMS systems will contribute to gradually blurring the distinction between them. Furthermore, they are converging to provide multi-model systems. For example, one system may provide all the features of key-value, document, and graph databases. This scenario will provide the impetus for cloud-hosted Database as a Service model.

NoSQL systems are likely to continue to maintain their niche in schema evolution, virtually unlimited horizontal scalability, and near real-time query performance. Their increased use may come from organizations employing an array of DBMS, each naturally suited to the type of data and application requirements. Migration from RDBMS to NoSQL systems will be eased by the JSON format to some extent.

As DRAM prices continue to fall, In-Memory Databases (IMDB) will help to offset the limitations of RDBMS in scalability and performance areas to certain extent. Systems such as SAP HANA, Aerospike, VoltDB and McObject have already embraced this principle.

7 Research Issues in Data Management

Security is a major research issue for NoSQL systems. Security risks range from an assumption that NoSQL systems run in a trusted environment and no authentication is required to clear text transmission to no data encryption on disk. Authorization and fine-grained access control is another security risk. Though RDBMS solutions to these problems apply to NoSQL systems in principle, the differences in data model, query languages, and client access methods of NoSQL warrant further investigation.

Typically, a NoSQL system employs one data model — key-value, document, graph, column-family, or XML. However, recent move towards multi-model NoSQL systems is on the rise as a means to market expansion. Though such systems employ multiple data models, an application may choose only one data model. If the application chooses more than one data model, there is no transparent support for queries to execute across data stored in multiple models. Though these issues appear to be similar to query processing in distributed databases, the data model heterogeneity requires further investigation. It is interesting to note that the issues here are similar to providing a unified programming environment in multi-paradigm programming languages such as Oz.

Generally, NoSQL systems for enterprise-scale applications are deployed on a centralized or distributed cluster computer. Due to lack of in-house technical expertise for installing and running NoSQL systems and based on economic considerations, some organizations may opt to use NoSQL systems running on a cloud under Database as a Service (DBaaS) model. Research is needed to operate NoSQL systems under the DBaaS model.

The next logical step for the DBaaS is the Data Services Facade model. The latter provides data services for multi-model data in a transparent manner in public or private cloud environment. Research advances in the three areas mentioned above are needed to make Data Services Facade a reality.
Finally, standards are essential to prevent vendor lock-in through application portability. Furthermore, existence of standards is an indication of the maturity of the associated technologies. Active participation of professional societies and vendors is needed to establish standards. Standardization is vital to the long-term sustainability and advancement of the data management field.

8 Conclusions

Unprecedented data volumes, heterogeneous data, performance and scalability requirements of modern applications challenged RDBMS as the only approach for data management. This ushered in a spectrum of choices. Levels of data consistency, high availability, and partition tolerance are the three primary concerns that determine which data management system is suitable for a given application.

Several research issues remain to be addressed for NoSQL systems to achieve the level of robustness and maturity of RDBMS. This is an exciting time for database researchers to work on these issues and develop an assortment of choices for data management.

9 Sidebar – Data Management Terms and Concepts

ACID Atomicity, Consistency, Isolation and Durability (ACID) comprise a set of properties that are used to characterize database transaction execution. Atomicity refers to ensuring that all the steps in a transaction are executed as one logical step. The consistency property guarantees that execution of a transaction does not take the database to an invalid state. All clients see the same data even with concurrent updates. The isolation property ensures that the effect of a set of concurrently executing transactions on the database is same as executing them in some serial order. The durability property assures that the effects of a committed transaction are permanent against any type of failure.

BASE ACID properties are to RDBMS as BASE is to NoSQL. BASE refers to Basic Availability, Soft state, and Eventual consistency. Basic availability insures that system services its users amid network failures. Eventual consistency means that if no further updates are made to a given updated database item for long enough period of time, all clients will see the same value for the updated item. In other words, an updated data item may not be available at all nodes of a distributed system. Soft state refers to state change without input, which is required for eventual consistency.

