THE EXPERT'S VOICE® IN DATABASES

Pro Couchbase Development

A NoSQL Platform for the Enterprise

A HANDS-ON GUIDE TO LEARNING AND DEVELOPING WITH COUCHBASE

Deepak Vohra

Apress®

Pro Couchbase Development

A NoSQL Platform for the Enterprise

Deepak Vohra

Apress®

Pro Couchbase Development

ISBN-13 (pbk): 978-1-4842-1435-0

ISBN-13 (electronic): 978-1-4842-1434-3

Managing Director: Welmoed Spahr
Lead Editor: Steve Anglin
Technical Reviewer: Massimo Nardone
Editorial Board: Steve Anglin, Louise Corrigan, Jonathan Gennick, Robert Hutchinson, Michelle Lowman, James Markham, Susan McDermott, Matthew Moodie, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Gwenan Spearing, Steve Weiss
Coordinating Editor: Mark Powers
Copy Editor: Karen Jameson
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a Delaware corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com/9781484214350. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/. Readers can also access source code at SpringerLink in the Supplementary Material section for each chapter.

Contents at a Glance

Contents

About the Author

Deepak Vohra is a consultant and a principal member of the NuBean.com software company. Deepak is a Sun-certified Java programmer and Web component developer. He has worked in the fields of XML, Java programming, and Java EE for over ten years. Deepak is the coauthor of *Pro XML Development with Java Technology* (Apress, 2006). Deepak is also the author of the *JDBC 4.0* and *Oracle JDeveloper for J2EE Development, Processing XML Documents with Oracle JDeveloper 11g, EJB 3.0 Database Persistence with Oracle Fusion Middleware 11g*, and *Java EE Development in Eclipse IDE* (Packt Publishing). He also served as the technical reviewer on *WebLogic: The Definitive Guide* (O'Reilly Media, 2004) and *Ruby Programming for the Absolute Beginner* (Cengage Learning PTR, 2007).

About the Technical Reviewer

Massimo Nardone holds a Master of Science degree in Computing Science from the University of Salerno, Italy. He worked as a PCI QSA and Senior Lead IT Security/Cloud/SCADA Architect for many years and currently works as Security, Cloud and SCADA Lead IT Architect for Hewlett Packard Finland. He has more than twenty years of work experience in IT including Security, SCADA, Cloud Computing, IT Infrastructure, Mobile, Security, and WWW technology areas for both national and international projects. Massimo has worked as a Project Manager, Cloud/SCADA Lead IT Architect, Software Engineer, Research Engineer, Chief Security Architect, and Software Specialist. He worked as visiting a lecturer and supervisor for exercises at the Networking Laboratory of the Helsinki University of Technology (Aalto University). Massimo has been programming and teaching how to program with Perl, PHP, Java, VB, Python, C/C++, and MySQL for more than twenty years. He holds four international patents (PKI, SIP, SAML, and Proxy areas).

Massimo is the author of *Pro Android Games* (Apress, 2015).

CHAPTER 1

Why NoSQL?

NoSQL databases refer to the group of databases that are not based on the relational database model. Relational databases such as Oracle database, MySQL database, and DB2 database store data in tables, which have relations between them and make use of SQL (Structured Query Language) to access and query the tables. NoSQL databases, in contrast, make use of a storage and query mechanism that is predominantly based on a non-relational, non-SQL data model.

The data storage model used by NoSQL databases is not some fixed data model, but the common feature among the NoSQL databases is that the relational and tabular database model of SQL-based databases is not used. Most NoSQL databases make use of no SQL at all, but NoSQL does not imply that absolutely no SQL is used, because of which NoSQL is also termed as "not only SQL." Some examples of NoSQL databases are discussed in Table 1-1.

Table 1-1. *NoSQL Databases*

NoSQL Database	Database Type	Data Model	Support for SQL-like query language
Couchbase Server	Document	Key-Value pairs in which the value is a JSON (JavaScript Object Notation) document.	Supports N1QL, which is an SQL-like query language.
Apache Cassandra	Columnar	Key-Value pairs stored in a column family (table).	Cassandra Query Language (CQL) is an SQL-like query language.
MongoDB	Document	Key-Value pairs in which the value is a Binary JSON (BSON) document.	MongoDB query language is an SQL-like query language.
Oracle NoSQL Database	Key-Value	Key-Value pairs. The value is a byte array with no fixed data structure. The value could be simple fixed string format or a complex data structure such as a JSON document.	SQL query support from an Oracle database External Table.

This chapter covers the following topics.

- What is JSON?
- What is wrong with SQL?
- Advantages of NoSQL Databases
- What has Big Data got to do with NoSQL?
- NoSQL is not without Drawbacks
- Why Couchbase Server?
- Who Uses Couchbase Server and for what?

What Is JSON?

As mentioned in Table 1-1, the Couchbase Server data model is based on key-value pairs in which the value is a JSON (JavaScript Object Notation) document. JSON is a data-interchange format, which is easy to read and write and also easy to parse and generate by a machine. The JSON text format is a language format that is language independent but makes use of conventions familiar to commonly used languages such as Java, C, and JavaScript.

