获取教材和讲义 PPT 等各种课程资料请访问 http://dblab.xmu.edu.cn/node/422

=课程教材由林子雨老师根据网络资料编著=

厦门大学计算机科学系教师 林子雨 编著

http://www.cs.xmu.edu.cn/linziyu

2013 年 9 月

前言

本教程由厦门大学计算机科学系教师林子雨编著，可以作为计算机专业研究生课程《大数据技术基础》的辅助教材。

本教程的主要内容包括：大数据概述、大数据处理模型、大数据关键技术、大数据时代面临的新挑战、NoSQL 数据库、云数据库、Google Spanner、Hadoop、HDFS、HBase、MapReduce、Zookeeper、流计算、图计算和 Google Dremel 等。

本教程是林子雨通过大量阅读、收集、整理各种资料后精心制作的学习材料，与广大数据库爱好者共享。教程中的内容大部分来自网络资料和书籍，一部分是自己撰写。对于自写内容，林子雨老师拥有著作权。

本教程 PDF 文档及其全套教学 PPT 可以通过网络免费下载和使用（下载地址：http://dblab.xmu.edu.cn/node/422）。教程中可能存在一些问题，欢迎读者提出宝贵意见和建议！

本教程已经应用于厦门大学计算机科学系研究生课程《大数据技术基础》，欢迎访问 2013 班级网站 http://dblab.xmu.edu.cn/node/423。

林子雨的 E-mail 是：ziyulin@xmu.edu.cn。

林子雨的个人主页是：http://www.cs.xmu.edu.cn/linziyu。

林子雨于厦门大学海韵园

2013 年 9 月

第 10 章 NoSQL 数据库

厦门大学计算机科学系教师 林子雨 编著

个人主页：http://www.cs.xmu.edu.cn/linziyu

课程网址：http://dblab.xmu.edu.cn/node/422

2013 年 9 月

第 10 章 NoSQL 数据库

NoSQL 数据库，指的是非关系型的数据库。随着互联网 web2.0 网站的兴起，传统的关系数据库在应付 web2.0 网站，特别是超大规模和高并发的 SNS 类型的 web2.0 纯动态网站方面，已经显得力不从心，暴露了很多难以克服的问题，而非关系型的数据库则由于其本身的特点得到了非常迅速的发展。

本章介绍 NoSQL 数据库相关知识，内容要点如下：

- NoSQL 简介
- NoSQL 现状
- 为什么要使用 NoSQL 数据库
- NoSQL 数据库的特点
- NoSQL 的五大挑战
- 对 NoSQL 的质疑
- NoSQL 的三大基石
- NoSQL 数据库与关系数据库的比较
- 典型的 NoSQL 数据库分类
- NoSQL 数据库开源软件

10.1 NoSQL 简介

NoSQL，意即反 SQL 运动，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。NoSQL 的拥护者们提倡运用非关系型的数据存储，相对于目前铺天盖地的关系型数据库运用，这一概念无疑是一种全新的思维的注入。反 SQL 运动的主要倡导者都是 Web 和 Java 开发者，他们中许多人都在创业的初期历经了资金短缺并因此与

Oracle 说再见，然后效仿 Google 和 Amazon 的道路建设起自己的数据存储解决方案，并随后将自己的成果开源发布。现在，他们的开源数据商店管理着成百 TB 甚至 PB 的数据，由于 Web 2.0 和云计算的兴起，无论从技术上还是从经济上他们都无需再返回从前，甚至连想也不用想。

“Web 2.0 的企业应该抓住机会，他们需要可扩展性”，总部设在伦敦的 NoSQL 会议组织者 Johan Oskarsson 说。Oskarsson 任职于著名的音乐网站 Last.fm，其他的大多数与会者也都是网络开发者。

Oskarsson 说，许多人甚至抛弃了 MySQL 开源数据库这个长期以来 Web 2.0 的宠儿，而改由 NoSQL 的方案来替代，因为优势实在是引人注目。过度的商业化使得 MySQL 失去原来的优势。

例如，Facebook 建立了自己的 Cassandra 数据商店，并且在其网站上重点推出一项新的搜索功能，没有使用到现有的 MySQL 数据库。据 Facebook 的工程师 Avinash Lakshma 介绍，Cassandra 仅用 0.12 毫秒就可以写入 50GB 的数据，比 MySQL 快了超过 2500 倍。Google 也开始公测他们的云数据库 Fusion Tables，这是一个和传统数据库完全不同的数据库，主要优势在于能够简单地解决关系型数据库中管理不同类型数据的麻烦，以及排序整合的常见操作的性能问题等。

10.2 NoSQL 现状

现今的计算机体系结构在数据存储方面要求具备庞大的水平扩展性（即能够连接多个软硬件的特性，这样可以将多个服务器从逻辑上看成一个实体），而 NoSQL 致力于改变这一现状。目前 Google 的 BigTable 和 Amazon 的 Dynamo 使用的就是 NoSQL 型数据库。

NoSQL 项目的名字上看不出什么相同之处，但是，它们通常在某些方面相同：它们可以处理超大量的数据。

这场革命目前仍然需要等待。的确，NoSQL 对大型企业来说还不是主流，但是，一两年之后很可能就会变个样子。在 NoSQL 运动的最新一次聚会中，来自世界各地的 150 人挤满了 CBS Interactive 的一间会议室，分享他们如何推翻缓慢而昂贵的关系数据库的“暴政”，怎样使用更有效和更便宜的方法来管理数据。

