一文综述非关系型数据库（NoSQL）

韩润超

## 前言



NoSQL(NoSQL = Not Only SQL )，意即"不仅仅是SQL"。

在现代的计算系统上每天网络上都会产生庞大的数据量。这些数据有很大一部分是由关系数据库管理系统（RDMBSs）来处理，其严谨成熟的数学理论基础使得数据建模和应用程序编程更加简单。

但随着信息化的浪潮和互联网的兴起，传统的RDBMS在一些业务上开始出现问题。首先，对数据库存储的容量要求越来越高，单机无法满足需求，很多时候需要用集群来解决问题，而RDBMS由于要支持join，union等操作，一般不支持分布式集群。其次，在大数据大行其道的今天，很多的数据都“频繁读和增加，不频繁修改”，而RDBMS对所有操作一视同仁，这就带来了优化的空间。另外，互联网时代业务的不确定性导致数据库的存储模式也需要频繁变更，不自由的存储模式增大了运维的复杂性和扩展的难度。

NoSQL 是一项全新的数据库革命性运动，早期就有人提出，发展至2009年趋势越发高涨。这类数据库主要有这些特点：非关系型的、分布式的、开源的、水平可扩展的。原始的目的是为了大规模web 应用。NoSQL 的拥护者们提倡运用非关系型的数据存储，通常的应用如：模式自由、支持简易复制、简单的API、最终的一致性（非ACID）、大容量数据等。

笔者是MongoDB用户，也使用过Redis。关系型数据库使用过MySQL与Oracle，对两者的区别有一定的体会。Mongo和Redis的操作都非常简单，速度很快，很多用SQL需要很多条语句的操作在NoSQL数据库中都是2句以内完成。另外NoSQL配置cluster也很容易，且可以随时更改partition和replica的数量，Mongo的新版本还内置了MapReduce操作，使其有了做大数据分析的能力。

**本文涉及以下内容：**

[一． NoSQL理论基础 3](#_Toc477687609)

[1. 关系型数据库理论 - ACID 3](#_Toc477687610)

[2. 分布式系统理论 4](#_Toc477687611)

[3. 分布式存储算法 7](#_Toc477687612)

[4. NoSQL的优点/缺点 8](#_Toc477687613)

[二． NoSQL与SQL的对比 10](#_Toc477687614)

[三． NoSQL数据库的分类 11](#_Toc477687615)

[1. 键值(Key-Value)存储数据库 11](#_Toc477687616)

[2. 列存储数据库 11](#_Toc477687617)

[3. 文档型数据库 12](#_Toc477687618)

[4. 图形(Graph)数据库 12](#_Toc477687619)

[四． 主流NoSQL数据库介绍及其适用场景 13](#_Toc477687620)

[1. Redis 13](#_Toc477687621)

[2. MongoDB 14](#_Toc477687622)

[3. Neo4j 14](#_Toc477687623)

[4. Cassandra 15](#_Toc477687624)

[5. HBase 15](#_Toc477687625)

[6. CouchDB 15](#_Toc477687626)

[五． NoSQL优秀应用实例 16](#_Toc477687627)

[1. 新浪微博 - Redis 16](#_Toc477687628)

[2. 淘宝数据平台 – Oceanbase, Tair（均为自研） 16](#_Toc477687629)

[3. 优酷运营数据分析 – HBase, MongoDB, Redis 16](#_Toc477687630)

[4. 豆瓣社区 – BeansDB（自研KV数据库） 16](#_Toc477687631)

[六． 学习资料 17](#_Toc477687632)

[1. 书籍 17](#_Toc477687633)

[2. 视频 18](#_Toc477687634)

[3. 边用边Google 19](#_Toc477687635)

[七．进一步学习 19](#_Toc477687636)

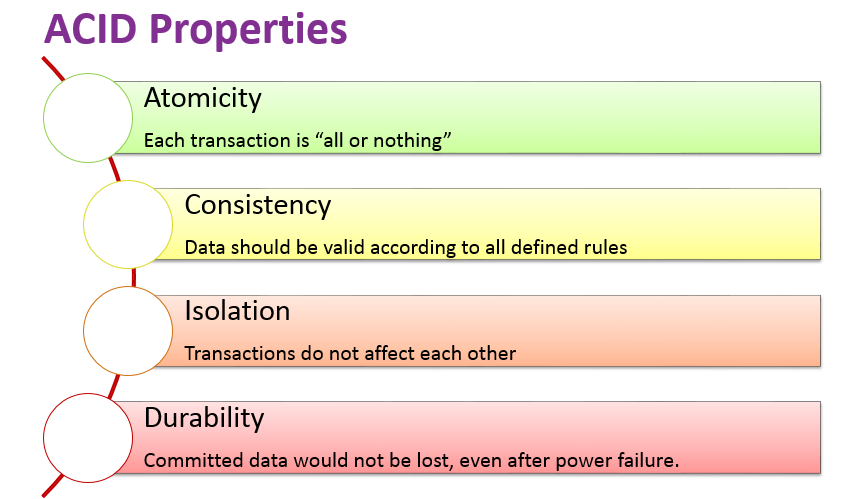
[1. 分布式算法 19](#_Toc477687637)

[2. NoSQL数据库的研究及底层实现 19](#_Toc477687638)

[3. NoSQL应用案例 19](#_Toc477687639)

## NoSQL理论基础

### 关系型数据库理论 - ACID



ACID，是指数据库管理系统（DBMS）在写入或更新资料的过程中，为保证事务（transaction）是正确可靠的，所必须具备的四个特性：原子性（atomicity，或称不可分割性）、一致性（consistency）、隔离性（isolation，又称独立性）、持久性（durability）。

1. **A – Atomicity – 原子性**

一个事务（transaction）中的所有操作，要么全部完成，要么全部不完成，不会结束在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。

