大数据时代的数据存储—MongoDB

导语：在过去的很长一段时间中，关系型数据库一直是最主流的数据库解决方案，她运用真实世界中事物与事物的关系来解释数据库中抽象的数据。然而，在信息技术、用户基数、数据量等爆炸式发展的今天，大数据已经成为了继云计算，物联网后新的技术革命，关系型数据库在处理大数据量时已经开始吃力，开发者只能通过不断地优化数据库来解决数据量的问题，但优化毕竟不是一个长期方案，所以人们提出了一种新的数据库解决方案来迎接大数据时代的到来——NOSQL。

一、RDBMS面对数据（以MySQL为例）访问的压力，通常采取的解决方案步骤

1、主从复制，实现读写分离或分布读；
2、读请求比较多，可添加缓存服务器，如Memcached/Redis，以提升读性能；但手动维护数据的一致性往往也是一个不小的一个挑战；
3、写请求较多的场景，可简单进行向上扩展，使用性能更强的服务器以应付更多的写请求；同时，为了保证从服务器跟得上主服务器的更新速度，可能需要从服务器使用与主服务器相同的配置；此法性价比不高；
4、数据访问压力进一步增大时，联结查询性能会急剧下降；此时就得进行“反模式”化设计，将表根据业务需求进行合并，以增大数据冗余来换取系统性能；
5、停用存储过程、存储函数或触发器等代码，将对应的功能在应用程序中完成；
6、删除表的各辅助索引，改写查询使其仅使用主键索引；
7、数据库切分，此法复杂度较大，维护成本较高；且数据规模再次提升时重新切分的成本高昂，二次扩展能力受限；

二、RDBMS与NoSQL

实际使用中，只要架构得当，关系型数据库完全能够服务于各种级别的数据存储应用，比如国外知名企业Facebook、Google等各自有着运转良好的MySQL服务器集群，服务于不同层次不同领域的数据存储场景。但此等规模的应用需要强大的技术实力突破各式各样的应用限制，这也会带来居高不下的维护成本（高端人才和高配物理服务器），尽管如此关系型数据库一些“天生性”的限制依然会成为应用中的梦魇和沼泽之

地。于是，近几年来，一些被归类为NoSQL的新项目或框架在多个组织或企业中爆发式涌现。这些新项目或框架很少提供类似SQL语言一样的查询语言，而是提供了一种简化的、类API的数据访问接口。但RDBMS与NoSQL真正的不同之处在于低层，即存储级别，因为NoSQL通常不支持事务或辅助索引的功能等。

另一方面，NoSQL的著名项目中彼此间有许多功能是重叠的，甚至有不少特性与传统的关系型数据库的功能也存在相同之处，因此NoSQL算不上革命性的技术，尽管从工程师的眼下其绝对是革命性的。于是，现实中，memcached也被划归了NoSQL阵营，似乎不属于RDBMS的存储管理类程序都自然而然的属于NoSQL，NoSQL也因而成为了非RDBMS系统的“海纳百川”之地。然而，“有容乃大”就难免“鱼龙混杂”，为了便于理解，这里从多个维度来对NoSQL的主流技术进行简单的归类，以便对此能有个概括性的认识，并能够在实际应用场景中有个可以参照的选择标准。

1、数据模型
数据模型指数据的存储方式，其有好几个流派，如关系、键值、列式、文档及图像等。在它们的各自实现中，关系型数据库有Oracle、SQL Server、MySQL、PostgreSQL等，键值数据库有memcached、membase、Riak、Redis等，列式数据库有HBase、Cassandra、Hypertable等，文档数据库有MongoDB、CouchDB等，图像数据库有Neo4J等。在选用某特定的NoSQL产品时，应该事先评估应用程序是如何访问数据的，以及数据的Schema是否经常演进等。