BSON Binary Javascript Object Notation (BSON) is a format for binary-coded serialization of JSON-like documents. See JSON.

Consistency This property ensures that all transactions transform a database from one valid state to another. Once a transaction updates a database item, all database clients (i.e., programs and users) will see the same value for the updated item.
**Database as a Service** A cloud operator provides traditional database administration functions for a fee. Resources are dynamically provisioned to meet the application workload.

**Database Transaction** A unit of work executed as an indivisible operation. See ACID.

**Hashing** Generating a fixed-length hash code as a unique and shortened representation for a given piece of data using a hash function.

**Horizontal Scaling** Accommodating increased workloads by simply adding new nodes. No changes to application code are made.

**JSON** JavaScript Object Notation (JSON) is a lightweight, text-based, language-neutral, open standard format for exchanging data between applications.

**MapReduce** A computational paradigm for processing massive datasets in parallel if the computation fits a three-step pattern: map, shard and reduce. The map process is a highly parallel one comprised of several processes. Each one processes a different segment of data and produces (key, value) pairs. The shard process collects the generated pairs, sorts and partitions them. Each partition is assigned to a different reduce process, which produces one result.

*Incremental MapReduce* enables leveraging previous computations and processing only the new data that has been added since the previous computation.

**Partition Tolerance** A distributed system is partition tolerant if it continues to provide services amid network failures.

**Replication** Multiple copies of the same data are stored on different nodes of a distributed system to improve data availability and query performance. When a data item is updated at one node, its copies at other nodes are either updated simultaneously (synchronous replication) or at a later time (asynchronous replication). Replication can be continuous or done according to a schedule.

**REST API** Representational State Transfer (REST) is a minimal overhead Hypertext Transfer Protocol (HTTP) API for interacting with software applications.

**Sharding** Partitioning data in a non-overlapping manner across the nodes of a distributed system. It is called *auto-sharding* when this task is done by a system in a manner transparent to the user.

**Shared-Nothing Architecture** A distributed computer architecture where each node is self-sufficient and acts independently to remove single point of resource contention or failure. Nodes share neither memory nor disk storage. In master-slave variant, one node is designated as the master and is responsible for cluster coordination and load balancing. In master-master variant, every node is capable of performing the master functions. Another variation is multi-level master, in which there is a master-slave configuration for each geographic zone and a universal master coordinates the activities of zone-level masters.
**Vertical Scaling** Adding more processing power to the same computer by increasing memory and number of CPUs and cores.

**Web 2.0 Applications** A new generation of Web applications which allow users to interact and collaborate with each other, and contribute content. Examples include social media applications, blogs, wikis, and folksonomies.

**References**


**Biographies**

Venkat N Gudivada is a professor and Interim Chair of the Weisberg Division of Computer Science at Marshall University, Huntington, WV, USA. His research interests are in database
management, information retrieval, high performance computing, and personalized eLearning. He received his Ph.D. in Computer Science from the University of Louisiana at Lafayette, Louisiana. He is a member of the IEEE Computer Society. He can be reached at gudivada@marshall.edu

Dhana Rao is an assistant professor of Biological Sciences at Marshall University, Huntington, WV, USA. Her research interests are microbial ecology and bioinformatics. She received her Ph.D. from the University of New South Wales, Sydney, Australia. She can be reached at raod@marshall.edu

Vijay V Raghavan is the Alfred and Helen Lamson/BoRSF Endowed Professor in Computer Science at the Center for Advanced Computer Studies and the Director of the NSF-sponsored Industry/University Cooperative Research Center for Visual and Decision Informatics, at the University of Louisiana at Lafayette, USA. His research interests are in data mining, information retrieval, machine learning and Internet computing. He is a senior member of the IEEE Computer Society. He can be reached at vijay@cacs.louisiana.edu