1. **C – Consistency – 一致性**

在事务开始之前和事务结束以后，数据库的完整性没有被破坏。这表示写入的资料必须完全符合所有的预设规则，这包含资料的精确度、串联性以及后续数据库可以自发性地完成预定的工作。

1. **I – Isolation – 隔离性**

数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。事务隔离分为不同级别，包括读未提交（Read uncommitted）、读提交（read committed）、可重复读（repeatable read）和串行化（Serializable）。

1. **D – Durability – 持久性**

事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失。

关系型数据库严格遵循ACID理论。但当数据库要开始满足横向扩展、高可用、模式自由等需求时，需要对ACID理论进行取舍，不能严格遵循ACID。以CAP理论和BASE理论为基础的NoSQL数据库开始出现。

### 分布式系统理论

#### 2.1 分布式系统介绍

分布式系统的核心理念是让多台服务器协同工作，完成单台服务器无法处理的任务，尤其是高并发或者大数据量的任务。分布式是NoSQL数据库的必要条件。

分布式系统由独立的服务器通过网络松散耦合组成的。每个服务器都是一台独立的PC机，服务器之间通过内部网络连接，内部网络速度一般比较快。因为分布式集群里的服务器是通过内部网络松散耦合，各节点之间的通讯有一定的网络开销，因此分布式系统在设计上尽可能减少节点间通讯。此外，因为网络传输瓶颈，单个节点的性能高低对分布式系统整体性能影响不大。比如，对分布式应用来说，采用不同编程语言开发带来的单个应用服务的性能差异，跟网络开销比起来都可以忽略不计。

因此，分布式系统每个节点一般不采用高性能的服务器，而是性能相对一般的普通PC服务器。提升分布式系统的整体性能是要通过横向扩展（增加更多的服务器），而不是纵向扩展（提升每个节点的服务器性能）。

分布式系统最大的特点是可扩展性，它能够适应需求变化而扩展。企业级应用需求经常随时间而不断变化，这也对企业级应用平台提出了很高的要求。企业级应用平台必须要能适应需求的变化，即具有可扩展性。比如移动互联网2C应用，随着互联网企业的业务规模不断增大，业务变得越来越复杂，并发用户请求越来越多，要处理的数据也越来越多，这个时候企业级应用平台必须能够适应这些变化，支持高并发访问和海量数据处理。分布式系统有良好的可扩展性，可以通过增加服务器数量来增强分布式系统整体的处理能力，以应对企业的业务增长带来的计算需求。

#### 2.2 分布式存储的问题 – CAP理论

如果我们期待实现一套严格满足ACID的分布式事务，很可能的情况就是系统的可用性和严格一致性出现冲突。在可用性和一致性之间永远无法存在一个两全其美的方案。由于NoSQL的基本需求就是支持分布式存储，严格一致性与可用性需要互相取舍，由此延伸出了CAP理论来定义分布式存储遇到的问题。

CAP理论告诉我们：一个分布式系统不可能同时满足一致性(C:Consistency)、可用性(A:Availability)、分区容错性(P:Partition tolerance)这三个基本需求，并且最多只能满足其中的两项。

对于一个分布式系统来说，分区容错是基本需求，否则不能称为分布式系统。因此架构师需要在C和A之间寻求平衡。



1. **C – Consistency – 一致性（与ACID的C完全不同）**

一致性指“all nodes see the same data at the same time”，即更新操作成功并返回客户端完成后，所有节点在同一时间的数据完全一致。

对于一致性，可以分为从客户端和服务端两个不同的视角。

从客户端来看，一致性主要指的是多并发访问时更新过的数据如何获取的问题。

从服务端来看，则是更新如何复制分布到整个系统，以保证数据最终一致。一致性是因为有并发读写才有的问题，因此在理解一致性的问题时，一定要注意结合考虑并发读写的场景。

从客户端角度，多进程并发访问时，更新过的数据在不同进程如何获取的不同策略，决定了不同的一致性。对于关系型数据库，要求更新过的数据能被后续的访问都能看到，这是强一致性。如果能容忍后续的部分或者全部访问不到，则是弱一致性。如果经过一段时间后要求能访问到更新后的数据，则是最终一致性。

1. **A – Availability – 可用性**

可用性指“Reads and writes always succeed”，即服务一直可用，而且是正常响应时间。

对于一个可用性的分布式系统，每一个非故障的节点必须对每一个请求作出响应。也就是，该系统使用的任何算法必须最终终止。当同时要求分区容忍性时，这是一个很强的定义：即使是严重的网络错误，每个请求必须完成。

好的可用性主要是指系统能够很好的为用户服务，不出现用户操作失败或者访问超时等用户体验不好的情况。可用性通常情况下可用性和分布式数据冗余，负载均衡等有着很大的关联。

1. **P – Partition tolerance – 分区容错性**

分区容错性指“the system continues to operate despite arbitrary message loss or failure of part of the system”，即分布式系统在遇到某节点或网络分区故障的时候，仍然能够对外提供满足一致性和可用性的服务。

分区容错性和扩展性紧密相关。在分布式应用中，可能因为一些分布式的原因导致系统无法正常运转。好的分区容错性要求能够使应用虽然是一个分布式系统，而看上去却好像是在一个可以运转正常的整体。比如现在的分布式系统中有某一个或者几个机器宕掉了，其他剩下的机器还能够正常运转满足系统需求，或者是机器之间有网络异常，将分布式系统分隔未独立的几个部分，各个部分还能维持分布式系统的运作，这样就具有好的分区容错性。

1. **CA without P：**

如果不要求P（不允许分区），则C（强一致性）和A（可用性）是可以保证的。但其实分区不是你想不想的问题，而是始终会存在，因此CA的系统更多的是允许分区后各子系统依然保持CA。

1. **CP without A**

如果不要求A（可用），相当于每个请求都需要在Server之间强一致，而P（分区）会导致同步时间无限延长，如此CP也是可以保证的。很多传统的数据库分布式事务都属于这种模式。