2、存储模型
指数据存储是基于内存存储还是持久存储。

3、一致性模型
存储系统在何种级别实现数据一致性，严格一致性还是结果一致性。一致性的等级可能会对数据访问延迟带来巨大影响。

4、物理模型
在物理模型上可归类分布式存储及单机存储。对分布式存储而言，其扩展能力及易扩展性如何也是一个重要的衡量指标。

5、读/写性能
对于工作在不同应用场景中的应用程序而言，其读/写需求有着显著不同。而不同的NoSQL产品也有着不同的适用性。

6、辅助索引
辅助索引有助于实现在非主键字段上完成排序、查询操作等；有的NoSQL产品不提供此类功能。

7、故障处理
不同的应用场景其故障恢复的时间容忍度不同，而不同的NoSQL产品也故障恢复能力方面也有着不同的表现。

8、数据压缩
当存储TB级别的数据时，尤其是存储中包含大量文件（文本文件、图片、音频、视频）数据时，数据压缩可以大量节约存储空间。故是否支持高效、高性能压缩就需要慎重考量。

9、负载均衡
分布式存储将用户的读/写请求分布于多个节点同时进行能够极大提升系统性能。

10、锁、等待和死锁
RDBMS的事务处理过程分为两个阶段（prepare-commit/rollback），多用户并发访问的场景中，这将显著增加用户在访问资源时的等待时间，甚至会导致死锁。而一般来说，NoSQL系列产品在这方面就会好很多。

三、数据一致性模型

概括来讲，数据一致性是指在应用程序访问时，数据的有效性(validity)、可用性(usability)、精确性(accuracy)及完整性(integrity)方面的表现，其用于保证在用户自身事务或其他用户的事务执行过程中，每个用户看到的数据是一致的。在各种场景中都有可能产生数据一致性问题，但提到的较多的通常有应用程序一致性、事务一致性和时间点一致性等。

在数据库上，每个操作都可能促使数据库从一种状态转换为另一种状态，但这种转换的实现方式或过程是非特定的，因此其有着多种不同的模型。不过，无论是基于哪种实现，其最终结果要么是转换为的状态，要么恢复回原有的状态以保证数据的一致性。根据数据库在保证数据一致性实现的严格程度来分，大致有如下几类：

严格一致性(strict)—数据的改变是原子性的并且会立即生效；这是最高级别的一致性实现；
顺序一致性(Sequential)—每个客户端以他们提交应用的次序看到数据的改变；

因果一致性(Causal)—因果关联的所有的改变，在所有的客户端以同样的次序获取；
结果一致性(Eventual)—当一段时间内没有更新发生时，所有更新会通过系统进行传播，最终所有的副本都是一致的；也即当事务完成时，必须使所有数据都具有一致的状态；
弱一致性(Weak)—不保证所有的更新都能通告给所有客户端，也不保证所有客户端都能以同样的次序获取数据更新；

其中，结果一致性还可以进一步细分为多种不同的子类别，如我们熟知的CAP定理，它指出对于一个分布式计算系统来说，不可能同时满足以下三点:

一致性(Consistency) ：所有节点在同一时间具有相同的数据
可用性(Availability) ：保证每个请求不管成功或者失败都有响应
分区/隔容忍(Partition tolerance) ：系统中任意信息的丢失或失败不会影响系统的继续运作

CAP理论的核心是：一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求，最多只能同时较好的满足两个。因此，根据 CAP 原理将 NoSQL 数据库分成了满足 CA 原则、满足 CP 原则和满足 AP 原则三 大类：

CA（单点集群）：满足一致性，可用性的系统，通常在可扩展性上不太强大。
CP（满足一致性）：通常性能不是特别高。
AP（满足可用性）：分区容忍性的系统，通常可能对一致性要求低一些。

即分布式系统仅能同时实现一致性、可用性和分区容错性三种属性中的两种。

四、爆炸式发展的NoSQL技术

NoSQL目前还非常年轻，但她拥有的众多优秀的特性已经让众多企业和开发者开始接受并大规模使用，让我们来看一下来自于美国数据库知识网站DB-engines最近一个月的数据库排名情况。