“关系型数据库给你强加了太多东西。它们要你强行修改对象数据，以满足 RDBMS（relational database management system，关系型数据库管理系统）的需要”，在 NoSQL 拥护

者们看来，基于 NoSQL 的替代方案“只是给你所需要的”。

虽然有些人认为这是摆脱 MySQL 和 PostgreSQL 等传统的开源关系数据库的机会，实际上事情并不是这么简单，从这些有趣的变化中我们得出一些启示：

1）关系数据库并不适合所有的数据模型；

2）关系数据库扩展难度大，特别是当你一开始就设计为单机配置，未进行分布式设计时；

3）标准化通常会伤害到性能；

4）在许多应用中，主键就是你的一切。

新的 NoSQL 数据存储完全改变了传统的观念，但总的来说，它们借鉴了一套类似的高级特征，但它们并非能够满足一切。下面给出一个列表，让我们来看看 NoSQL 正试图实现什么：

1）反标准化，通常是无模式的，文档型存储；

2）以 key/value 为基础，支持通过 key 进行查找；

3）水平扩展；

4）内置复制；

5）HTTP/REST 或很容易编程的 API；

6）支持 MapReduce 的风格的编程；

7）最终一致性。

如果还要列的话，可能还可以列出一打来。但前面两个是对传统数据库最大的叛离，当然你也可以坚持使用 MySQL，并将其去关系化，这也是 FriendFeed 要做的事情，FriendFeed 使用 MySQL 作为后端，实现分布式 key/value 存储。对这些分布式无模式的数据存储，开始有一个新名称来称呼，那就是 NoSQL。

10.3 为什么要使用 NoSQL 数据库?

随着互联网 web2.0 网站的兴起，非关系型的数据库现在成了一个极其热门的新领域，非关系数据库产品的发展非常迅速。而传统的关系数据库在应付 web2.0 网站，特别是超大规模和高并发的 SNS 类型的 web2.0 纯动态网站方面，已经显得力不从心，暴露了很多难以克服的问题，主要包括以下几个方面：

- **对数据库高并发读写的性能需求：**web2.0 网站要根据用户个性化信息来实时生成

动态页面和提供动态信息，所以，基本上无法使用动态页面静态化技术，因此数据库并发负载非常高，往往要达到每秒上万次读写请求。关系数据库应付上万次SQL查询还勉强顶得住，但是应付上万次SQL写数据请求，硬盘IO就已经无法承受了。其实对于普通的BBS网站，往往也存在对高并发写请求的需求。

- **对海量数据的高效率存储和访问的需求：**对于大型的SNS网站，每天用户产生海量的用户动态，以国外的Friendfeed为例，一个月就达到了2.5亿条用户动态，对于关系数据库来说，在一张2.5亿条记录的表里面进行SQL查询，效率是极其低下甚至是不可忍受的。再例如大型web网站的用户登录系统，例如腾讯和盛大，动辄数以亿计的帐号，关系数据库也很难应付。
- **对数据库的高可扩展性和高可用性的需求：**在基于web的架构当中，数据库是最难进行横向扩展的，当一个应用系统的用户量和访问量与日俱增的时候，你的数据库却没有办法像网页服务器和应用服务器那样简单地通过添加更多的硬件和服务节点来扩展性能和负载能力。对于很多需要提供24小时不间断服务的网站来说，对数据库系统进行升级和扩展是非常痛苦的事情，往往需要停机维护和数据迁移，为什么数据库不能通过不断地添加服务器节点来实现水平扩展呢？

在上面提到的“三高”需求面前，关系数据库遇到了难以克服的障碍，而对于 web2.0网站来说，关系数据库的很多主要特性却往往无用武之地，主要表现在以下几个方面：

- **数据库事务一致性需求：**很多web实时系统并不要求严格的数据库事务，对读一致性的要求很低，有些场合对写一致性要求也不高。因此，数据库事务管理成了数据库高负载下一个沉重的负担。
- **数据库的写实时性和读实时性需求：**对于关系数据库来说，插入一条数据之后立刻查询，是肯定可以读出来这条数据的，但是对于很多web应用来说，并不要求这么高的实时性。
- **对复杂的SQL查询，特别是多表关联查询的需求：**任何大数据量的web系统，都非常忌讳多个大表的关联查询以及复杂数据分析类型的复杂SQL报表查询，特别是SNS类型的网站，往往从需求以及产品设计角度就避免了这种情况的产生。一般而言，这类web系统更多的只是单表的主键查询以及单表的简单条件分页查询，SQL的功能被极大地弱化了。

因此，关系数据库在这些越来越多的应用场景下就显得不那么合适了，为了解决这类问题的非关系数据库由此应运而生。

NoSQL 是非关系型数据存储的广义定义，它打破了长久以来关系型数据库与 ACID 理论大一统的局面。NoSQL 数据存储不需要固定的表结构，通常也不存在连接操作。在大数据存取上具备关系型数据库无法比拟的性能优势。该术语在 2009 年初得到了广泛的认同。