1. **AP without C**

要高可用并允许分区，则需放弃一致性。一旦分区发生，节点之间可能会失去联系，为了高可用，每个节点只能用本地数据提供服务，而这样会导致全局数据的不一致性。现在众多的NoSQL都属于此类。

CAP理论定义了分布式存储的根本问题，但并没有指出一致性和可用性之间到底应该如何权衡。于是出现了BASE理论，给出了权衡A与C的一种可行方案。

#### 2.3 权衡一致性与可用性 - BASE理论

Base = Basically Available + Soft state + Eventually consistent 基本可用性+软状态+最终一致性，由eBay架构师Dan Pritchett提出。Base是对CAP中一致性A和可用性C权衡的结果，源于提出者自己在大规模分布式系统上实践的总结。核心思想是无法做到强一致性，但每个应用都可以根据自身的特点，采用适当方式达到最终一致性。

1. **BA - Basically Available - 基本可用**

基本可用。这里是指分布式系统在出现故障的时候，允许损失部分可用性，即保证核心功能或者当前最重要功能可用。对于用户来说，他们当前最关注的功能或者最常用的功能的可用性将会获得保证，但是其他功能会被削弱。

1. **S – Soft State - 软状态**

允许系统数据存在中间状态，但不会影响到系统的整体可用性，即允许系统在不同节点的数据副本之间进行数据同步时存在延时。

1. **E - Eventually Consistent - 最终一致性**

要求系统数据副本最终能够一致，而不需要实时保证数据副本一致。最终一致性是弱一致性的一种特殊情况。最终一致性有5个变种：

1. 因果一致性
2. 读己之所写(因果一致性特例)
3. 会话一致性
4. 单调读一致性
5. 单调写一致性

### 分布式存储算法

#### 一致性算法 – Paxos

Paxos 算法解决的问题是一个分布式系统如何就某个值（决议）达成一致。一个典型的场景是，在一个分布式数据库系统中，如果各节点的初始状态一致，每个节点执行相同的操作序列，那么他们最后能得到一个一致的状态。为保证每个节点执行相同的命令序列，需要在每一条指令上执行一个“一致性算法”以保证每个节点看到的指令一致。一个通用的一致性算法可以应用在许多场景中，是分布式计算中的重要问题。因此从20世纪80年代起对于一致性算法的研究就没有停止过。节点通信存在两种模型：共享内存（Shared memory）和消息传递（Messages passing）。Paxos 算法就是一种基于消息传递模型的一致性算法。

不仅仅是分布式系统中，凡是多个过程需要达成某种一致的场合都可以使用Paxos 算法。一致性算法可以通过共享内存（需要锁）或者消息传递实现，Paxos 算法采用的是后者。Paxos 算法适用的几种情况：一台机器中多个进程/线程达成数据一致；分布式文件系统或者分布式数据库中多客户端并发读写数据；分布式存储中多个副本响应读写请求的一致性。

#### 分区（Partitioning）

原来所有的数据都是在一个数据库上的，网络IO及文件IO都集中在一个数据库上的，因此CPU、内存、文件IO、网络IO都可能会成为系统瓶颈。而分区的方案就是把某一个或某几张相关的表的数据放在一个独立的数据库上，这样就可以把CPU、内存、文件IO、网络IO分解到多个机器中，从而提升系统处理能力。

#### 分片（Replication）

分区有两种模式，一种是主从模式，用于做读写分离；另外一种模式是分片模式，也就是说把一个表中的数据分解到多个表中。一个分区只能是其中的一种模式。

#### 一致性哈希（Consistent Hashing）

一致性哈希算法是分布式系统中常用的算法。比如，一个分布式的存储系统，要将数据存储到具体的节点上，如果采用普通的hash方法，将数据映射到具体的节点上，如key%N，key是数据的key，N是机器节点数，如果有一个机器加入或退出这个集群，则所有的数据映射都无效了，如果是持久化存储则要做数据迁移，如果是分布式缓存，则其他缓存就失效了。

一致性哈希基本解决了在P2P环境中最为关键的问题——如何在动态的网络拓扑中分布存储和路由。每个节点仅需维护少量相邻节点的信息，并且在节点加入/退出系统时，仅有相关的少量节点参与到拓扑的维护中。所有这一切使得一致性哈希成为第一个实用的DHT算法。

### NoSQL的优点/缺点

|  |  |
| --- | --- |
| 优点 | 缺点 |
| 1.易扩展  2.高性能  3.数据类型灵活  4.高可用 | 1.没有标准  2.没有存储过程  3.不支持sql  4.功能不够完善 |

#### 优点

1. **易扩展**

NoSQL数据库种类繁多，但是一个共同的特点都是去掉关系数据库的关系型特性。数据之间无关系，这样就非常容易扩展。也无形之间，在架构的层面上带来了可扩展的能力。

1. **大数据量，高性能**

NoSQL数据库都具有非常高的读写性能，尤其在大数据量下，同样表现优秀。这得益于它的无关系性，数据库的结构简单。一般MySQL使用Query Cache，每次表的更新Cache就失效，是一种大粒度的Cache，在针对web2.0的交互频繁的应用，Cache性能不高。而NoSQL的Cache是记录级的，是一种细粒度的Cache，所以NoSQL在这个层面上来说就要性能高很多了。

1. **灵活的数据模型**

NoSQL无需事先为要存储的数据建立字段，随时可以存储自定义的数据格式。而在关系数据库里，增删字段是一件非常麻烦的事情。如果是非常大数据量的表，增加字段简直就是一个噩梦。这点在大数据量的web2.0时代尤其明显。