Essentially a JSON document is an object, a collection of name/value pairs enclosed in curly braces {}. Each name in the collection is followed by ':' and each subsequent name/value pair is separated from the preceding by a ','. An example of a JSON document is as follows in which attributes of a catalog are specified as name/value pairs.

```
{
"journal":"Oracle Magazine",
"publisher":"Oracle Publishing",
"edition": "January February 2013"
}
```

The name in name/value pairs must be enclosed in double quotes "". The value must also be enclosed in "" if a string includes at least a single character. The value may have one of the types discussed in Table 1-2.

Table 1-2. *JSON Data Types*

Type	Description	Example

string	A string literal. A string literal must be enclosed in "".	{
		"c1":"v1",
		"c2":"v2"
		}
		
		The string may consist of any Unicode character except " and \. Each value in the following JSON document is not valid.
		
		{
		"c1":""",
		"c2":"\"
		}
		
		The " and \ may be included in a string literal by preceding them with a \.
		The following JSON document is valid.
		
		{
		"c1":"\"",
		"c2":"\\"
		}

number	A number may be positive or negative, integer or decimal.	{
		"c1": 1,
		"c2": -2.5,
		"c3":0
		}

array	An array is a list of values enclosed in [ ].	{
		"c1":[1,2,3,4,5,"v1","v2"],
		"c2":[-1,2.5,"v1",0]
		}

true false	The value may be a Boolean true or false.	{
		"c1":true,
		"c2":false
		}

null	The value may be null.	{
		"c1":null,
		"c2":null
		}

object	The value may be another JSON object.	{
		"c1":{"a1":"v1", "a2":"v2", "a3":[1,2,3]},
		"c2":{"a1":1, "A2":null, "a3":true},
		"c3":{}
		}

The JSON document model is most suitable for storing unstructured data, as the JSON objects can be added in a hierarchical structure creating complex JSON documents. For example, the following JSON document is a valid JSON document consisting of hierarchies of JSON objects.

```
{
  "c1": "v1",
  "c2": {
    "c21":[1,2,3],
    "c22":
    {
      "c221":"v221",
     "c222":
     {
       "c2221":"v2221"
     },
     "c223":
      {
       "c2231":"v2231"
     }
    }
  }
}
```

What Is Wrong with SQL?

NoSQL databases were developed as a solution to the following requirements of applications:

- Increase in the volume of data stored about users and objects, also termed as big data.
- Rate at which big data influx is increasing.
- Increase in the frequency at which the data is accessed.
- Fluctuations in data usage.
- Increased processing and performance required to handle big data.
- Ultra-high availability.
- The type of data is unstructured or semi-structured.

SQL-based relational databases were not designed to handle the scalability, agility, and performance requirements of modern applications using real-time access and processing big data. While most RDBMS databases provide scalability and high availability as features, Couchbase Server provides higher levels of scalability and high availability. For example, while most RDBMS databases provide replication within a datacenter, Couchbase Server provides Cross Datacenter Replication (XDCR), which is replication to multiple, geographically distributed datacenters. XDCR is discussed in more detail in a later section. Couchbase Server also provides rack awareness, which traditional RDBMS databases don't.

Big data is growing exponentially. Concurrent users have grown from a few hundred or thousand to several million for applications running on the Web. It is not just that once big data has been stored new data is not added. It is not just that once a web application is being accessed by millions of users it shall continue to be accessed by as many users for a predictable period of time. The number of users could drop to a few

thousand within a day or a few days. Relational database is based on a single server architecture. A single database is a single point of failure (SPOF). For a highly available database, data must be distributed across a cluster of servers instead of relying on a single database. NoSQL databases provide the distributed, scalable architecture required for big data. "Distributed" implies that data in a NoSQL database is distributed across a cluster of servers. If one server becomes unavailable another server is used. The "distributed" feature is a provision and not a requirement for a NoSQL database. A small scale NoSQL database may consist of only one server.

The fixed schema data model used by relational databases makes it necessary to break data into small sets and store them in separate tables using table schemas. The process of decomposing large tables into smaller tables with relationships between tables is called database normalization. Normalized databases require table joins and complex queries to aggregate the required data. In contrast, the JSON document data model provided by NoSQL databases such as Couchbase provide a denormalized database. Each JSON document is complete unto itself and does not have any external references to other JSON documents. Self-contained JSON documents are easier to store, transfer, and query.

Advantages of NoSQL Databases

In this section I'll cover the advantages of NoSQL databases.

Scalability

NoSQL databases are easily scalable, which provides an elastic data model. Why is scalability important? Suppose you are running a database with a fixed capacity and the web site traffic fluctuates, sometimes rising much in excess of the capacity, sometimes falling below the capacity. A fixed capacity database won't be able to serve the requests of the load in excess of the capacity, and if the load is less than the capacity the capacity is not being utilized fully. Scalability is the ability to scale the capacity to the workload. Two kinds of scalability options are available: *horizontal scalability* and *vertical scalability*. With horizontal scalability or scaling-out, new servers/machines are added to the database cluster. With vertical scalability or scaling-up, the capacity of the same server or machine is increased. Vertical scalability has several limitations.