当今的应用体系结构需要数据存储在横向伸缩性上能够满足需求。而 NoSQL 存储就是为了实现这个需求。Google 的 BigTable 与 Amazon 的 Dynamo 是非常成功的商业 NoSQL 实现。一些开源的 NoSQL 体系，如 Facebook 的 Cassandra，Apache 的 HBase，也得到了广泛的认同。从这些 NoSQL 项目的名字上看不出什么相同之处，比如 Hadoop、Voldemort、Dynomite 等，当然，还存在其它很多 NoSQL 项目。

10.4 NoSQL 数据库的特点

NoSQL 数据库的主要特点包括以下几个方面：

- **灵活的可扩展性**

多年以来，数据库管理员们都是通过“纵向扩展”的方式（当数据库的负载增加的时候，购买更大型的服务器来承载增加的负载）来进行扩展的，而不是通过“横向扩展”的方式（当数据库负载增加的时候，在多台主机上分配增加的负载）来进行扩展。但是，随着交易率和可用性需求的增加，数据库也正在迁移到云端或虚拟化环境中，“横向扩展”在商用硬件方面的经济优势变得更加明显了，对各大企业来说，这种“诱惑”是无法抗拒的。在商业硬件集群上，要对 RDBMS 做“横向扩展”，并不是很容易，但是，各种新类型的 NoSQL 数据库主要是为了进行透明的扩展来利用新节点而设计的，而且，它们通常都是为了低成本的商用硬件而设计的。

- **大数据**

在过去的十年里，正如交易率发生了翻天覆地的增长一样，需要存储的数据量也发生了急剧的膨胀。O’Reilly 把这种现象称为“数据的工业革命”。为了满足数据量增长的需要，RDBMS 的容量也在日益增加，但是，对于一些企业来说，随着交易率的增加，单一数据库需要管理的数据约束的数量也变得越来越让人无法忍受了。现在，大量的“大数据”可以通过 NoSQL 系统（例如：Hadoop）来处理，它们能够处理的数据量远远超出了最大型的 RDBMS 所能处理的极限。

- **“永别了”？DBA 们！**

在过去的几年里，虽然一些 RDBMS 供应商们声称在可管理性方面做出了很多的改进，

但是，高端的 RDBMS 系统维护起来仍然十分昂贵，而且还需要训练有素的 DBA 们的协助。DBA 们需要亲自参与高端的 RDBMS 系统的设计、安装和调优。NoSQL 数据库从一开始就是为了降低管理方面的要求而设计的；从理论上来说，自动修复、数据分配和简单的数据模型，的确可以让管理和调优方面的要求降低很多。但是，DBA 的“死期将至”的谣言未免有些过于夸张了，毕竟总是需要有人对关键性的数据库的性能和可用性负责的。

- **经济**

NoSQL 数据库通常使用廉价的商用服务器集群来管理膨胀的数据和事务数量，而 RDBMS 通常需要依靠昂贵的专有服务器和存储系统来做到这一点。使用 NoSQL，每 GB 的成本或每秒处理的事务的成本，都比使用 RDBMS 的成本少很多倍，这可以让你花费更低的成本存储和处理更多的数据。

- **灵活的数据模型**

对于大型的生产性的 RDBMS 来说，变更管理是一件很令人头痛的事情。即使只对一个 RDBMS 的数据模型做了很小的改动，也必须要十分小心地管理，也许还需要停机或降低服务水平。NoSQL 数据库在数据模型约束方面是更加宽松的，甚至可以说并不存在数据模型约束。NoSQL 的键值数据库和文档数据库，可以让应用程序在一个数据元素里存储任何结构的数据。即使是规定更加严格的基于“大表”的 NoSQL 数据库（比如 Cassandra 和 HBase），通常也允许创建新列，这并不会造成什么麻烦。应用程序变更和数据库模式的变更，并不需要作为一个复杂的变更单元来管理。从理论上来说，这可以让应用程序迭代得更快，但是，很明显，如果应用程序无法维护数据的完整性，那么这会带来一些不良的副作用。

10.5 NoSQL 的五大挑战

NoSQL 的种种承诺引发了一场热潮，但是，在它们得到主流的企业的青睐以前，它们还有许多困难需要克服。下面是 NoSQL 需要面对的一些挑战：

- **成熟度**

RDBMS 系统已经发展很长时间了。NoSQL 的拥护者们认为，RDBMS 系统那超长的发展的年限，恰恰表示它们已经过时了；但是，对于大多数的 CIO 们来说，RDBMS 的成熟度更加令它们放心。大多数情况下，RDBMS 系统更加稳定，而且功能也更加丰富。相比之下，大多数的 NoSQL 数据库都是“前期制作”版本，许多关键性的功能还有待实现。对于大多数开发者来说，处于技术的最前沿的确是很令人兴奋的，但是，企业应该怀着极端谨慎的态

度来处理此事。

- **技术支持**

企业都希望能得到这样的保证：如果一个关键性的系统出现问题了，他们可以获得及时有效的技术支持。所有的 RDBMS 供应商都在竭尽全力地提供高水平的企业技术支持。相反，大多数的 NoSQL 系统都是开源项目，虽然对于每个 NoSQL 数据库来说，通常也会有一个或多个公司对它们提供支持，但是，那些公司通常是小型的创业公司，在支持的范围、支持的资源或可信度方面，它们和 Oracle、Microsoft 或 IBM 是无法相提并论的。