1. **高可用**

NoSQL在不太影响性能的情况，就可以方便的实现高可用的架构。比如Cassandra，HBase模型，通过复制模型也能实现高可用。

#### 缺点

1. **没有标准**

没有对NoSQL数据库许多定义的标准，所以没有两个NoSQL数据库是平等的。

1. **没有存储过程**

NoSQL数据库中大多没有存储过程。

1. **不支持SQL**

NoSQL大多不提供对SQL的支持：如果不支持SQL这样的工业标准，将会对用户产生一定的学习和应用迁移成本。

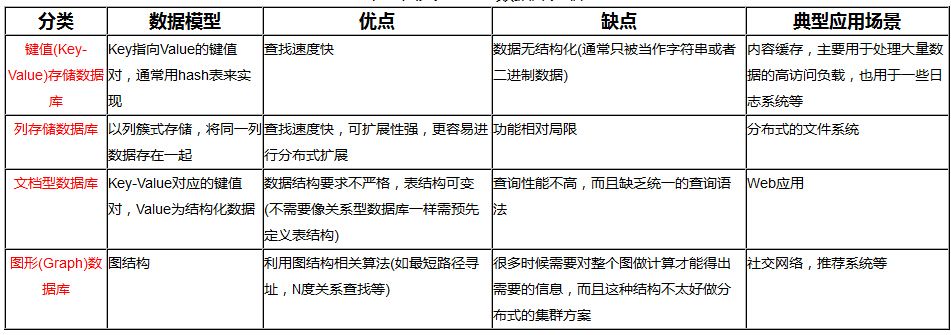
1. **支持的特性不够丰富，产品不够成熟**

现有产品所提供的功能都比较有限，不像MS SQL Server和Oracle那样能提供各种附加功能，比如BI和报表等。大多数产品都还处于初创期，和关系型数据库几十年的完善不可同日而语。

## NoSQL与SQL的对比

|  |  |  |
| --- | --- | --- |
|  | RDBMS | NoSQL |
| 模式 | 预定义的模式 | 没有预定义的模式 |
| 查询语言 | 结构化查询语言（SQL） | 没有声明性查询语言 |
| 一致性 | 严格的一致性 | 最终一致性 |
| 事务 | 支持 | 不支持 |
| 理论基础 | ACID | CAP, BASE |
| 扩展 | 纵向扩展 | 横向扩展(分布式) |

## NoSQL数据库的分类



### 1. 键值(Key-Value)存储数据库

这一类数据库主要会使用到一个哈希表，这个表中有一个特定的键和一个指针指向特定的数据。Key/value模型对于IT系统来说的优势在于简单、易部署。但是如果DBA只对部分值进行查询或更新的时候，Key/value就显得效率低下了。

E. g.:

1. Tokyo Cabinet/Tyrant
2. Redis
3. Voldemort
4. Oracle BDB

### 2. 列存储数据库

这部分数据库通常是用来应对分布式存储的海量数据。键仍然存在，但是它们的特点是指向了多个列。这些列是由列家族来安排的。

E. g.:

1. Cassandra
2. HBase
3. Riak

### 3. 文档型数据库

文档型数据库的灵感是来自于Lotus Notes办公软件的，而且它同第一种键值存储相类似。该类型的数据模型是版本化的文档，半结构化的文档以特定的格式存储，比如JSON。文档型数据库可 以看作是键值数据库的升级版，允许之间嵌套键值。而且文档型数据库比键值数据库的查询效率更高。

E. g.:

1. CouchDB
2. MongoDB
3. SequoiaDB

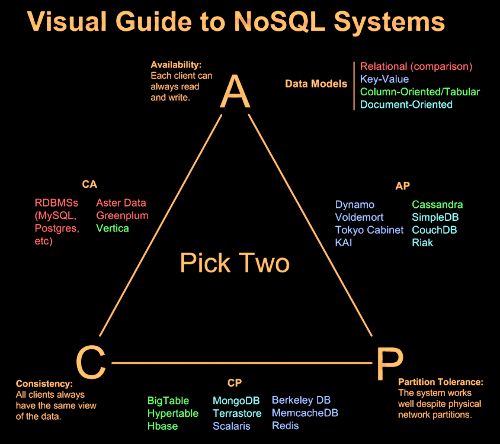
### 4. 图形(Graph)数据库

图形结构的数据库同其他行列以及刚性结构的SQL数据库不同，它是使用灵活的图形模型，并且能够扩展到多个服务器上。NoSQL数据库没有标准的查询语言(SQL)，因此进行数据库查询需要制定数据模型。许多NoSQL数据库都有REST式的数据接口或者查询API。

E. g.:

1. Neo4J
2. InfoGrid
3. Infinite Graph

## 主流NoSQL数据库介绍及其适用场景



### 1. Redis

#### 1.1 介绍

Redis是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。从2010年3月15日起，Redis的开发工作由VMware主持。从2013年5月开始，Redis的开发由Pivotal赞助。

#### 1.2 适用场景

* 数据变化较少，执行预定义查询，进行数据统计的应用程序
* 需要提供数据版本支持的应用程序
* 例如：股票价格、数据分析、实时数据搜集、实时通讯、分布式缓存

### 2. MongoDB

#### 2.1 介绍

MongoDB 是一个基于分布式文件存储的数据库。由 C++ 语言编写。旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。

MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。

#### 2.2 适用场景

* 需要动态查询支持
* 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

### 3. Neo4j

#### 3.1 介绍

Neo4j是一个高性能的,NOSQL图形数据库，它将结构化数据存储在网络上而不是表中。它是一个嵌入式的、基于磁盘的、具备完全的事务特性的Java持久化引擎，但是它将结构化数据存储在网络(从数学角度叫做图)上而不是表中。Neo4j也可以被看作是一个高性能的图引擎，该引擎具有成熟数据库的所有特性。

#### 3.2 适用场景

* 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

### 4. Cassandra

#### 4.1 介绍

Apache Cassandra 是一套开源分布式 Key-Value 存储系统。它最初由 Facebook 开发，用于储存特别大的数据。 Cassandra 不是一个数据库，它是一个混合型的非关系的数据库，类似于 Google 的 BigTable。Cassandra 的数据模型是基于列族（Column Family）的四维或五维模型。