- Requires the database to be shut down so that additional capacity may be added, which incurs a downtime.
- A single server has an upper limit.
- A single server is a single point of failure. If the single server fails, the database becomes unavailable.

While relational databases support vertical scalability, NoSQL databases support horizontal scalability. Horizontal scalability does not have the limitations that vertical scalability does. Additional server nodes may be added to a Couchbase cluster without a dependency on the other nodes in the cluster. The capacity of the NoSQL database scales linearly, which implies that if you add four additional servers to a single server, the total capacity becomes five times the original, not a fraction multiple of the original due to performance loss. The NoSQL cluster does not have to be shut down to add new servers. Ease of scalability is provided by the shared-nothing architecture of NoSQL databases. The monolithic architecture provided by traditional SQL databases is not suitable for the flexible requirements of storing and processing big data. Traditional databases support scale-up architecture (vertical scaling) in which additional resources may be added to a single machine. In contrast, NoSQL databases provide a scale-out (horizontal scaling), nothing shared architecture, in which additional machines may be added to the cluster. In a shared-nothing architecture, the different nodes in a cluster do not share any resources, and all data is distributed (partitioned) evenly (load balancing) across the cluster by a process called sharding.

Ultra-High Availability

Why is high availability important? Because interactive real-time applications serving several users need to be available all the time. An application cannot be taken offline for maintenance, software, or hardware upgrade or capacity increase. NoSQL databases are designed to minimize downtime, though different NoSQL databases provide different levels of support for online maintenance and upgrades. Couchbase Server supports online maintenance, software and hardware upgrades, and scaling-out. As mentioned earlier, Couchbase Server provides ultra-high availability.

Commodity Hardware

NoSQL databases are designed to be installed on commodity hardware, instead of high-end hardware. Commodity hardware is easier to scale-out: simply add another machine and the new machine added does not even have to be of similar specification and configuration as the machine/s in the NoSQL database cluster.

Flexible Schema or No Schema

While the relational databases store data in the fixed tabular format for which the schema must be defined before adding data, the NoSQL databases do not require a schema to be defined or provide a flexible dynamic schema. Some NoSQL databases such as Oracle NoSQL database and Apache Cassandra have a provision for a flexible schema definition, still others such as Couchbase are schema-less in that the schema is not defined at all. Any valid JSON document may be stored in a Couchbase Server. One document may be different from another and the same document may be modified without adhering to a fixed schema definition. The support for flexible schemas or no schemas makes NoSQL databases suitable for structured, semi-structured, and unstructured data. In an agile development setting the schema definition for data stored in a database may need to change, which makes NoSQL databases suitable for such an environment. Dissimilar data may be stored together. For example, in the following JSON document the `c21` name has an array of dissimilar data types as value.

```
{
  "c1": "v1",
    "c21":[1,"c213", 2.5, null, true]
}
```

In contrast, a value in a relation database column must be of the schema definition type such as a string, an integer, or a Boolean. Flexible schemas make development faster, code integration uninterrupted by modifications to the schema, and database administration almost redundant.

Big Data

NoSQL databases are designed for big data. Big data is in the order of tens or even hundreds of PetaByte (PB). For example, eBay, which makes use of Couchbase stores 5.3 PB on a 532 node cluster. TuneWiki uses Couchbase to store more than one billion documents. Big data is usually associated with a large number of users and a large number of transactions. Viber, a messaging and VoIP services company handles billions of messages a month and thousands of ops per second with Couchbase for its big data requirements.

Object-Oriented Programming

The key-value data model provided by NoSQL databases supports object-oriented programming, which is both easy to use and flexible. Most NoSQL databases are supported by APIs in object-oriented programming languages such as Java, PHP, and Ruby. All client APIs support simple put and get operations to add and get data.

Performance

Why is performance important? Because interactive real-time applications require low latency for read and write operations for all types and sizes of workloads. Applications need to serve millions of users concurrently at different workloads. The shared-nothing architecture of NoSQL databases provides low latency, high availability, reduced susceptibility to failure of critical sections, and reduced bandwidth requirement. The performance in a NoSQL database cluster does not degrade with the addition of new nodes.

Failure Handling

NoSQL databases typically handle server failure automatically to failover to another server. Why is auto-failover important? Because if one of the nodes in a cluster were to fail and if the node was handling a workload, the application would fail and become unavailable. NoSQL databases typically consist of a cluster of servers and are designed with the failure of some nodes as expected and unavoidable. With a large number of nodes in a cluster the database does not have a single point of failure, and failure of a single node is handled transparently with the load of the failed server being transferred to another server. Couchbase keeps replicas (up to three) of each document across the different nodes in the cluster with a document on a server being either in active mode or as an inactive replica. The map of the different document replicas on the different servers in the cluster is the cluster topology. The client is aware of the cluster topology. When a server fails, one of the inactive replica is promoted to active state, and the cluster topology is updated, without incurring any downtime as is discussed in a later section.

