## Elasticsearch与hbase

HBase支持简单的行或者range查询，比如给一个PK查该行的数据，或者给一个begin/end查这个范围的数据，如果想完成更复杂的功能就不太容易。而ES支持的查询比较丰富，或者说这些查询都带有一点复杂计算的味道了。比如你有个论坛，你想查帖子里面是否包含敏感词，如果采用HBase就比较麻烦，使用HBase你可以将帖子存进来、读出去，但是要查内容里面的东西，只能一点点过滤；而ES是可以比较方便的帮助你完成这个功能的；**所以，如果是规则匹配或者简单的统计计算数据的话，es更合适。**

**ES作为主要存储需要考虑的内容：**

**硬件资源：**

ES是基于lucene开发的，它的许多局限性从根本上都是由lucene引进的。例如，为了提高性能，lucene会将同一个term重复地index到各种不同的数据结构中，以支持不同目的的搜索，基于选用的分析器，最终index数倍于原本的数据大小是有可能的。内存方面，es的排序和聚合（aggregation）操作会把几乎所有相关不相关的文档都加载到内存中，一个复杂的query就可能吃光所有内存，现在新的lucene版本优化了基于硬盘的排序，但也仅当你试用SSD的情况下，才不会牺牲过多的搜索性能。Es在横向规模方面做的很出色，但前提是有足够的预算买硬件。

**可靠性**：

OutOfMemory，当存储的数据量达到一定的规模之后，一个复杂的搜索请求，可能就会导致OutOfMemory。

**安全性**：

Es的安全相关的功能单独做成了一个shield收费插件。

**权限管理**：

没有细致的权限管理机制，也就是说，没有像mysql那样的分多种用户，每个用户又有不同的权限。所以在操作上的限制需要自己开发一个系统来完成。

**脑裂问题**

**无事务**

Lucene, 是 Elasticsearch 的构建的基础, 它是由一个事务的概念的. 而Elasticsearch在另外的方面, 并没有典型意义的事务. 对于已经提交的文档并没有办法回滚, 而你也不能提交一组文档并且为它们所有或者其中一些建立索引. 然而它所具备的, 是一个用来确保业务过程持久性而不用做昂贵的Lucene提交的预写日志. 你也可以指定索引操作的一致性级别, 以确保在返回之前有多少副本可以拿来确认操作条件. 默认的是法定人数, 例如 ⌊n2⌋

在逐个切片进行处理的方式中，当一个索引被刷新时，默认是一秒钟一次，就需要对变更的可见性进行控制。

通过制定提交文档的版本，可以进行乐观并发控制。

Elasticsearch追求的是速度。支持分布式事务是一大块工作。不支持分布式事务会使得很多事情变得容易起来。只要我们能接受读取到的数据有些陈旧，而且所有人看到的是同一时间点的数据，那么Elasticsearch就可以利用缓存提供很多服务 - 这对于我们钟爱的极速性能来说是至关重要的。

**模式灵活**

Elasticsearch 不要求你先指定模式。扔给它一个 JSON 文档，它就会进行一些训练有素的猜测来推断其类型。对于数值、布尔、时间戳它可以做的很好。对于字符串，它会使用“标准化”的分析，这通常是良好的开始。

它是有商榷的“无模式”，在这个意义上你不必指定一个模式，我们更愿意把它认做是“模式灵活”。为了开发大规模的搜索、分析，你确实需要对模式进行微调。Elasticsearch 有大量的强大工具可以帮助你，例如动态模板、多字段对象等。这在我们关于映射的文章里会谈及更多。

**关系和约束**

Elasticsearch是一种面向文档的数据库。你想要对之进行搜索的整个对象关系图，都需要进行索引，在对文档进行索引之前，它们必须先被反规范化。反规范化提升了查询性能(因为不再需要进行关联查询)，使用了更多存储空间(因为数据必须被存储多次)，但是，要保持数据一致性和实时性则更加困难(因为任何数据改变都必须被写入到所有实例中去)。不过，对于一次写入频繁读取的工作场景，它的表现相当优异。