#### 适用场景

* + 银行业，金融业
  + 写比读更快

### 5. HBase

#### 5.1 介绍

HBase是一个分布式的、面向列的开源数据库，该技术来源于Google论文“Bigtable：一个结构化数据的分布式存储系统”。就像Bigtable利用了Google文件系统（File System）所提供的分布式数据存储一样，HBase在Hadoop之上提供了类似于Bigtable的能力。它是一个适合于非结构化数据存储的数据库。另一个不同的是HBase基于列的而不是基于行的模式。

#### 5.2适用场景

* 对大数据进行随机、实时访问的场合
* 例如： Facebook消息数据库

### CouchDB

#### 6.1介绍

CouchDB 是一个开源的面向文档的数据库管理系统，可以通过 RESTful JavaScript Object Notation (JSON) API 访问。术语 “Couch” 是 “Cluster Of Unreliable Commodity Hardware” 的首字母缩写，它反映了 CouchDB 的目标具有高度可伸缩性，提供了高可用性和高可靠性，即使运行在容易出现故障的硬件上也是如此。

#### 6.2适用场景

* 数据变化较少，执行预定义查询，进行数据统计的应用程序
* 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。

## NoSQL优秀应用实例

### 1. 新浪微博 - Redis

新浪微博从技术上来说，每天用户发表特别容易，这造成每天新增的数据量都是百万级的、上千万级的这样一个量。这样你经常要面对的一个问题就是增加服务器，因为一般一台MySQL服务器，它可能支撑的规模也就是几千万，或者说复杂一点只有几百万，这样，可能每天都要增加服务器，从而解决所你面对的这些问题。

目前新浪微博是Redis全球最大的用户，在新浪有200多台物理机，400多个端口正在运行着Redis, 有 4G的数据跑在Redis上来为微博用户提供服务。

新浪微博面临的问题如下：

* 数据结构(Data Structure)需求越来越多, 但memcache中没有, 影响开发效率
* 随着读操作的量的上升，性能问题需要解决，经历的过程有:

数据库读写分离(M/S)-->数据库使用多个Slave-->增加Cache (memcache)-->转到Redis

* 解决写的问题：

水平拆分，对表的拆分，将有的用户放在这个表，有的用户放在另外一个表。

* 可靠性需求

Cache的"雪崩"问题难以解决，面临着快速恢复的挑战。

* 开发成本需求

Cache和DB的一致性维护成本越来越高，但开发需要跟上不断涌入的产品需求。且硬件成本最贵的就是数据库层面的机器，基本上比前端的机器要贵几倍，主要是IO密集型，很耗硬件。

* 维护性复杂

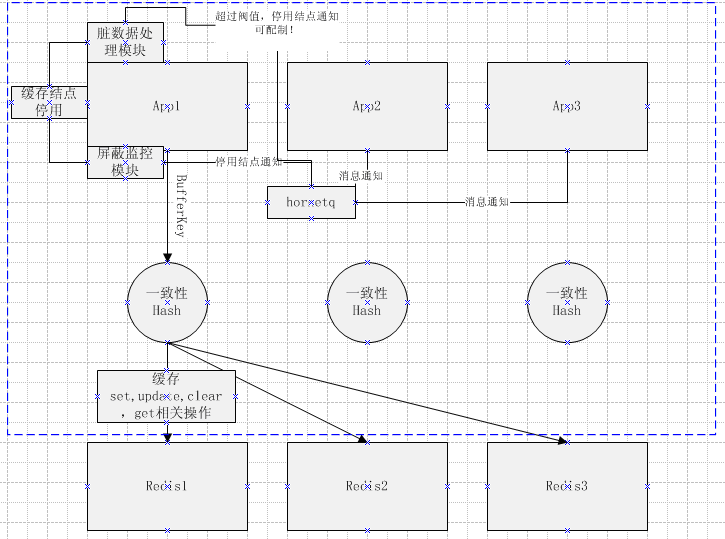
一致性维护成本越来越高

BerkeleyDB使用B树，会一直写新的，内部不会有文件重新组织；这样会导致文件越来越大；大的时候需要进行文件归档，归档的操作要定期做，这样，就需要有一定的down time。

基于以上考虑， 新浪微博选择了Redis。

在新浪NoSQL和MySQL在大多数情况下是结合使用的，根据应用的特点选择合适存储方式。譬如：关系型数据，例如：索引使用MySQL存储，非关系数据库，例如：一些K/V需求的，对并发要求比较高的放入Redis存储。

新浪微博团队通过修改Redis源码满足自己的业务需求：完善它的replication机制，加入position的概念，让维护更容易，同时failover能力也大大增强。改善Hashset在RDB里面的存储方式，提升复杂数据类型的加载速度。



**业务场景如下：**

**1. 业务使用方式：**

* hash sets: 关注列表, 粉丝列表, 双向关注列表(key-value(field), 排序)
* string(counter): 微博数, 粉丝数, ...(避免了select count(\*) from ...)
* sort sets(自动排序): TopN, 热门微博等, 自动排序
* lists(queue): push/sub提醒,...
* 上述四种, 从精细化控制方面，hash sets和string(counter)推荐使用, sort sets和lists(queue)不推荐使用
* 还可通过二次开发，进行精简。比如: 存储字符改为存储整形, 16亿数据, 只需要16G内存
* 存储类型保存在3种以内，建议不要超过3种；
* 将memcache +myaql 替换为Redis：
* Redis作为存储并提供查询，后台不再使用mysql，解决数据多份之间的一致性问题；

**2. 对大数据表的存储**（eg：140字微博的存储）

* 一个库就存唯一性id和140个字；
* 另一个库存id和用户名，发布日期、点击数等信息，用来计算、排序等，等计算出最后需要展示的数据时再到第一个库中提取微博内容；