Less Administration

NoSQL databases are easier to install and administer without the need for specialized DBAs. A developer is able to handle the administration of a NoSQL database, but a specialized NoSQL DBA should still be used. Schemas are flexible and do not need to be modified periodically. Failure detection and failover is automatic without requiring user intervention. Data distribution in the cluster is automatic using auto-sharding. Data replication to the nodes in a cluster is also automatic. When a new server node is added to a cluster, data gets distributed and replicated to a new node as required automatically.

Asynchronous Replication with Auto-Failover

Most NoSQL databases such as Couchbase provide *asynchronous* replication across the nodes of a cluster. Replication is making a copy of data and storing the data in a different node in the cluster. Couchbase stores up to three replicas. The replication is illustrated in Figure 1-1 in which a JSON document is replicated to three nodes in a Couchbase cluster. On each node the document is available either in Active state or as a passive Replica. If the Active document on a node becomes unavailable due to server failure or some reason such as power failure, a replica of the document on another server is promoted to Active state. The promotion from Replica passive state to Active Sate is transparent to the client without any downtime or very less downtime.

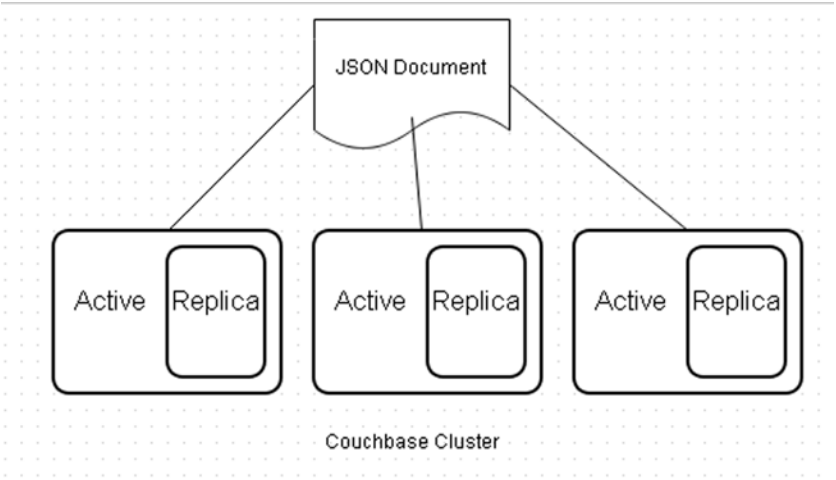

Figure 1-1. *Replication on a Couchbase Cluster*

Replication within a cluster provides durability, reliability, and high availability in the eventuality of a single node failure. The terms durability, reliability, and high availability seem similar but have different connotations. Durability is a measure of the time for which the data is not lost and is in a persistent state. Reliability is a measure of the operational efficiency of the database. Common measures of reliability are Mean Time Between Failure (MTBF=total time in service/number of failures during the same time) and Failure rate (number of failures/total time in service). High availability is the measure of time for which the database is available; Available time/(Available time+Not Available time).

Couchbase also supports Cross Datacenter Replication (XDCR), which is replication of data from one data center to another. In addition to failure recovery, XDCR provides data locality because, with the same data replicated across multiple data centers, it is more likely to find a cluster/node closer to a client. I cover XDCR in the section "Cross Data Center Replication."

"Asynchronous" implies that a server does not wait for the replication to complete before sending an ACK (acknowledgment) to the client. The difference between *synchronous* and *asynchronous* mode is explained next. In synchronous mode a data is replicated in the following sequence and illustrated in Figure 1-2.

1. Client sends a new data record to Server1.
2. The data record is stored in NoSQL database on Server1.
3. The data record is propagated to Server2 for replication.
4. The data record is stored in NoSQL database on Server2.
5. The Server2 sends ACK to Server1 that the data record has been replicated.
6. The Server1 sends ACK to Client that the data record has been replicated.