299 systems in ranking, March 2016

Rank Mar 2016	Rank Feb 2016	Rank Mar 2015	DBMS	Database Model	Score Mar 2016	Score Feb 2016	Score Mar 2015
1.	1.	1.	Oracle	Relational DBMS	1472.01	-4.13	+2.93
2.	2.	2.	MySQL	Relational DBMS	1347.71	+26.59	+86.62
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1136.49	-13.73	-28.31
4.	4.	4.	MongoDB	Document store	305.33	-0.27	+30.32
5.	5.	5.	PostgreSQL	Relational DBMS	299.62	+10.97	+35.19
6.	6.	6.	DB2	Relational DBMS	187.94	-6.55	-10.91
7.	7.	7.	Microsoft Access	Relational DBMS	135.03	+1.95	-6.66
8.	8.	8.	Cassandra	Wide column store	130.33	-1.43	+23.02
9.	↑ 10.	↑ 10.	Redis	Key-value store	106.22	+4.14	+9.17
10.	↓ 9.	↓ 9.	SQLite	Relational DBMS	105.77	-1.01	+4.06
11.	↑ 12.	↑ 15.	Elasticsearch	Search engine	80.17	+2.33	+21.24
12.	↓ 11.	↓ 11.	SAP Adaptive Server	Relational DBMS	76.64	-3.39	-8.72
13.	13.	13.	Teradata	Relational DBMS	74.07	+0.69	+1.29
14.	14.	↓ 12.	Solr	Search engine	69.37	-2.91	-12.52
15.	↑ 16.	↓ 14.	HBase	Wide column store	52.41	+0.39	-8.32
16.	↓ 15.	↑ 17.	Hive	Relational DBMS	50.51	-2.26	+11.18
17.	17.	↓ 16.	FileMaker	Relational DBMS	47.93	+0.90	-4.41
18.	18.	↑ 19.	Splunk	Search engine	43.73	+0.90	+8.01
19.	19.	↑ 21.	SAP HANA	Relational DBMS	39.99	+1.91	+7.82
20.	↑ 21.	↑ 23.	Neo4j	Graph DBMS	32.36	+0.07	+4.73
21.	↓ 20.	↓ 18.	Informix	Relational DBMS	31.87	-1.15	-5.95
22.	↑ 23.	↑ 25.	MariaDB	Relational DBMS	29.88	+1.11	+7.79
23.	↓ 22.	↓ 20.	Memcached	Key-value store	29.24	+0.31	-6.27
24.	24.	24.	Couchbase	Document store	25.80	+0.41	+2.62
25.	25.	↓ 22.	CouchDB	Document store	23.38	-0.16	-4.53

根据榜单发现，处于榜首的数据库仍然是Oracle，其次就是MySQL和Microsoft SQL Server 分别位居二三名。可见，它们三个数据库以绝对的优势霸占着市场中最多的用户。同时也不难发现MongoDB从2015年就一直雄踞NoSQL系列产品“榜首”。

NoSQL的最大几个优点：高可扩展性、分布式计算、低成本、架构的灵活性，半结构化数据、没有复杂的关系等。而其中BASE是NoSQL数据库通常对可用性及一致性的弱要求原则，即：Basically Availble（基本可用）、Soft-state（软状态/柔性事务）、Eventual Consistency（最终一致性）。

ACID VS BASE

原子性(Atomicity)	基本可用(Basically Available)
一致性(Consistency)	软状态/柔性事务(Soft state)
隔离性(Isolation)	最终一致性 (Eventual consistency)
持久性 (Durable)	

SQL术语 VS MongoDB术语

SQL术语	MongoDB术语	概念
database	database	数据库
table	collection	数据库表/集合
row	document	数据记录行/文档
column	field	数据字段/域
index	index	索引
primary key	primary key	主键,MongoDB自动将_id字段设置为主键
join	embedding&linking	表连接
partition	shard	分区、分片
partition key	shard key	分区键、分片键