举例来说，假设你在数据库中存储了客户、订单和产品等数据，现在你想要通过产品名字和客户姓名来查找订单。可以这样来解决这个问题：在对订单进行索引时，把客户和产品的所有必要信息都加进来。这样的话，查询就非常简单，但是当你想要改变某个产品的名字时会出现什么情况呢? 在进行了良好规范化的关系型模型中，你只需要修改该产品对应的单条记录就搞定了。这是关系型数据库所擅长的。而在反规范化的文档数据库中，将不得不更新与该产品有关的所有订单。

换句话说，在面向文档类型的数据库中，比如Elasticsearch, 我们对文档进行映射和存储设计只是为了优化查询和信息获取的性能。

在介绍中已经提到，Elasticsearch中可以使用父/子-关系进行“查询时”连结，也可以使用内嵌类型进行“索引时”连结。我们会在以后的文章中对该主题进行深入介绍。我们推荐Martijn van Groningen的一篇文章“Document relations with Elasticsearch”.

大多数关系型数据库也会允许你指定约束关系，来定义什么需要保持一致性，什么不需要保持一致性。比如，参照完整性和唯一性都是强制性的。你可以要求账户变更金额必须是正数，等等。而面向文档的数据库不倾向于这么做，Elasticsearch就是如此。

**鲁棒性**

一个数据库应当是鲁棒的，尤其是当它是你权威的记录系统时。理想情况下，一个耗费资源的查询应当可以被撤销，你肯定不希望数据库停止工作，除非你命令它停下来。

不幸的是，Elasticsearch (以及它的组件)目前并没有很好的处理OutOfMemory错误。我们在Elasticsearch in Production, OutOfMemory-Caused Crashes一文中进行了更加深入的讨论。所以，给Elasticsearch配置足够多的内存就显得非常重要，而且在生产集群中要谨慎运行那些无法预知将会耗费多少内存的查询。

不过，随着Elasticsearch越来越成熟，这个问题很可能会得到改善，记住Elasticsearch的目标是追求速度，所以它假设内存永远是充足的。

**分布式**

分布式系统天生的特性，意味着很多事情都可能会出错。同样的，不同的数据库系统致力于拥有不同强项: 有些力争强大的安全保证，另外一些则致力于高可用性，即使某些时候（甚至多数时候）出现错误也在所不惜。而且，正如Kyle Kingsbury在他杰出的网络分区的风险系列文章中所指出的那样，当问题发生时，实际上数据库系统并没有像它们所宣称的那样来处理问题。简言之，他发现当分布式数据库运行良好晴空万里时，绝大部分数据库在遭受到大量可能的问题时所做的努力，都会以失败而告终。

在一致性，可用性，以及分区容错性方面，Elasticsearch 是一个CP系统，一个相当不靠谱的“一致性”定义。如果你有一个只读的工作负载， Elasticsearch 允许你通过不严格的“最小主节点”要求（如不需要quorum）实现AP行为。然而，通常你需要集群中大量节点是可用的。写入一个配置错误的没有大量节点的集群，像“split brain”集群，可能引起无法恢复的数据丢失。这不只是特定于Elasticsearch的。

Elasticsearch 选择自己的主节点。然而这是一个相当简单但不是非常可靠的算法，这在现实世界的网络压力下可能导致很多问题。事实上，我们管理成百上千的集群，经常看到主节点选择问题，因此我们主动将主节点移入ZooKeeper， 当然我们也有其他目的。

在拓展性方面，索引被分到了一个或多个碎片中。这在索引创建时就指定了并且不能改变。因此，随着预期的增长，一个索引应该被合理的分割。随着越来越多的节点添加到Elasticsearch集群中来，它在重新分配和移动碎片上表现良好。因此，Elasticsearch很容易拓展。

**安全**

Elasticsearch没有提供授权和认证特性。你可以认为，任何人只要能连接到你的Elasticsearch集群，就拥有了“超级用户”权限，尤其是当Elasticsearch的强大脚本功能被激活时。

Elasticsearch通常被用作其它数据库的补充。那样的数据库系统要有强大的数据约束保证、容错性和鲁棒性、高可用性和带事务支持的数据更新能力，它维护着核心数据 - 这些数据随后会被异步推送到Elasticsearch中去(也可能是抽取，前提是你使用了Elasticsearch的某一种“rivers”)。 保持数据同步是我们打算将来撰文深入探讨的话题。这里，在Found网站的文章中，我们使用具有代表性的PostgreSQL和ZooKeeper来提供源数据，并把这些数据输入给Elasticsearch来提供强大的搜索。