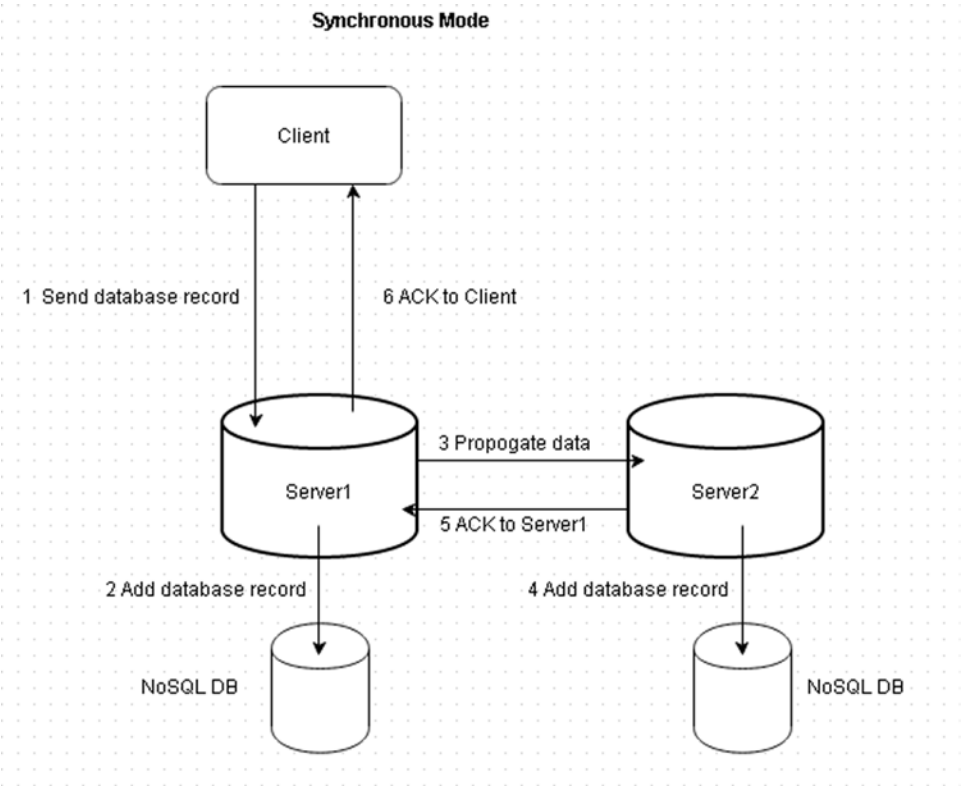

Figure 1-2. *Data Replication in Synchronous Mode*

In asynchronous mode, a data is replicated in the following sequence and illustrated in Figure 1-3.

1. Client sends a new data record to Server1.
2. The data record is stored in NoSQL database on Server1.
3. The data record is propagated to Server2 for replication.
4. The Server1 sends ACK to Client that the data record has been replicated.
5. The data record is stored in NoSQL database on Server2.
6. The Server2 sends ACK to Server1 that the data record has been replicated.

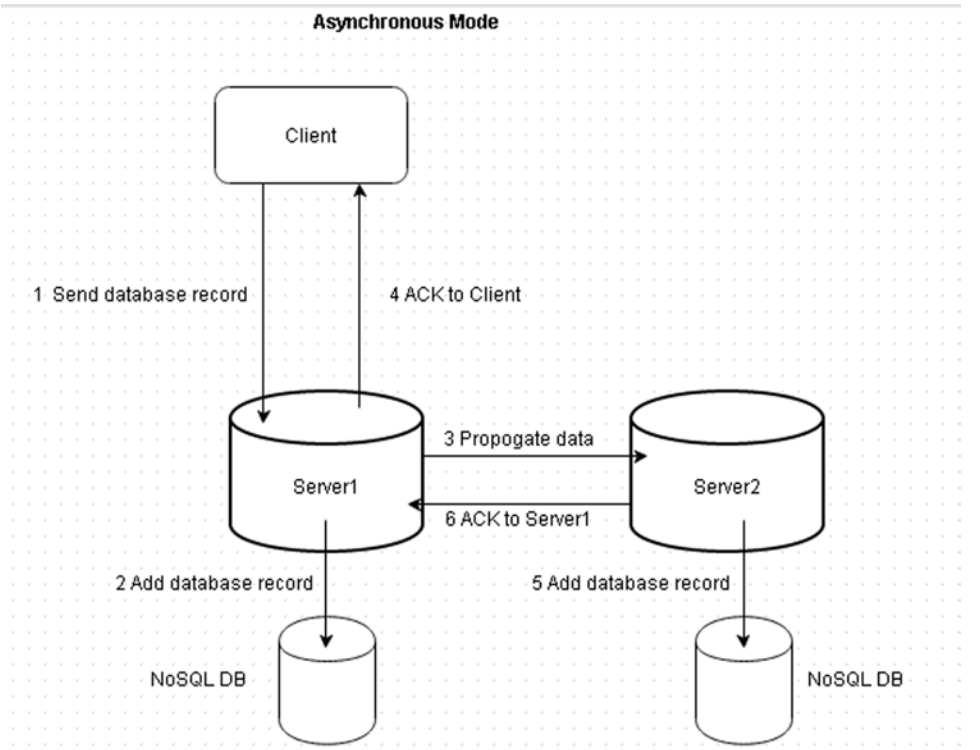

Figure 1-3. *Data Replicationin Asynchronous Mode*

Asynchronous mode prevents the latency associated with waiting for a response from the servers to which data is propagated for replication. But, the servers in the cluster could be an inconsistent state while data is being replicated. The client, however, gets an ACK for data replication before the consistent state is stored. Data in asynchronous mode is eventually consistent.

Caching for Read and Write Performance

Most NoSQL databases, including Couchbase Server, provide integrated object-level caching to improve read and write performance. With caching, applications are able to read and write data with a latency of less than a millisecond. Caching improves read performance more than it improves write performance.

Cloud Enabled

Cloud computing has made unprecedented capacity and flexibility in choice of infrastructure available. Cloud service providers such as Amazon Web Services (AWS) provide fully managed NoSQL database services and also the option to develop custom NoSQL database services. AWS has partnered with Couchbase to provide support and training to those running Couchbase Server on Amazon EC2 and Amazon EBS.

What Has Big Data Got to Do with NoSQL?