NoSQL 数据库分类

类型	部分代表	简介
列存储	Hbase	按列存储数据的。最大的特点是方便存储结构化和半结构化数据，方便做数据压缩，对针对某一列或者某几列的查询有非常大的IO优势。
文档存储	MongoDB	文档存储一般用类似json的格式存储，存储的内容是文档型的。这样也就有有机会对某些字段建立索引，实现关系数据库的某些功能。
key-value存储	Memcached、 Redis	可以通过key快速查询到其value。一般来说，存储不管value的格式，照单全收。（Redis包含了其他更丰富的功能）
图存储	FlockDB、Neo4j	图形关系的最佳存储。使用传统关系数据库来解决的话性能低下，而且设计使用不方便。
对象存储	Versant	通过类似面向对象语言的语法操作数据库，通过对象的方式存取数据。
xml数据库	BaseX	高效的存储XML数据，并支持XML的内部查询语法，比如XQuery,Xpath。

五、主流NoSQL对比介绍

1. CouchDB

所用语言： Erlang
特点：DB一致性，易于使用
使用许可： Apache
协议： HTTP/REST
双向数据复制；
持续进行或临时处理；
处理时带冲突检查；
采用master-master复制
MVCC – 写操作不阻塞读操作；
可保存文件之前的版本；
Crash-only（可靠的）设计；
需要不时地进行数据压缩；

视图：嵌入式 映射/减少；
格式化视图：列表显示；
支持进行服务器端文档验证；
支持认证；
根据变化实时更新；
支持附件处理；

最佳应用场景：适用于数据变化较少，执行预定义查询，进行数据统计的应用程序。适用于需要提供数据版本支持的应用程序。

例如： CRM、CMS系统。 master-master复制对于多站点部署是非常有用的。

2. Redis

所用语言：C/C++
特点：运行异常快
使用许可： BSD
协议：类 Telnet
有硬盘存储支持的内存数据库；
Master-slave复制；
虽然采用简单数据或以键值索引的哈希表，但也支持复杂操作，例如ZREVRANGEBYSCORE；
INCR & CO （适合计算极限值或统计数据）；
支持 sets（同时也支持 union/diff/inter）；
支持列表（同时也支持队列、阻塞式 pop操作）；
支持哈希表（带有多个域的对象）；
支持排序 sets（高得分表，适用于范围查询）；
Redis支持事务；
支持将数据设置成过期数据（类似快速缓冲区设计）；
Pub/Sub允许用户实现消息机制；

最佳应用场景：适用于数据变化快且数据库大小可遇见（适合内存容量）的应用程序。

例如：股票价格、数据分析、实时数据搜集、实时通讯。

3. MongoDB

所用语言：C++
特点：保留了SQL一些友好的特性（查询，索引）。
使用许可： AGPL（发起者： Apache）
协议： Custom, binary（ BSON）

Schema free；
二级索引；
全文搜索；
聚合框架、MapReduce；
地理位置索引；
多变易用复制集：高可用，自动故障切换，并且能够实现多数据中心支持，自动容灾、滚动维护无下线；
读写性能高：在关系型数据库中，我们经常会进行join、子查询等关联性需求，这时候往往会带来较多的随机IO，而在MongoDB中，我们可以通过合理的数据模型设计来将很多的关联需求通过内嵌、反范式的方式实现，减少随机IO；
支持动态查询：查询指令也使用JSON形式的标记，可轻易查询文档中内嵌的对象及数组；
地理分布；
Master/slave复制（支持自动错误恢复）；
内建分片机制；
支持 javascript表达式查询；
可在服务器端执行任意的 javascript函数；
update-in-place支持比CouchDB更好；
支持多种存储引擎（MMAP、WT、Memory—开发中、事务—开发中）；
支持弱事务；
采用 GridFS存储大数据或元数据；

最佳应用场景：适用于需要动态查询支持；schema变化频繁；低延迟、持久、高吞吐的读写性能；实时数据分析；日志收集；需要使用索引而不是 map/reduce功能；需要对大数据库有性能要求；需要使用CouchDB但因为数据改变太频繁而占满内存的应用程序。