**3. 一些技巧**

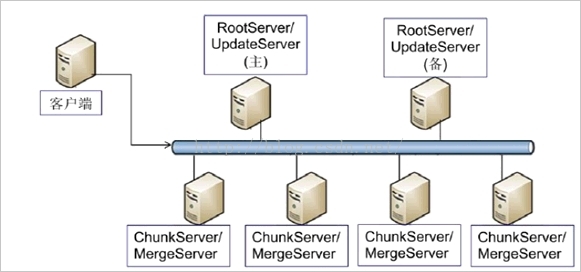
* 很多应用, 可以承受数据库连接失败, 但不能承受处理慢
* 一份数据, 多份索引(针对不同的查询场景)
* 解决IO瓶颈的唯一途径: 用内存
* 在数据量变化不大的情况下，优先选用Redis

### 2. 淘宝数据平台 – Oceanbase, Tair（均为自研）

数据产品的一个最大特点是数据的非实时写入，正因为如此，可以认为在一定的时间段内，整个系统的数据是只读的。这为设计缓存奠定了非常重要的基础。一些对实效性要求很高的数据，例如针对搜索词的统计数据，希望能尽快推送到数据产品前端，所以在内存中做实时计算，并把计算结果在尽可能短的时间内刷新到 NoSQL存储设备中，供前端产品调用。

淘宝Oceanbase的设计之初，是这样的。公司通过对淘宝的在线存储需求进行分析发现：

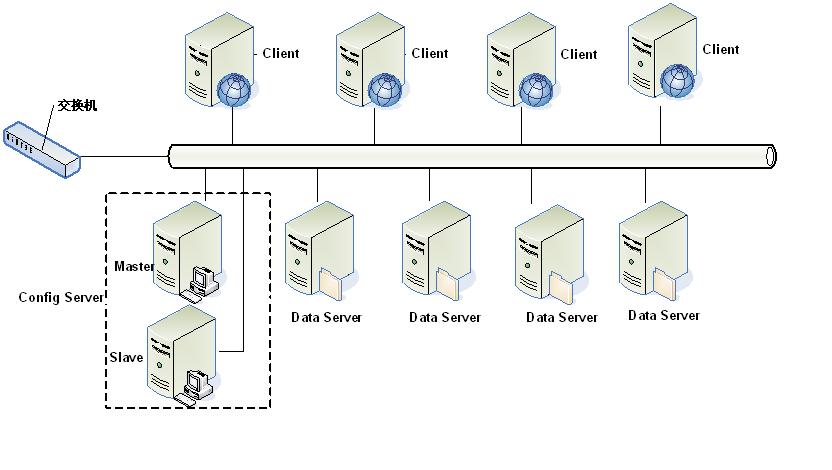
淘宝的数据总量比较大，未来一段时间，比如五年之内的数据规模为百TB级别，千亿条记录，另外，数据膨胀很快，传统的分库分表对业务造成很大的压力，必须设计自动化的分布式系统。所以有了淘宝Oceanbase，它以一种很简单的方式满足了未来一段时间的在线存储需求，并且还获得了一些其它特性，如高效支持跨行跨表事务，这对于淘宝的业务是非常重要的。



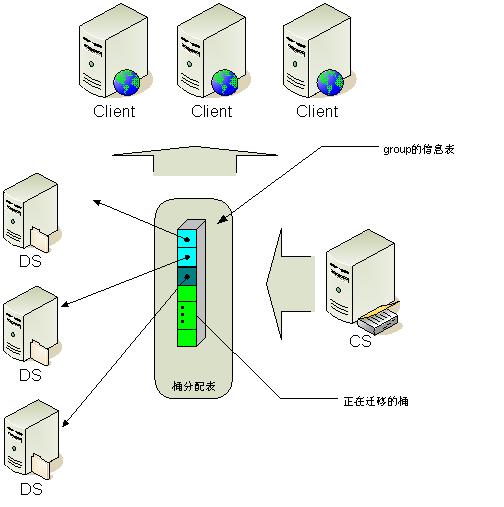
OceanBase由如下几个部分组成：

* 客户端：用户使用OceanBase的方式和MySQL数据库完全相同，支持JDBC、 C客户端访问，等等。基于MySQL数据库开发的应用程序、工具能够直接迁移到OceanBase。
* RootServer：管理集群中的所有服务器，子表（tablet）数据分布以及副本管理。 RootServer一般为一主一备，主备之间数据强同步。
* UpdateServer：存储OceanBase系统的增量更新数据。UpdateServer一般为一主一备，主备之间可以配置不同的同步模式。部署时，UpdateServer进程和RootServer进程往往共用物理服务器。
* ChunkServer：存储OceanBase系统的基线数据。基线数据一般存储两份或者三份，可配置。
* Merge Server：接收并解析用户的SQL请求，经过词法分析、语法分析、查询优化等一系列操作后转发给相应的ChunkServer或者UpdateServer。如果请求的数据分布在多台ChunkServer上，MergeServer还需要对多台ChunkServer返回的结果进行合并。客户端和MergeServer之间采用原生的MySQL通信协议，MySQL客户端可以直接访问MergeServer。

淘宝Tair是由淘宝自主开发的Key/Value结构数据存储系统，并且于 2010年6月30号在淘宝开源平台上正式对外开源，在淘宝网有着大规模的应用。用户在登录淘宝、查看商品详情页面或者在淘江湖和好友“捣浆糊”的时候，都在直接或间接地和Tair交互。淘宝将Tair开源，希望有更多的用户能从我们开发的产品中受益，更希望依托社区的力量，使Tair有更广阔的发展空间。

