几款主流的 NoSQL 数据库介绍

HBase

HBase 是 Apache Hadoop 中的一个子项目，属于 bigtable 的开源版本，实现的语言为 Java，依托于 Hadoop 的 HDFS（分布式文件系统）作为最基本存储基础单元。

优点

存储容量大，一个表可以容纳上亿行，上百万列
可通过版本进行检索，能搜到所需的历史版本数据
负载高时，可通过简单的添加机器来实现水平切分扩展，跟 Hadoop 的无缝集成保障了其数据可靠性（HDFS）和海量数据分析的高性能(MapReduce)
在第 3 点的基础上可有效避免单点故障的发生。
缺点

基于 Java 语言实现及 Hadoop 架构意味着其 API 更适用于 Java 项目
node 开发环境下所需依赖项较多、配置麻烦（或不知如何配置，如持久化配置），缺乏文档
占用内存很大，且鉴于建立在为批量分析而优化的 HDFS 上，导致读取性能不高。
使用场景

bigtable 类型的数据存储
对数据有版本查询需求
应对超大数据量要求扩展简单的需求。
使用许可： Apache

Redis

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。

优点

数据结构丰富
Redis 提供了事务的功能，可以保证命令的原子性，中间不会被任何操作打断
数据存在内存中，读写非常快，可以达到 10w/s 的频率。
缺点

持久化功能体验不佳，通过快照方法实现的话，需要每隔一段时间将整个数据库的数据写到磁盘上，代价非常高
由于是内存数据库，存储的数据量跟机器本身的内存大小有关。虽然 redis 本身有 key 过期策略，但还是需要提前预估和节约内存。如果内存增长过快，需要定期删除数据。
使用场景

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。
使用许可： BSD

MongoDB

MongoDB 是一个高性能，开源，无模式的文档型数据库，开发语言是 C++。

优点

强大的自动化 shading 功能
全索引支持，查询非常高效
面向文档（BSON）存储，数据模式简单而强大
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
支持 javascript 表达式查询，可在服务器端执行任意的 javascript 函数。
缺点

对内存要求比较大，至少要保证热数据都能装进内存，例如：索引等
非事务机制，无法保证事件的原子性。
使用场景

适用于实时的插入、更新与查询的需求，并具备应用程序实时数据存储所需的复制及高度伸缩性
非常适合文档化格式的存储及查询
高伸缩性的场景：非常适合由数十或者数百台服务器组成的数据库
对性能的关注超过对功能的要求。
使用许可： AGPL（发起者： Apache）

Couchbase

CouchDB，Membase 合并后的杰作，合并之后的公司基于 Membase 与 CouchDB 开发的新产品。Couchbase 可以说是集合众家之长，目前应该是最先进的 Cache 系统，其开发语言是 C/C++。

优点

高并发性，高灵活性，高拓展性，容错性好
以 vBucket 的概念实现更理想化的自动分片以及动态扩容。
缺点

Couchbase 的存储方式为 Key/Value，但 Value 的类型很为单一，不支持数组。另外也不会自动创建 doc id，需要为每一文档指定一个用于存储的 Document Indentifer
各种组件拼接而成，都是 c++实现，复杂度过高，（中文）文档比较欠缺
采用缓存全部 key 的策略，需要大量内存。节点宕机时 failover 过程有不可用时间，并且有部分数据丢失的可能，在高负载系统上有假死现象

免费版和商业版本之间差距比较大。
使用场景

适合对读写速度要求较高，但服务器负荷和内存花销可遇见的需求
需要支持 memcached 协议的需求。
使用许可： Apache

LevelDB

LevelDB 是由谷歌重量级工程师（Jeff Dean 和 Sanjay Ghemawat）开发的开源项目，它是能处理十亿级别规模 key-value 型数据持久性存储的程序库，开发语言是 C++。除了持久性存储，还有一个特点是写性能远高于读性能。

优点

操作接口简单，基本操作包括写记录，读记录和删除记录，也支持针对多条操作的原子批量操作
写入性能远强于读取性能
数据量增大后，读写性能下降趋平缓。
缺点

随机读性能一般
对分布式事务的支持还不成熟，而且机器资源浪费率高。
使用场景

适用于对写入需求远大于读取需求的场景
使用许可： MIT

Neo4j

1、带标签的属性图由节点、关系、属性和标签组成
2、节点上包含属性。
3、节点可以被打上一个或多个标签，标签把节点组织在一起，并表示它们在这个数据集中的角色。
4、关系连接节点，从而构成图。每条关系都有一个方向、一个名字（type）、一个开始节点和一个结束节点。
5、关系也可以有属性，可以增加额外的语义（包括特性和权重），还可以用于运行时的约束查询。