4. Riak

所用语言：Erlang和C，以及一些Javascript
特点：具备容错能力
使用许可： Apache
协议： HTTP/REST或者 custom binary
可调节的分发及复制(N, R, W)；
用 JavaScript or Erlang在操作前或操作后进行验证和安全支持；
使用JavaScript或Erlang进行 Map/reduce；
连接及连接遍历：可作为图形数据库使用；
索引：输入元数据进行搜索（1.0版本即将支持）；
大数据对象支持（ Luwak）；
提供“开源”和“企业”两个版本；
全文本搜索，索引，通过 Riak搜索服务器查询（ beta版）；
支持Masterless多站点复制及商业许可的 SNMP监控；

最佳应用场景：适用于想使用类似 Cassandra（类似Dynamo）数据库但无法处理 bloat及复杂性的情况。适用于你打算做多站点复制，但又需要对单个站点的扩展性，可用性及出错处理有要求的情况。

例如：销售数据搜集，工厂控制系统；对宕机时间有严格要求；可以作为易于更新的 web服务器使用。

5. Membase

所用语言： Erlang和C
特点：兼容 Memcache，但同时兼具持久化和支持集群
使用许可： Apache 2.0
协议：分布式缓存及扩展
非常快速（200k+/秒），通过键值索引数据；
可持久化存储到硬盘；
所有节点都是唯一的（ master-master复制）；
在内存中同样支持类似分布式缓存的缓存单元；
写数据时通过去除重复数据来减少 IO；
提供非常好的集群管理 web界面；
更新软件时软无需停止数据库服务；
支持连接池和多路复用的连接代理；

最佳应用场景：适用于需要低延迟数据访问，高并发支持以及高可用性的应用程序

例如：低延迟数据访问比如以广告为目标的应用，高并发的 web 应用比如网络游戏（例如 Zynga）

6. Neo4j

所用语言： Java
特点：基于关系的图形数据库
使用许可： GPL，其中一些特性使用 AGPL/商业许可
协议： HTTP/REST（或嵌入在 Java中）
可独立使用或嵌入到 Java应用程序；
图形的节点和边都可以带有元数据；
很好的自带web管理功能；
使用多种算法支持路径搜索；
使用键值和关系进行索引；
为读操作进行优化；
支持事务（用 Java api）；
使用 Gremlin图形遍历语言；
支持 Groovy脚本；
支持在线备份，高级监控及高可靠性支持使用 AGPL/商业许可；

最佳应用场景：适用于图形一类数据。这是 Neo4j与其他nosql数据库的最显著区别

例如：社会关系，公共交通网络，地图及网络拓谱

7. Cassandra

所用语言： Java
特点：对大型表格和 Dynamo支持得最好
使用许可： Apache
协议： Custom, binary (节约型)

可调节的分发及复制(N, R, W)；
支持以某个范围的键值通过列查询；
类似大表格的功能：列，某个特性的列集合；
写操作比读操作更快；
基于 Apache分布式平台尽可能地 Map/reduce；

最佳应用场景：当使用写操作多过读操作（记录日志）如果每个系统组建都必须用 Java编写（没有人因为选用 Apache的软件被解雇）

例如：银行业，金融业（虽然对于金融交易不是必须的，但这些产业对数据库的要求会比它们更大）写比读更快，所以一个自然的特性就是实时数据分析

8. HBase

（配合 ghshephard使用）

所用语言： Java
特点：支持数十亿行X上百万列
使用许可： Apache
协议：HTTP/REST （支持 Thrift，见编注4）
在 BigTable之后建模；
采用分布式架构 Map/reduce；
对实时查询进行优化；
高性能 Thrift网关；
通过在server端扫描及过滤实现对查询操作预判；
支持 XML, Protobuf, 和binary的HTTP；
Cascading, hive, and pig source and sink modules；
基于 Jruby（ JIRB）的shell；
对配置改变和较小的升级都会重新回滚；
不会出现单点故障；
堪比MySQL的随机访问性能；

六、NoSQL对传统数据库设计思维的影响

1.预设计模式与动态模式

传统数据库设计思维中，项目的设计阶段需要对数据库表中的字段名称、字段类型、进行规定，如果尝试插入不符合设计的数据，数据库不会接受这条数据以保证数据的完整性。

NoSQL采用的是对集合中的文档进行动态追加，在创建集合之初不会对数据类型进行限定，任何文档都可以追加到任何集合中去，MongoDB中文档的格式类似于我们常见的JSON（保留基本json的键/值对特性之上，还添加了一些，如null、Boolean、Timestamp、Object、Date、Object ID、Binary data、Code、Regular expression、Max/Min），这样一来我们便可以“随时随地”增加、修改数据模式。