Though NoSQL databases may be used for storing small quantities of data, NoSQL databases were motivated by big data and the dynamic requirements of big data storage and processing. Couchbase Server is designed for big data with features such as scalability, intra cluster, and cross datacenter replication. In some of the examples in the book we shall use small quantities of data to demonstrate features and client APIs. The quantity of data stored or fetched may be scaled as required in a big data application. The same application that is used to stored ten lines of data in Couchbase may be modified to store a million lines of data. The same application that is used to migrate five rows of data from Apache Cassandra to Couchbase Server may be used to migrate a million rows of data. The performance of Couchbase Server does not deteriorate with increase in data processed.

NoSQL Is Not without Drawbacks

While much has been discussed about their merits, NoSQL databases are not without drawbacks. Some of the aspects in which NoSQL databases have limitations are as follows.

BASE, Not ACID

NoSQL databases do not provide the ACID (Atomicity, Consistency, Isolation, and Durability) properties in transactions that relational databases do.

- Atomicity ensures that either all task/s within a transaction are performed or none are performed.
- Consistency ensures that the database is always in a consistent state without any partially completed transactions.
- Isolation implies that transactions are isolated and do not have access to the data of other transactions until the transactions have completed. Isolation provides consistency and performance.
- A transaction is durable when it has completed.

NoSQL database provide BASE (Basically Available, Soft state, and Eventually consistent) transactional properties.

- Basically Available implies that a NoSQL database returns a response to every request though the response could be a failure to provide the requested data, or the requested data could be returned in an inconsistent state.
- Soft state implies that the state of the system could be in transition during which time the state is not consistent.
- Eventually consistent implies that when the database stops receiving input, eventually the state of a NoSQL database becomes consistent when the data has replicated to the different nodes in the cluster as required. But, while a NoSQL database is receiving input, the database does not wait for its state to become consistent before receiving more data.

Still New to the Field

The NoSQL databases are still new to the field of databases and not as functionally stable and reliable as the established relational databases.

Vendor Support Is Lacking

Most NoSQL databases such as MongoDB and Apache Cassandra are open source projects and lack the official support provided by established databases such as Oracle database or IBM DB2 database. Couchbase Server is also an open source project. Couchbase, however does provide subscription-based support for its Enterprise Edition server.

Why Couchbase Server?

Couchbase Server is a high-performance, distributed, NoSQL database. Couchbase Server provides several benefits additional or similar to those provided by some of the other leading NoSQL databases.

Flexible Schema JSON Documents

Interactive, real-time applications, processing unstructured data required to support a varying data model as the unstructured data does not conform to any fixed schema. Not all NoSQL databases are based on the JSON data model. In Couchbase Server, data is stored as JSON documents with each document assigned an Id. The JSON data storage and exchange format is a schema-less data model as discussed earlier and stores hierarchies of name/value pairs. The JSON document structure is not fixed and may vary from document to document and may be modified in the same document. The only requirement is that the document is a valid JSON document. A flexible schema data model does not require an administrator's intervention to modify schema, which could lead to downtime.

Scalability

While all NoSQL databases are scalable Couchbase's scalability feature has the following advantages.

- Adding and removing nodes is a one-click solution without incurring downtime. All nodes are the same type, which precludes the requirement to configure different types of nodes.
- Auto-sharding, which is discussed in more detail in the next subsection, provides automatic load balancing across the cluster with no hot spots on overloaded servers.
- The Cross Data Center Replication feature is unique to Couchbase and makes Couchbase scalable across geographies.

Auto-Sharding Cluster Technology

When a new server is added or removed from a Couchbase cluster, data is automatically redistributed to the nodes in the cluster and rebalanced without downtime in serving client requests. The process of evenly distributing data across the cluster automatically is called auto-sharding. If more RAM and I/O capacity is required, simply add a server. Data is available continuously while being balanced evenly among the cluster nodes. Client requests are routed to a server closest to the client making use of data locality. Data locality improves response time and reduces network traffic as data is being served from a server that is close to the client.

High Performance from High Throughput and Low Latency

Latency may be defined in different forms but all imply a delay: for example, the delay in receiving requested data or a delay in data transfer to another server for replication. Throughput is defined as the rate of data transfer over a network.

Couchbase is designed for the flexible data management requirements of interactive web applications providing high throughput and low latency. While most NoSQL databases provide a fast response, Couchbase's sub-millisecond latency is consistent across read and write operations and consistent across varying workloads. The latency of some of the other NoSQL databases such as MongoDB and Apache Cassandra increases as the number of ops/sec increases, but Couchbase's latency stays low even at high workloads. While most NoSQL databases provide a high throughput, Couchbase's high throughput is consistent across a mix of read and write operations. Throughput scales linearly with additional nodes. In a performance benchmark (http://www.slideshare.net/renatko/couchbase-performance-benchmarking) comparing Apache Cassandra, MongoDB, and Couchbase, Couchbase showed the lowest latencies and highest throughput. One of the reasons Viber cited for choosing Couchbase was that “Couchbase was able to provide several times more throughput using less than half the number of nodes.”