- **分析和商业智能化**

NoSQL 数据库现在已经可以满足现代的 Web2.0 应用程序的高度的可扩展性的要求了，这直接导致的结果是，它们的大多数功能都是面向这些应用程序而设计的。但是，在一个应用程序中，具有商业价值的数据早就已经超出了一个标准的 Web 应用程序需要的“插入-读取-更新-删除”的范畴了。在公司的数据库中进行商业信息的挖掘，可以提高企业的效率和竞争力，而且，对于所有的中到大型的公司来说，商业智能化（BI）一直是一个至关重要的 IT 问题。NoSQL 数据库几乎没有提供什么专用的查询和分析工具，即使是一个简单的查询，也要求操作者具有很高超的编程技术，而且，常用的 BI 工具是无法连接到 NoSQL 的。像 Hive 或 Pig 那样的新出现的一些解决方案，在这个方面或许可以提供一些帮助，它们可以让访问 Hadoop 集群中的数据变得更加容易，最后也许还会支持其他的 NoSQL 数据库。Quest 软件已经开发了一个产品——Toad for Cloud Databases——它给各种 NoSQL 数据库提供了专用的查询功能。

- **管理**

NoSQL 的设计目标是提供一个“零管理”的解决方案，但是，就目前而言，还远远没有达到这个目标。安装 NoSQL 还是需要很多技巧的，同时，维护它也需要付出很多的努力。

- **专业知识**

毫不夸张地说，全世界有数百万的开发者，他们都对 RDBMS 的概念和编程方法很熟悉，在每个业务部门中都有这样的开发者。相反，几乎每一个 NoSQL 开发者都正处于学习状态中。虽然这种情况会随着时间的推移而改变，但是现在，找到一些有经验的 RDBMS 程序员或管理员要比找到一个 NoSQL 专家容易得多。

10.6 对 NoSQL 的质疑

不可否认，NoSQL 拥有众多的支持者，但是，这里也不妨让我们聆听一下对 NoSQL 质疑的声音。有业界人士指出，NoSQL 这个项目的背景是站不住脚的。基于 SQL 的关系型数据库，确实在性能上存在一些瓶颈，但是，这大部分并不是这门 SQL 技术所造成的，而是因为在设计数据库的时候，表与表之间的关系、表的索引或者表空间的部署等等没有设计好而造成的。所以，关系型数据库性能不理想，并不能全部怪罪到这个技术本身上。通常情况下，对原有的数据库设计进行优化，往往可以在很大程度上提升数据库的性能。

有些业界人士仍然不是很看好 NoSQL 项目的前景，甚至有些人对其前途感到很悲观，认为 NoSQL 项目很难跟传统的关系型数据库相抗衡，甚至其想达到 MySQL 这个开源数据库的高度都很难。NoSQL 的质疑者主要从以下几个方面考虑问题：

- **NoSQL 很难实现数据的完整性**

很多关系型数据库中优秀的、实用的功能，在 NoSQL 数据库却无法实现。比如，在任何一个关系型的数据库中，都可以很容易地实现数据的完整性。如在 Oracle 数据库中，可以轻而易举地实现实体完整性(通过主键或者非空约束来实现)、参照完整性(通过主键、外键来实现)、用户定义完整性(通过约束或者触发器来实现)。

NoSQL 支持者也承认关系型数据库在数据完整性上的作用是不可替代的。但是他们却反驳说，企业可能用不到这么复杂的功能。对于这一点，很多人是不敢认同的。现在企业的任何一个应用，基本上都需要用到数据完整性。如现在大部分应用至少都需要有一个用户认证的过程。为此，在系统实现的过程中，需要在数据库中保存用户名。由于这个用户名涉及到用户的认证问题，为此用户名必须要唯一，此时就需要用到唯一性约束。在关系型数据库中，只需要在表格设计过程中，将用户名设置为唯一即可。而在 NoSQL 中，还需要通过代码来实现唯一性。本来很容易就可以实现，现在却要绕个弯去实现，这有点不可思议。由于在 NoSQL 项目中很难实现数据的完整性，而在企业应用中这个数据完整性又是少不了的。因此，我们有理由认为，NoSQL 项目很难在企业中普及开来。至少在短时间内，NoSQL 革命仍然需等待。

- **缺乏强有力的技术支持**

到目前为止，NoSQL 项目都是开源的。所以说，他们缺乏供应商技术人员提供的正式支持。在这一点上，NoSQL 项目与大多数的开源项目一样，不得不从社区中寻求支持。但是，NoSQL 项目比其他的开源项目要难得多。首先，NoSQL 项目是一个数据库系统的项目，

或者说，是一些网络应用的最基层的设施。如果其出错的话，后果很严重。由于缺乏正式的官方支持，万一数据库运行出现了错误，后果是很严重的。而且到时候用户也是投诉无门的。所以，现在 NoSQL 项目基本上还是属于研究的阶段，如果要普及到大量企业中使用，被数据库管理员所接受，至少其稳定性上要有所改善。或者说，当问题出现时，数据库管理员要能够及时修复运行故障。由于缺乏强有力的技术支持，数据库管理员担心故障出现时难以迅速解决，所以，很多管理员都拒绝使用 NoSQL 项目，即使其是开源免费的。如 NoSQL 项目的组织者 Oskarsson 也坦言，他们自己的公司现在使用的也不是 NoSQL 数据库，甚至在短期内也没有这个打算。他们现在使用的虽然是开源的数据库系统，但是，仍然是基于 SQL 的关系型数据库。