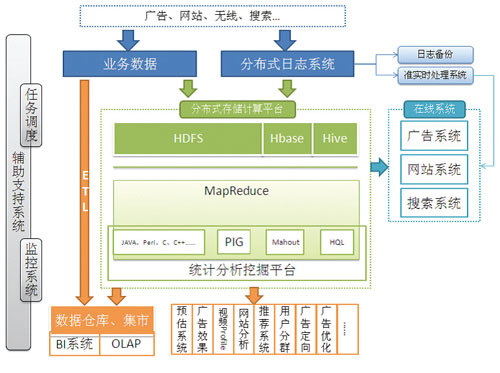


Tair 的分布采用的是一致性哈希算法, 对于所有的key, 分到Q个桶中, 桶是负载均衡和数据迁移的基本单位. config server 根据一定的策略把每个桶指派到不同的data server上. 因为数据按照key做hash算法, 所以可以认为每个桶中的数据基本是平衡的. 保证了桶分布的均衡性, 就保证了数据分布的均衡性。



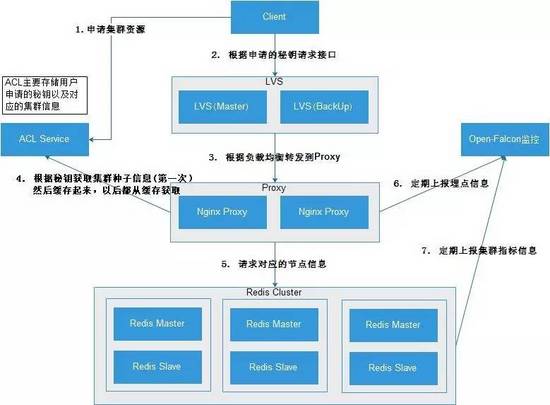
### 3. 优酷运营数据分析 – HBase, MongoDB, Redis

优酷作为一家大型视频网站，拥有海量播放流畅的视频。它秉承注重用户体验这一产品技术理念，将绝大部分存储用在视频资源上。通过建设专用的视频CDN，建立了可自由扩展、性能优异的架构，在提供更好用户体验的同时优化了存储资源。在除视频资源外的其他方面，优酷也累积了海量数据：仅运营数据，每天收集到的网站各类访问日志总量已经达到TB级，经分析及压缩处理后留存下来的历史运营数据已达数百TB，很快将会达到 PB级，5年后数据量将会达到几十PB级。



目前优酷的在线评论业务已部分迁移到MongoDB，运营数据分析及挖掘处理目前在使用Hadoop/HBase;在Key-Value产品方面，它也在寻找更优的 Memcached替代品，如Redis，相对于Memcached，除了对Value的存储支持三种不同的数据结构外，同一个Key的Value进行部分更新也会更适合一些对Value频繁修改的在线业务;同时在搜索产品中应用了Tokyo Tyrant;对于Cassandra等产品也进行过研究。

对于优酷来说，仍处于飞速发展阶段，已经在考虑未来自建数据中心，提高数据处理能力，从网站的运营中发掘出更多信息，为用户提供更好的视频服务。

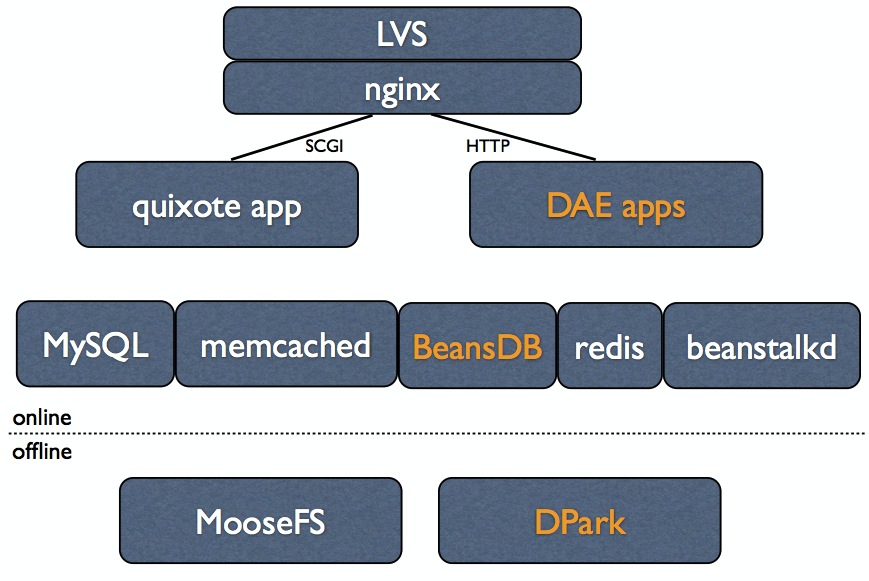


### 4. 豆瓣社区 – BeansDB（自研KV数据库）

它采用类似memcached的去中心化结构，在客户端实现数据路由。目前只提供了Python版本的客户端，其它语言的客户端可以由memcached的客户端稍加改造得到。它具有如下特性：

1. 高可用：通过多个可读写的用于备份实现高可用
2. 最终一致性：通过哈希树实现快速完整数据同步(短时间内数据可能不一致)
3. 容易扩展：可以在不中断服务的情况下进行容量扩展。
4. 高性能：异步网络IO, 日志结构的存储方式Bitcask.
5. 简单协议：Memcache兼容协议，大量可用客户端

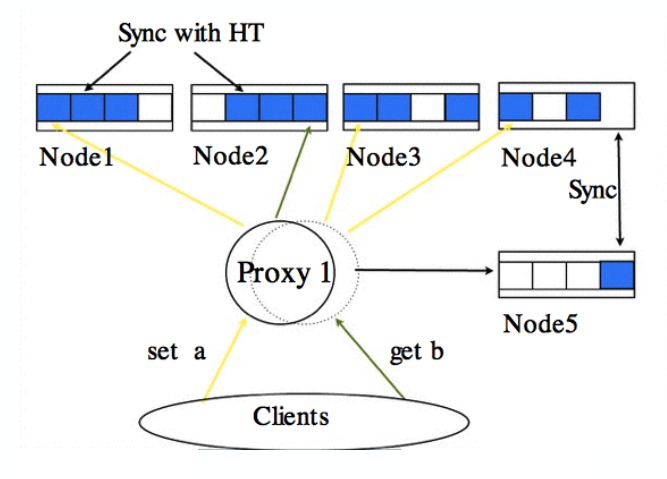
目前，BeansDB在豆瓣主要部署了两个集群：一个集群用于存储数据库中的大文本数据，比如日记、帖子一类;另外一个豆瓣FS集群，主要用于存储媒体文件，比如用户上传的图片、豆瓣电台上的音乐等。