Es不适合做为数据存储的理由：

1. **mapping不可改，不能改index属性。**ES中以定义的mapping不能修改名字和属性，无法修改名字勉强还能接受，但无法修改属性。官方文档中介绍了几种修改mapping的方法。一个是新建一个字段，程序中所有地方修改名字，这对于复杂的项目容易出错，而且无法保留原来的数据；另一个是利用alias创建一个新的索引，但是所有数据需要重新导入，这需要很长时间，操作性不强。
2. **无法多对多。**ES中提供3种关联关系：Field collapsing（严格来说不是关联），Nested object，Parent-child。前两种都是直接将一个mapping声明在另一个mapping中，第三种关联是在创建子文档是指明他的父文档，但是一个子文档只能有一个父文档，因此也不能实现多对多的关联。其实如果理解了ES的目的是提升检索效率，就不难理解为什么没有多对多关联了，在关系数据库里这就是个效率瓶颈。
3. **没有用户验证和权限控制。**ES本身的访问权限可以通过nginx进行控制，但是同一个ES中不同索引间目前是没有权限控制的。
4. **从ES设计的初衷看，为了检索，为了统计。**这个从字段的store属性中可以看出来，查看ES手册（https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-store.html）可以发现，默认情况下字段的原始值是不会被保存的，这跟数据存储是南辕北辙了。
5. **项目开始时不好确定shards数量。**少了以后扩展不方便，多了一开始影响性能。这个可以通过将type命名为doctype-yyyymmdd来解决，每天都生成新的一个或多个shard，但是注意在搜索时需要在doctype-\*中搜索。
6. **ES非常适合特定的需求，但不适合用于数据存储。**ES索引速度快，扩展方便，性能优异，但在功能上不适合作为数据库使用。数据存储的目的是为了以后能方便的使用，不仅是针对当前的需求，也要为未来可能出现的需求做准备。由于ES有以上几点问题，无法适应需求变化。

**Es适合的场景：**

1. 检索。ES本身作为一个搜索引擎，用来处理检索的任务再合适不过。你可以在线上项目中直接将内容写入ES以提供检索服务，也可以把以往的数据导入ES以处理特定的需求。
2. 统计。ES的统计也是基于检索功能的，聚合功能使得统计结果处理起来非常方便。如果你只需要统计而不用检索，可能有其他工具更适合你，比如Spark SQL。

Es的缺陷：硬件资源要求高、可靠性（及OutOfMemory的问题）、安全性

建议的方案：在另外的数据库里存一份，然后实时同步到es，一个用于键值查询，一个用于其他查询。如果es升级了或者数据结构变了，可以不要老版本数据，直接nosql再导入一份到新版本，还可以灰度

同时，也有一个说法是如果数据并不频繁的update操作，也没有事务性操作，es完全可以替代nosql。

Es的优点：

高并发、容错能力比mongodb强（比如一主多从，主片挂了从片会自动顶上）

易扩展（实例间做下配置即可扩展并发性和容积，自动分配的写入机制，无需操心传统db中多主同步的诟病）、支持较复杂的条件查询（如group by、排序等）、具有一定的关系性但不用担心大字段的问题（什么意思）

缺点：

不支持事务、无权限管理（es5.x好像是有了）

Es的写入速度跟**磁盘**io有关，用ssd明显会高点，跟副本数量有关，跟内存核cpu没太大关系。聚合查询是消耗cpu内存的地方。

对于读多写少的存储es是可以代替mongodb的，特别是运行在SSD服务器上时；

对于复杂查询，mongo能做的es也能做到；扩展性方面，都很优秀靠谱，es维护起来要稍微方便一些；

性能上来说，读方面mongodb要比es稍强（都足够快，一天千万级pv的单服两者都可以轻松顶住，es的cpu占用会比mongo稍多），写方面mongo秒杀es，特别是mongo3以后wiredtiger的引入，写性能差距更大，所以满足读多写少的情况的时候，才考虑用es代替mongo；