- **开源数据库从出现到被用户接受需要一个漫长的过程**

假设这个 NoSQL 技术能够被企业用户所接受，但是，从其出现到被用户最终接受需要一个漫长的过程。如 MySQL 这个开源的数据库系统，其从出现到流行也是花了好多年的时间。而且，MySQL 数据库是基于比较成熟的关系数据库模型的。其在开发设计的时候，已经有不少完善的产品可以参考。至少 SQL 语句的语法其可以直接拿来使用，而不用从零开始设计。而现在 NoSQL 是一个从零开始的产品，所有内容都需要重新设计。在没有供应商技术人员的支持下，这个过程可能是很漫长的。即使退一万步来说，最终其可以向 MySQL 数据库那样受中小企业的欢迎，但是，由于其自身技术的薄弱，在大型的数据库应用中就会显得心有余而力不足。

- **关系型数据库在设计时更能够体现实际**

其实，关系型数据库也是从非关系型数据库升级过来的。现在大部分数据库都是建立在关系型数据库模型之上的，这恰好说明了关系型数据库存在的价值。关系型数据库最大的价值就在于其设计方便。因为其数据库对象之间的关系模型(如第三范式等等)，对于数据库设计时很有帮助的，其在很大程度上体现了业务的实际情况。如在设计一个 ERP 系统时，主键与外键的关系可以反映出产品信息表与采购订单之间的关联。这种关系是那么地符合实际。而现在 NoSQL 项目想把这种关系剥离掉，那么在数据库设计的时候，必然会增加很多的麻烦，会增加数据库的难度。最重要的是，这些数据库对象之间的关系不仅仅是关系而已，其还是一种强有力的准则，对于所有的关系型数据库管理员都会产生约束。比如，Oracle 数据库的管理员经过简短的学习之后，也能够很快地掌握 SQLServer 数据库的技术，因为其内部的准则是共同的，数据库管理员只要学习其表现形式即可。这就好像学汽车，你只要拿出驾照，那么什么牌子的车都可以开。因为其数据库对象的关系、运行模式等等都是固定的。

但是，NoSQL 项目由于缺乏这种关系，导致基于 NoSQL 技术的不同产品之间可能会存在很大的差异。这不仅在数据库设计的时候会增加不少的难度。而且在维护的时候，也需要花费更多的时间与精力。

总之，照目前的情况来看，有些业界人士对 NoSQL 项目的思路仍然是反对的。至少在近期很难有像样的 NoSQL 产品面世。NoSQL 项目的组织者 Oskarsson 也承认，NoSQL 项目这场数据库革命仍然需要等待。在短时间内，无法跟关系型数据库相互抗衡，也许永远没有这个机会。

10.7 NoSQL 的三大基石

NoSQL 的三大基石包括：CAP、BASE、最终一致性。

10.7.1 CAP

2000 年，Eric Brewer 教授指出了著名的 CAP 理论，后来 Seth Gilbert 和 Nancy lynch 两人证明了 CAP 理论的正确性。所谓的 CAP 指的是：

- **C（Consistency）**：一致性；
- **A:（Availability）**：可用性(指的是快速获取数据)；
- **P（Tolerance of network Partition）**：分区容忍性(分布式) 。

CAP 理论告诉我们，一个分布式系统不可能同时满足一致性、可用性和分区容错性这三个需求，最多只能同时满足其中两个。

正所谓“鱼和熊掌不可兼得也”。如果你关注的是一致性，那么你就需要处理因为系统不可用而导致的写操作失败的情况；而如果你关注的是可用性，那么你应该知道系统的读操作可能不能精确地读取到写操作写入的最新值。因此，系统的关注点不同，相应采用的策略也是不一样的，只有真正地理解了系统的需求，才有可能利用好 CAP 理论。

CAP 理论认为，在一个系统中，对于某个数据而言不存在一个算法同时满足 Consistency、Availability 和 Partition-tolerance。 注意，这里边最重要和最容易被人忽视的是限定词“对于某个数据而言不存在一个算法”。这就是说，在一个系统中，可以对某些数据做到 CP，对另一些数据做到 AP，就算是对同一个数据，调用者可以指定不同的算法，某些算法可以做到 CP，另一些算法可以做到 AP。