Couchbase provides built-in *memcache*-based caching technology. What is memcache? Memcache is a cache in the memory (RAM) to store temporarily (also called to cache) frequently used data. Memcache is used to optimize disk I/O; if data is made available from the RAM the disk does not have to be accessed. Memcache is also used to optimize CPU; results of CPU intensive computations are stored in the cache to avoid recomputation. What is "frequently used data" is determined by the server based on the number and frequency of requests for the data. The RAM not being used for other purposes is used as memcache, and memcache is temporary as the RAM may be reclaimed for other use if required. Couchbase Server coordinates with the disk to keep sufficient RAM to serve incoming requests with low latency for high performance. When the frequently used information is re-requested it is served from the memcache instead of fetching from the database. Memcache improves response time, which results in reduced latency and high throughput. With sub-millisecond read and write performance, Couchbase Server is capable of hundreds of thousands of ops per second per server node. Couchbase Server persists data from RAM to disk asynchronously while keeping a set of data for client access in the object-level cache in RAM. An append-only storage tier appends data contiguously to the end of a file, improving performance. Updates are first committed to RAM and subsequently to disk using per-document commit. A cache miss is defined as a direct access of a database disk when the cache does not provide the required data. Orbitz mentioned caching mechanism as the main reason for choosing Couchbase.

Cluster High Availability

Couchbase cluster stays highly available without downtime. While most NoSQL databases provide high availability, Couchbase has the following advantages over the others.

- Cross Data Center Replication, which is discussed in detail in the next section, provides high availability even in the eventuality of a whole data center failing.
- Software upgrades are done online, without shutting down the Couchbase Server.
- Hardware upgrades are done online.
- Maintenance operations such as compaction are done online.

Cross Data Center Replication

Replication of data stores multiple copies of data on different nodes in a cluster for durability and high availability. Durability implies that if one copy of the data is lost due to machine failure or some other reason such as power failure, another copy of the data is still available. High Availability implies that the database does not have downtime due to the failure of a single node in the cluster as a copy of the data from another node is fetched. In additional to replication within a cluster (intra cluster replication), Couchbase 2.0 added a feature called *Cross Data Center Replication* (XDCR) in which data is replicated across data centers to cluster/s in another data center, which could be at a geographically remote location. XDCR provides data locality in addition to the benefits discussed previously in this section. Data locality is the closeness of data to a client. If each client is able to access a node that is close to the client, data is not required to be transmitted across the network. If data is available at a data center close to a client, data is fetched from the data center instead of fetching over the network from a distant data center. Transmitting data across the network incurs delay (latency) and increased bandwidth requirement. Data locality improves response time. Cross Datacenter Replication is illustrated in Figure 1-4 in which a JSON Document A is replicated using intra cluster replication on Datacenter1, and JSON Document B is replicated using intra cluster replication on Datacenter2. JSON document is also replicated using XDCR on Datacenter2 and JSON Document B is replicated using XDCR on Datacenter1. The number of replicas may vary based on requirement.

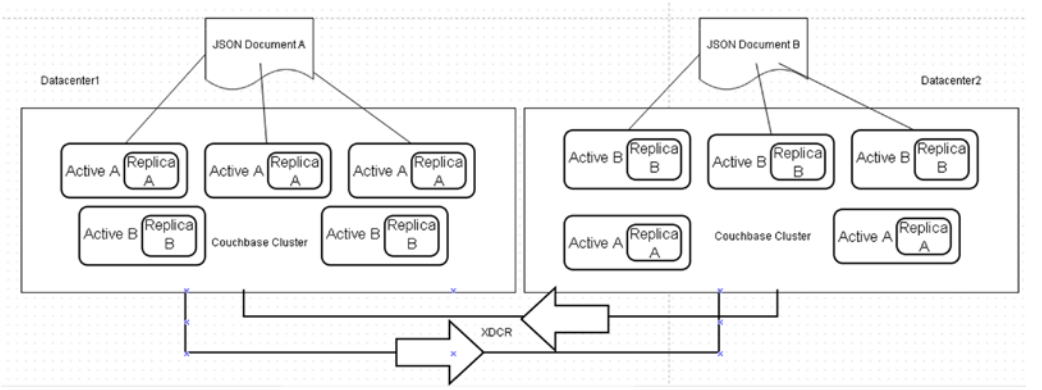

Figure 1-4. *Cross Data Center Replication (XDCR)*

XDCR replicates data unidirectionally or bidirectionally between data centers. With bidirectional replication, data may be added in either data center and read from another data center.

Data Locality

Data locality is the closeness of a Couchbase Server to its client. Cross Data Center Replication makes it feasible to replicate data across geographies. A client is served from a data center that is closest to the client, thereby reducing the network latency.

Rack Awareness

Couchbase Servers in a cluster are stored across several racks and each rack has its own power supply and switches. Failure of a single rack makes data stored on the rack susceptible to loss. To prevent loss of all copies of a data and provide high availability, Couchbase Server 2.5 Enterprise Edition introduced Rack Awareness. Using Couchbase, Rack Awareness replicas of a document are placed on nodes across different racks so that failure of a single rack does not cause all replicas of the document to be lost or become unavailable, even temporarily.