BeansDB采用Key-Value存储架构，其最大的特点是具有高度的可伸缩性;在BeansDB的架构下，在大数据量下，扩展数据节点将轻而易举，只需要添加硬件，安装软件，修改相应的配置文件即可。

BeansDB项目可以说是一个简化版的AWS DynamoDB，。BeansDB对key做哈希运算找到节点来实现分布和冗余， 一个写操作会写好几个节点，而现在的配置是写三份读一份。BeansDB主要的特点是支持海量KV数据库——相比Redis这种支持几十个G到几百个G的内存KV数据库，BeansDB可以支持到上百T的数据。另外BeansDB最大的好处就是运维很简单，性能、扩容都很好，也实现了最终一致性。

BeansDB在可用性方面也有很大的优势，任何一个节点宕机都不会受到影响，数据是自动伸缩冗余的。在运维方面也很简单，基本上没有什么用户数据的冗余残余，所有数据通过一个同步脚本可以快速同步。

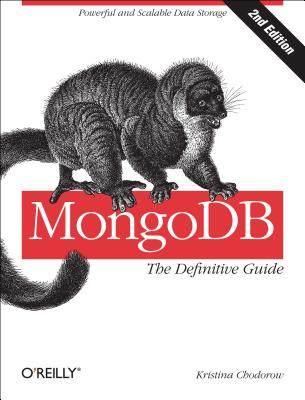


## 学习资料

### 1. 书籍

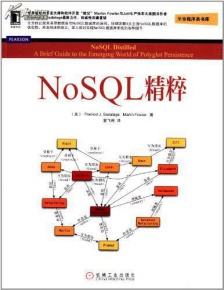
#### 1.1 *MongoDB: The Definitive Guide*（Kristina Chodorow）

MongoDB是入门NoSQL数据库的最好选择之一。本书讲解了所有关于MongoDB的基础知识，是本很好的入门书籍。



#### 1.2 *NoSQL精粹* （Pramod J.Sadalage, Martin Fowler）

本书全方位比较关系型数据库与NoSQL数据库的异同，详细讲解4大主流NoSQL数据库的优劣势、用法和适用场合，深入探讨实现NoSQL数据库系统的各种细节。此书对于技术选型有很好的指导作用。



#### 1.3 各种NoSQL数据库的官方文档

有一定计算机基础的人还是最推荐看官方文档，官方文档对其产品的理解永远是最深的，对于开发者若能理解其设计原则，上手比看书要快。

### 2. 视频

#### 2.1 Getting Started - NoSQL - MongoDB

地址：

https://www.youtube.com/watch?v=5rbFoSGHErA&list=PLf0swTFhTI8ra5T5B7QsNuu5yxiEdd6Ro

老外的视频，MongoDB的一个比较通俗易懂的教程。

#### Cassandra-NoSQL-Tutorials

地址：

https://www.youtube.com/watch?v=8G4a4G3S654&list=PLpE\_8MUgZj5vSp1Q\_5GyDKBgy9dG1ifdE

同样是老外的Cassandra的系列教程。

#### Redis Server Tutorial

地址：

https://www.youtube.com/watch?v=fyV3OK1fCr0&list=PLpIXNzrq3JHQ8-QCJqrC2ihrGJkjdN2J6

Redis的系列教程，不过侧重于分布式缓存功能的实现。这也是Redis的主要使用场景。

### 3. 边用边Google

工具类的事物永远是边用边学最快，真正用过了（尤其是遇到过超高并发/存储的情况）才会体验到NoSQL的好处。

## 进一步学习

论文下载地址：http://pan.baidu.com/s/1eSP3sP0 密码：kmok

### 1. 分布式算法

1. [Paxos made simple](http://www.cs.utexas.edu/users/lorenzo/corsi/cs380d/past/03F/notes/paxos-simple.pdf)

一篇通俗讲解paxos算法的论文，由paxos算法发明者Leslie Lamport所写，是其发明paxos算法的论文的简化版。此算法用于确定分布式系统的共识

1. [Byzantine generals problem](http://people.cs.uchicago.edu/~shanlu/teaching/33100_wi15/papers/byz.pdf)

一篇研究“拜占庭将军”问题的论文。“拜占庭将军”是分布式场景的典型问题，paxos算法就是用来解决此问题的。

1. Research on the improvement of MongoDB Auto-Sharding in cloud environment

一篇研究MongoDB分片算法的论文。分片是NoSQL数据库的基本功能

### 2. NoSQL数据库的研究及底层实现

1. Bigtable: A distributed storage system for structured data

BigTable的设计论文，HBase是其开源实现，是一个典型的基于列的NoSQL数据库。此篇论文是Google的“三大马车”之一。

1. Optimizing event polling for network-intensive applications: A case study on redis

一篇研究Redis底层Networking IO的论文，并优化了原有的epoll模型，命名为FlexPoll。

1. Performance evaluation of a MongoDB and Hadoop platform for scientific data analysis

一篇研究MongoDB和Hadoop在科学计算场景的性能的论文（科学计算是cpu/gpu-intensive而非i/o密集型）。

### 3. NoSQL应用案例

1. Big data analysis with MongoDB for decision support system

这篇论文使用MongoDB对商业数据做了大数据分析，为企业提供决策，并比较了RDBMS与NoSQL在数据分析方面的优劣。

1. Implementing joins over HBase on cloud platform

一篇在论述如何在HBase上实现Join功能的论文。Join在分布式环境下实现非常困难，为此此篇论文设计了2种算法：MapReduceJoin与ParallelHashJoin。