当处理 CAP 的问题时，可以有几个选择，最明显的是：

- **放弃 Partition Tolerance**：如果你想避免分区（partition）问题发生，你就必须要阻止其发生。一种做法是将所有的东西（与事务相关的）都放到一台机器上；另一种做法是，放在像 rack 这类的自动失败恢复单元上，当然，仍然无法 100%地保证不发生失败，因为还是有可能部分失败，但是，你不太可能碰到由分区问题带来的负面效果。当然，这个选择会严重影响可扩展性。
- **放弃 Availability**：系统可以把这个数据只放在一个节点上，其他节点收到请求后向这个节点读或写数据，并返回结果。很显然，串行化是保证的。但是，如果报文可以任意丢失的话，接受请求的节点就可能永远不返回结果。
- **放弃 Consistency**：系统只要每次对写都返回成功，对读都返回固定的某个值就可以了。不同数据对于一致性的要求是不同的。举例来讲，用户评论对“不一致”是不敏感的，可以容忍相对较长时间的不一致，这种不一致并不会影响交易和用户体验。而产品价格数据则是非常敏感的，通常不能容忍超过 10 秒的价格不一致。对于大型网站而言，可用性与分区容忍性优先级要高于数据一致性，一般会尽量朝着 AP 的方向设计，然后，通过其它手段保证对于一致性的商务需求。架构设计师不要把精力浪费在如何设计能满足三者的完美分布式系统，而是应该进行取舍。
- **引入 BASE**：有一种架构的方法称作 BASE（Basically Available，Soft-state，Eventually consistent），支持最终一致性。BASE（注：化学中的含义是碱），如其名字所示，是 ACID（注：化学中的含义是酸）的反面。基于 BASE 实现一个满足最终一致性 (Eventually Consistency) 和 AP 的系统是可行的。现实中的一个例子是 Cassandra 系统。

10.7.2 BASE

说起来很有趣，BASE 的英文意义是碱，而 ACID 的英文含义是酸，看起来二者似乎是“水火不容”。BASE 的基本含义如下：

- **Basically Availble**：基本可用，支持分区失败；
- **Soft-state**：软状态/柔性事务，可以理解为"无连接"的，可以有一段时间不同步；而 "Hard state"是"面向连接"的；
- **Eventual Consistency**：最终一致性，也是 ACID 的最终目的，最终数据是一致的就可以了，而不是时时一致。

BASE 模型是反 ACID 模型的，完全不同于 ACID 模型，牺牲了高一致性，从而获得可用性或可靠性。BASE 思想主要强调基本的可用性，如果你需要高可用性，也就是纯粹的高性能，那么就要以一致性或容错性为牺牲，BASE 思想的方案在性能上还是有潜力可挖的。

10.7.3 最终一致性

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进程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。一致性是因为存在并发读写时才有的问题，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。

最终一致性根据更新数据后各进程访问到数据的时间和方式的不同，又可以区分为：

- **因果一致性**：如果进程 A 通知进程 B 它已更新了一个数据项，那么进程 B 的后续访问将返回更新后的值。与进程 A 无因果关系的进程 C 的访问遵守一般的最终一致性规则。
- **“读己之所写（read-your-writes）”一致性**：当进程 A 自己更新一个数据项之后，它总是访问到更新过的值，绝不会看到旧值。这是因果一致性模型的一个特例。
- **会话（Session）一致性**：这是上一个模型的实用版本，它把访问存储系统的进程放到会话的上下文中。只要会话还存在，系统就保证“读己之所写”一致性。如果由于某些失败情形令会话终止，就要建立新的会话，而且系统的保证不会延续到新的会话。
- **单调（Monotonic）读一致性**：如果进程已经看到过数据对象的某个值，那么任何后续访问都不会返回在那个值之前的值。
- **单调写一致性**：系统保证来自同一个进程的写操作顺序执行。要是系统不能保证这种程度的一致性，就非常难以编程了。

上述最终一致性的不同方式可以进行组合，例如单调读一致性和读己之所写一致性就可以组合实现。并且从实践的角度来看，这两者的组合，读取自己更新的数据，和一旦读取到

最新的版本不会再读取旧版本，对于此架构上的程序开发来说，会少很多额外的烦恼。

从服务端角度，如何尽快将更新后的数据分布到整个系统，降低达到最终一致性的时间窗口，是提高系统的可用度和用户体验非常重要的方面。对于分布式数据系统，定义如下参数：

- N — 数据复制的份数；
- W — 更新数据是需要保证写完成的节点数；
- R — 读取数据的时候需要读取的节点数；

如果 W+R>N，写的节点和读的节点重叠，则是强一致性。例如对于典型的一主一备同步复制的关系型数据库，N=2，W=2，R=1，则不管读的是主库还是备库的数据，都是一致的。

如果 W+R<=N，则是弱一致性。例如对于一主一备异步复制的关系型数据库，N=2，W=1，R=1，则如果读的是备库，就可能无法读取主库已经更新过的数据，所以是弱一致性。

对于分布式系统，为了保证高可用性，一般设置 N>=3。不同的 N、W、R 组合，是在可用性和一致性之间取一个平衡，以适应不同的应用场景。

如果 N=W，R=1，任何一个写节点失效，都会导致写失败，因此可用性会降低，但是由于数据分布的 N 个节点是同步写入的，因此可以保证强一致性。

如果 N=R，W=1，只需要一个节点写入成功即可，写性能和可用性都比较高。但是，读取其他节点的进程可能不能获取更新后的数据，因此是弱一致性。这种情况下，如果 W<(N+1)/2，并且写入的节点不重叠的话，则会存在写冲突。

10.8 NoSQL 数据库与关系数据库的比较

NoSQL 并没有一个准确的定义，但一般认为 NoSQL 数据库应当具有以下的特征：模式自由(schema-free)、支持简易备份(easy replication support)、简单的应用程序接口(simple API)、最终一致性(或者说支持 BASE 特性，不支持 ACID)、支持海量数据(Huge amount of data)。NoSQL 和关系型数据库的简单比较如表 10-1 所示。