Multiple Readers and Writers

As of Couchbase Server 2.1, multiple readers and writers are supported to persist/access data to/from a disk to fully utilize the increase in disk speeds to provide high read and write efficiency. With a single thread read and write, the data in the cache is less as compared to data on the disk resulting in cache misses, which results in increased response time and increased latency. With multiple threads accessing the same disk, more data may be fetched into the cache to improve efficiency of read and write to improve the response time and reduce the latency. Multithreaded engine includes synchronization among threads to prevent multiple threads from accessing the same data concurrently.

Support for Commonly Used Object-Oriented Languages

Couchbase Server provides client APIs for commonly used languages such as Java, PHP, Ruby, and C.

Administration and Monitoring GUI

Couchbase provides administration and monitoring graphical user interface (GUI), which some of the other NoSQL databases such as MongoDB don't. Some third-party admin GUIs are available for MongoDB but a built-in integrated admin GUI is not provided.

Who Uses Couchbase Server and for What?

A wide spectrum of companies from different industries use Couchbase Server. Different companies have different reasons for choosing Couchbase Server. Reasons cited by some of the companies who chose Couchbase are listed in Table 1-3.

Table 1-3. *Reasons for Using Couchbase*

Company	Reasons
AOL for ad targeting	AOL uses Couchbase in conjunction with Hadoop to make hundreds of user profiles and statistics available for their ad targeting platform with sub-millisecond latency.
DOCOMO Innovations for mobile services	Real-time data infrastructure, mobile-to-cloud-data synchronization, elastic scalability, production-ready solution with high availability.
OMGPOP for Draw Something	Scalability without downtime or performance degradation.
Orbitz for travel services	Couchbase provides no downtime. Couchbase is used store user online sessions. Couchbase provides integrated Memcache for fast response.

(*continued*)

Table 1-3. (*continued*)

Company	Reasons
Betfair for online betting	Scalability and Replication with auto-failover are suitable for Betfair's Continuous Delivery methodology. Betfair processes more than 7 million transactions per day with each completing in less than a second. Betfair uses Couchbase to store session data across sessions and for storing user preferences for customization. Couchbase provides high performance, scalability, schema flexibility and continuous delivery.
AdAction for ad serving	Couchbase is used to store large quantities of consumer data for about 75 million users per month. Couchbase chosen because of its performance, uptime, high response time, low administrative overhead, scalability without performance loss, and rapid deployment.
Amadeus for travel services	Couchbase server was chosen because of its low (sub-millisecond) latency, elasticity to handle traffic growth, high throughput and linear scalability when adding/removing nodes.
Concur for business travel	As Concur processes more than a billion Couchbase operations per day Couchbase's low latency was one of the reasons for being chosen. Couchbase's cluster management made it feasible to add/remove nodes without downtime. Couchbase's seamless transition when adding/removing nodes requires no configuration management with all clients being updated automatically. A single solution for multiple tiers and languages was one of the main reasons for choosing Couchbase.
LinkedIn for professional social networking	With hundreds of millions of users LinkedIn chose Couchbase for its performance and scalability that can be used for logging, monitoring and analyzing the metrics of user activity. High availability caching was one of the main reasons for choosing Couchbase.
Nami Media for enterprise class advertising solutions	Couchbase was chosen for its fault-tolerance, data persistence and high availability. Linear scalability with no downtime and Couchbase's monitoring of the cluster to provide RAM and disk persistence statistics were some of the other reasons.

Couchbase users include AOL, Orbitz, Cisco, LinkedIn, and Concur. Table 1-4 lists additional Internet companies and Enterprises who use Couchbase Server.

Table 1-4. *Some Companies Using Couchbase*

Internet Companies	Enterprises
AOL	LG
Orbitz	ADP
LinkedIn	Cisco
adscale	NTT Docomo
Ubisoft	Vodafone
Tapjoy	Skechers
Dotomi	NCR Corporation
Playdom	Comcast
Concur	ITT
Sabre	Experian

Couchbase Server is used for a wide variety of applications ranging from advertising to VoIP services. Couchbase Server and NoSQL, in general, are used for applications storing and processing big data. Examples of types of applications using NoSQL are discussed in Table 1-5.

Table 1-5. *Types of Applications Using Couchbase*

Application	Example
User profile management distributed globally	The user profile of millions of LinkedIn, Tunewiki, and AOL users is stored in Couchbase NoSQL database. The semi-transient device data of millions of musiXmatch users is stored in Couchbase Server.
Session store management	The user sessions of millions of clients who log on to Orbitz, Concur, Sabre, and musiXmatch are stored in Couchbase database.
Content and metadata store management	Some of the challenges in content and metadata store management are: Content and metadata are unstructured. Scalability to support millions of concurrent users. High-performance interactive, customized applications. Search across the full dataset. Couchbase is suitable for the following reasons: • Elastisearch provides real-time, integrated, distributed, full-text search. • Flexible data model to provide a wide variety of data. Scalability for fluctuations in workload. High performance with low latency and high throughput. No downtime.

(*continued*)