2.范式化与反范式化

范式化会将数据分散到不同的表中，利用关系模型进行关联，由此带来的优点是，在后期进行修改时，不会影响到与其关联的数据，仅对自身修改即可完成。

反范式化是针对范式化提出的相反理念，反范式化会将当前文档的数据集中存放在本表中，而不会采用拆分的方式进行存储。

范式化和反范式化之间不存在优劣的问题，范式化的好处是可以在我们写入、修改、删除时的提供更高性能，而反范式化可以提高我们在查询时的性能。当然NoSQL中是不存在关联查询的，以此提高查询性能，但我们依旧可以以在表中存储关联表ID的方式进行范式化。同时NoSQL的理念中反范式化的地位是大于范式化的。

七、为什么选择MongoDB

1.灵活动态文档模型

MongoDB采用的是NoSQL的设计方式，可以更加灵活的操作数据。在进行传统的RDBMS中你一定遇到过几十行甚至上百行的复杂SQL语句，传统的RDBMS的SQL语句中包含着大量关联，子查询等语句，在增加复杂性的同时还让性能调优变得更加困难。MongoDB的面向文档设计中采用更为灵活的Json文档来作为数据模型用来取代RDBMS中的行，面向文档的设计让开发人员获取数据的方式更加灵活，甚至于开发人员仅用一条语句即可查询复杂的嵌套关系，让开发人员不必为了获取数据而绞尽脑汁。其次，随着产品的迭代发展，当初设计的数据表结构、数据表关

系、数据字段等需要多次被修改，故不经感叹”有了Json在手，说走就走 “。

2.扩展性

现在互联网的数据量已经从过去的GB变为了现在的TB、PB级别，单一的数据库显然已经无法承受，扩展性成为重要的话题，然而现在的开发人员常常在选择扩展方式的时候犯了难，到底是选择横向扩展还是纵向扩展。

横向扩展（**SCALE OUT**）是以增加分区的方式将数据库拆分成不同的区块来分布到不同的机器中来，这样的优势是扩展成本低但管理困难。

纵向扩展（**SCALE UP**） 纵向扩展与横向扩展不同的是她会将原本的服务器进行升级，让其拥有更强大的计算能力、存储能力。这样的优势是易于管理无需考虑扩展带来的众多问题，但缺点也显而易见，那就是成本高。一台大型机的价格往往非常昂贵，现实往往比这更残酷，因为当数据达到某个极限时，可能就找不到如此强悍的机器了。

而MongoDB选择的是更为经济的横向扩展，她可以很容易的将数据拆分至不同的服务器中。而且在获取数据时开发者也无需考虑多服务器带来的问题，MongoDB可以将开发者的请求自动路由到正确的服务器，让开发者脱离横向扩展带来的弊病，更专注于程序的开发上。

3.性能

在大数据时代中，大数据量的处理已经成了考量一个数据库最重要的原因之一。而MongoDB的一个主要目标就是尽可能的让数据库保持卓越的性能，这很大程度地决定了MongoDB的设计。在一个以传统机械硬盘为主导的年代，硬盘很可能会成为性能的短板，而MongoDB选择了最大程度而利用内存资源用作缓存来换取卓越的性能，并且会自动选择速度最快的索引来进行查询。MongoDB尽可能精简数据库，将尽可能多的操作交给客户端，这种方式也是MongoDB能够保持卓越性能的原因之一（具体性能各项指标请参考MongoDB测试白皮书）。

4.高可用

我们由足够理由相信，在企业中有99%的产品都要提供99.99%的高可用，保障服务7*24小时在线。而大多数数据库产品中要实现此功能，并不是一件容易的事情。MongoDB在设计之初就完美地自实现高可用，甚至是零运维（当然了，提供几个简单的配置项还是要做滴）。