表 10-1 NoSQL 和关系型数据库的简单比较

比较标准	RDBMS	NoSQL	备注
数据库原理	完全支持	部分支持	RDBMS 有数学模型支持、NoSQL 则没有
数据规模	大	超大	RDBMS 的性能会随着数据规模的增大而降低；NoSQL 可以通过添加更多设备以支持更大规模的数据
数据库模式	固定	灵活	使用 RDBMS 都需要定义数据库模式，NoSQL 则不用
查询效率	快	简单查询非常高效、较复杂的查询性能有所下降	RDBMS 可以通过索引，能快速地响应记录查询(point query)和范围查询(range query)；NoSQL 没有索引，虽然 NoSQL 可以使用 MapReduce 加速查询速度，仍然不如 RDBMS
一致性	强一致性	弱一致性	RDBMS 遵守 ACID 模型；NoSQL 遵守 BASE (Basically Available、soft state、Eventually consistent)模型
扩展性	一般	好	RDBMS 扩展困难；NoSQL 扩展简单
可用性	好	很好	随着数据规模的增大，RDBMS 为了保证严格的一致性，只能提供相对较弱的可用性；NoSQL 任何时候都能提供较高的可用性
标准化	是	否	RDBMS 已经标准化（SQL）；NoSQL 还没有行业标准
技术支持	高	低	RDBMS 经过几十年的发展，有很好的技术支持；NoSQL 在技术支持方面不如 RDBMS

可维护性	复杂	复杂	RDBMS 需要专门的数据库管理员(DBA)维护；NoSQL 数据库虽然没有 DBMS 复杂，也难以维护

10.9 典型的 NoSQL 数据库分类

典型的 NoSQL 数据库分类如表 10-2 所示。

表 10-2 典型的 NoSQL 数据库分类

NoSQL 数据库类型	代表性产品	性能	扩展性	灵活性	复杂性	优点	缺点
键/值数据库	Redis Riak	高	高	高	无	查询效率高	不能存储结构化信息
列式数据库	HBase Cassandra	高	高	一般	低	查询效率高	功能较少
文档数据库	CouchDB MongoDB	高	可变的	高	低	数据结构灵活	查询效率较低
图形数据库	Neo4J OrientDB	可变的	可变的	高	高	支持复杂的图算法	只支持一定的数据规模

10.10 NoSQL 数据库开源软件

10.10.1 Membase

Membase 是 NoSQL 家族的一个新的重量级的成员。Membase 是开源项目，源代码采用了 Apache2.0 的使用许可。该项目托管在 GitHub.Source tarballs 上，目前可以下载 beta 版本的 Linux 二进制包。该产品主要是由 North Scale 的 memcached 核心团队成员开发完成，其中还包括 Zynga 和 NHN 这两个主要贡献者的工程师，这两个组织都是很大的在线游戏和社区网络空间的供应商。

Membase 容易安装、操作，可以从单节点方便的扩展到集群，而且为 memcached（有

线协议的兼容性）实现了即插即用功能，在应用方面为开发者和经营者提供了一个比较低的门槛。做为缓存解决方案，Memcached 已经在不同类型的领域（特别是大容量的 Web 应用）有了广泛的使用，其中 Memcached 的部分基础代码被直接应用到了 Membase 服务器的前端。

通过兼容多种编程语言和框架，Membase 具备了很好的复用性。在安装和配置方面，Membase 提供了有效的图形化界面和编程接口，包括可配置的告警信息。

Membase 的目标是提供对外的线性扩展能力，包括为了增加集群容量，可以针对统一的节点进行复制。另外，对存储的数据进行再分配仍然是必要的。

这方面的一个有趣的特性是 NoSQL 解决方案所承诺的可预测的性能，类准确性的延迟和吞吐量。通过如下方式可以获得上面提到的特性：

- 自动将在线数据迁移到低延迟的存储介质的技术（内存，固态硬盘，磁盘）；
- 可选的写操作一一异步，同步（基于复制，持久化）；
- 反向通道再平衡（未来考虑支持）；
- 多线程低锁争用；
- 尽可能使用异步处理；
- 自动实现重复数据删除；
- 动态再平衡现有集群；
- 通过把数据复制到多个集群单元和支持快速失败转移来提供系统的高可用性。

10.10.2 MongoDB

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。他支持的数据结构非常松散，是类似 json 的 bjson 格式，因此可以存储比较复杂的数据类型。Mongo 最大的特点是他支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。它的特点是高性能、易部署、易使用，存储数据非常方便。

MongoDB 主要功能特性包括以下几个方面：

- 面向集合存储，易存储对象类型的数据：“面向集合”（Collenction-Orented），意思是数据被分组存储在数据集中，被称为一个集合（Collenction)。每个 集合在数据库中都有一个唯一的标识名，并且可以包含无限数目的文档。集合的概念类似关系