存储灵活性方面mongo完胜es，用es做主存要想清楚：es的mapping是不可变的，所以当需求经常变的时候最好用mongo，当数据量多的需求变化很痛苦（如几亿的数据慢慢scan然后全部重建的痛苦）

Es多出个全文索引功能，当然mongo3也有，但是和es比起来是个玩笑。

Es的操作粒度太粗了

一般采集的数据是需要经过各种处理，才会去搜索，这样的话，仅仅一个es就够了，最好先把数据存储到一个nosql数据库中，然后对数据做处理，最后用es建索引去搜索。

es完全胜任mongodb能做的事情，而且还加上了检索功能，可以选择分词检索或者把要存的整个文档当做一个词（不是字段么），前者类似于搜索引擎，后者类似于数据库，而且es最擅长的就是用facet和agg做数据统计，当不分词时，可以结合redis等把词条映射为整形数，查询效率会非常高。而且数据分区更灵活，可以随时以编码的方式打开或者关闭部分数据节点，一般来说，把es以数据库的模式存储，合理使用查询语法，都可以秒级返回，不管多大的数据量，当然做统计肯定会慢一些。对某些特殊查询注意一下就行：比如用wildcard（通配符）的\*keyword模式就比keyword\*模式要慢很多，需要合理规划自己的业务场景和数据的mapping映射方式。

关键词搜索加地理位置以及条件过滤，es是可以实现的。

如果要做实时、动态的计数，则推荐ES。  
如果要跑些月报表什么的，则推荐Hbase。

业务偏复杂的话，推荐es给数据建索引，原始内容存hbase。

Hbase存多读少，不适合高并发查询，适合存数据；es是全文检索，适合日志分析日志统计之类。

**像其它所有事情一样，没有包治百病的灵丹妙药，没有一个数据库可以做到所有的事情。也许这种情况会永远持续下去，所以一定要了解你的数据库的优点和弱点!**

如果对可靠性要求不高，es是可以作为主要的存储的，当然，更推荐的做法是将其建立在其他的数据库之上，专门做它擅长的数据分析工作。

根据对一些文档的参考，es对hbase建索引的方法是：对可能要用到字符匹配以及计算的列（字段）进行索引，提高字符匹配效率，同时，聚合需要用的字段，通过相应的doc\_value和fileddata，速度不会减少，其余字段将其内容禁用即可。

**如果数据大部分是结构化数据，那么在es中的mapping设置也是不分词的，对内存要求相对而言降低不少。**

**总体来看，DocValues只比内存fielddata慢大概10~25%，稳定性则有了大幅度提升** （https://yq.aliyun.com/articles/6902）

**所以，es的mapping做好就行**

Es与nosql数据库：  
 当你的数据本身是关系型数据时，es会把你搞死（<https://www.zhihu.com/question/45510463>）

数据偏多于结构化数据也没什么，因为可以在上面部署hive，完全满足结构化的需求。

## 参考文档

<https://www.zhihu.com/question/22468244>

<https://www.zhihu.com/question/40961734>

<http://www.07net01.com/2015/04/830042.html>

<http://database.51cto.com/art/201402/430440.htm>

<https://www.zhihu.com/question/25535889>

<https://www.zhihu.com/question/45510463>

待定的文档：

NoSQL 比较 - Cassandra vs MongoDB vs Redis vs ElasticSearch vs HBase：（英文待翻译）

<http://www.fx114.net/qa-30-177555.aspx>

用Elasticsearch构建电商搜索平台，一个极有代表性的基础技术架构和算法实践案例：

<http://www.36dsj.com/archives/61886>

es+hbase实现海量数据秒回查询：

<http://www.07net01.com/2017/01/1766047.html>

elk教程（重点关注es聚合内容）：

<http://www.cnblogs.com/xing901022/p/4704319.html>

Es的docvalue的内容：

表格存储：<https://yq.aliyun.com/articles/55959>

Docvalue：<https://yq.aliyun.com/articles/6902>

Hbase存数据，es建索引：<http://www.cnblogs.com/bonelee/p/6432732.html>

Es建索引的一个实例：<http://blog.csdn.net/xiaoyancj/article/details/52667754>