型数据库（RDBMS）里的表（table），不同的是它不需要定 义任何模式（schema)。

- 模式自由：模式自由（schema-free)，意味着对于存储在 mongodb 数据库中的文件，我们不需要知道它的任何结构定义。如果需要的话，你完全可以把不同结构的文件存储在同一个数据库里。
- 支持动态查询；
- 支持完全索引，包含内部对象；
- 支持查询；
- 支持复制和故障恢复；
- 使用高效的二进制数据存储，包括大型对象（如视频等）；
- 自动处理碎片，以支持云计算层次的扩展性；
- 支持 RUBY，PYTHON，JAVA，C++，PHP 等多种语言；
- 文件存储格式为 BSON（一种 JSON 的扩展）：BSON（Binary Serialized dOcument Format）存储形式是指：存储在集合中的文档，被存储为键-值对的形式。键用于唯一标识一个文档，为字符串类型，而值则可以是各中复杂的文件类型。
- 可通过网络访问：MongoDB 服务端可运行在 Linux、Windows 或 OS X 平台，支持 32 位和 64 位应用，默认端口为 27017。推荐运行在 64 位平台，因为 MongoDB 在 32 位模式运行时支持的最大文件尺寸为 2GB。MongoDB 把数据存储在文件中（默认路径为：/data/db），为提高效率使用内存映射文件进行管理。

10.10.3 Hypertable

Hypertable 是一个开源、高性能、可伸缩的数据库，它采用与 Google 的 Bigtable 相似的模型。在过去数年中，Google 为在 PC 集群 上运行的可伸缩计算基础设施设计建造了三个关键部分。第一个关键的基础设施是 Google File System（GFS），这是一个高可用的文件系统，提供了一个全局的命名空间。它通过跨机器（和跨机架）的文件数据复制来达到高可用性，并因此免受传统 文件存储系统无法避免的许多失败的影响，比如电源、内存和网络端口等失败。第二个基础设施是名为 Map-Reduce 的计算框架，它与 GFS 紧密协作，帮助处理收集到的海量数据。第三个基础设施是 Bigtable，它是传统数据库的替代。Bigtable 让你可以通过一些主键来组织海量数据，并实现高效的查询。Hypertable 是 Bigtable 的一个开源实现，并且根据我们的想法进行了一些改进。

10.10.4 Apache Cassandra

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。Facebook 目前在使用此系统。

Cassandra 的主要特性包括以下几个方面：

- 分布式；
- 基于列的结构化；
- 高伸展性。

Cassandra 的主要特点就是它不是一个数据库，而是由一堆数据库节点共同构成的一个分布式网络服务，对 Cassandra 的一个写操作，会被复制到其他节点上去，对 Cassandra 的读操作，也会被路由到某个节点上面去读取。对于一个 Cassandra 群集来说，扩展性能是比较简单的事情，只管在群集里面添加节点就可以了。

Cassandra 是一个混合型的非关系的数据库，类似于 Google 的 BigTable。其主要功能比 Dynomite（分布式的 Key-Value 存储系统）更丰富，但支持度却不如文档存储 MongoDB（介于关系数据库和非关系数据库之间的开源产品，是非关系数据库当中功能最丰富，最像关系数据库的。支持的数据结构非常松散，是类似 json 的 bjson 格式，因此可以存储比较复杂的数据类型。）Cassandra 最初由 Facebook 开发，后转变成了开源项目。它是一个网络社交云计算方面理想的数据库。以 Amazon 专有的完全分布式的 Dynamo 为基础，结合了 Google BigTable 基于列族（Column Family）的数据模型。P2P 去中心化的存储。很多方面都可以称之为 Dynamo 2.0。

和其他数据库比较，Cassandra 的突出特点是：

- **模式灵活**：使用 Cassandra，像文档存储，你不必提前解决记录中的字段。你可以在系统运行时随意的添加或移除字段。这是一个惊人的效率提升，特别是在大型部署上。
- **真正的可扩展性**：Cassandra 是纯粹意义上的水平扩展。为给集群添加更多容量，可以指向另一台电脑。你不必重启任何进程，改变应用查询，或手动迁移任何数据。
- **多数据中心识别**：你可以调整你的节点布局来避免某一个数据中心起火，一个备用的数据中心将至少有每条记录的完全复制。
- **范围查询**：如果你不喜欢全部的键值查询，则可以设置键的范围来查询。
- **列表数据结构**：在混合模式可以将超级列添加到 5 维。对于每个用户的索引，这是

非常方便的。

- **分布式写操作**：有可以在任何地方任何时间集中读或写任何数据。并且不会有任何单点失败。

本章小结

本章首先介绍了 NoSQL 概念以及发展现状，阐述了为什么要使用 NoSQL 数据库以及 NoSQL 数据库的特点；然后，介绍了 NoSQL 数据库的五大挑战及其面临的质疑；接下来，介绍了 NoSQL 的三大基石，即 CAP、BASE 和最终一致性；接下来对 NoSQL 数据库和关系数据库进行了简单比较；最后，给出了典型的 NoSQL 数据库分类和开源软件。

参考文献

[1] 颜开. NoSQL 数据库笔谈. 百度文库. http://wenku.baidu.com/view/246e00d4195f312b3169a570.html

[2] CAP 原理. 百度文库. http://wenku.baidu.com/view/7f25f00d7cd184254b353530.html

[3] 孟小峰, 慈祥. 大数据管理：概念、技术与挑战. 计算机学报, 2013 年第 8 期.

附录 1:任课教师介绍

林子雨(1978－),男,博士,厦门大学计算机科学系助理教授,主要研究领域为数据库,数据仓库,数据挖掘.

主讲课程：《大数据技术基础》

办公地点：厦门大学海韵园科研 2 号楼

E-mail: ziyulin@xmu.edu.cn

个人网页：http://www.cs.xmu.edu.cn/linziyu