GFS: Google file system

<https://storage.googleapis.com/pub-tools-public-publication-data/pdf/035fc972c796d33122033a0614bc94cff1527999.pdf>

重点：revisit。失效是常态-内置监控、冗余、恢复；大文件-小文件支持但不优化；追加>>复写，流式读>>随机读-优化&原子性主要因素；与应用协同设计-放松一致性，必须提供多路原子追加；吞吐量>>低延迟。每集群单master多chunk节点，master 管map(file->chunkLocation)，client先找master然后找chunk要数据。master还监控chunk health并manage，访问控制，回收，迁移。chunk等大，有redundancy。客户端不考虑缓存因为大多要么流要么存不下，这也简化了设计；chunk node不考虑因为本地fs自带缓存。用缓存和整合request减少客户端和master的通信。

HDFS: Hadoop distributed file system

<http://www.alexanderpokluda.ca/coursework/cs848/CS848%20Paper%20Presentation%20-%20Alexander%20Pokluda.pdf>

<https://svn.eu.apache.org/repos/asf/hadoop/common/tags/release-0.16.3/docs/hdfs_design.pdf>

CAP: 一般P必选，AC二选一。

原始猜想：<http://awoc.wolski.fi/dlib/big-data/Brewer_podc_keynote_2000.pdf>

论证：<https://courses.e-ce.uth.gr/CE623/CAP_theorem_proof.pdf>

新综述：<https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed/>

不错的文章：<https://www.cnblogs.com/xybaby/p/6871764.html>

Consistent hashing

<https://blog.csdn.net/cywosp/article/details/23397179>

DynamoDB

论文：<https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf>

关注A，在某些故障场景中牺牲C，eventual consistency。无跨数据项操作，无关系。不提供Isolation，只保证单键原子操作。读负责解决冲突，写不管，以此保证写A。Consistent Hashing，虚节点解决主机能力不同问题。W+R>N提供无failure时的consistency，用户自行平衡写效率与读效率。即使有上述保证，failure情况下也会出现冲突。数据每个副本附带Vector clock（某种版本写历史），如有冲突，系统自行尝试根据历史解决，否则返回全部供用户自行解决。GFD，去中心。数据在协调者和其后N-1个满足redundancy的节点（称为Preference List）上备份。路由没用Chord，每个节点存全路由表，用Gossip协议扩散，一般0跳。如想用Hash Key / Range Key之外的东西查询，可创建索引。用Merkel tree同步。冲突数据不会丢失，但删除数据可能重新出现。

批评：<http://jsensarma.com/blog/?p=55>

Chubby

Chubby高可靠分布式锁(coordination)服务。

<https://amplab.github.io/cs262a-fall2016/notes/22-Chubby.pdf>

[https://static.googleusercontent.com/media/research.google.com/en//archive/chubby-osdi06.pdf](https://static.googleusercontent.com/media/research.google.com/en/archive/chubby-osdi06.pdf)

Bloom filter

<https://zh.wikipedia.org/wiki/%E5%B8%83%E9%9A%86%E8%BF%87%E6%BB%A4%E5%99%A8>

BigTable

论文：[https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf](https://static.googleusercontent.com/media/research.google.com/en/archive/bigtable-osdi06.pdf)

白皮书：<http://blog.bizcloudsoft.com/wp-content/uploads/Google-Bigtable%E4%B8%AD%E6%96%87%E7%89%88_1.0.pdf>

row, columnFamily:column -> string timestamp, schema-free。按row字典序排序并分片，按row原子操作。节点一主多从(tablet server)，主管分配片，增删从，balancing；从管片的读写合拆，无物理数据，代理GFS。片逻辑上组织为B+树，Chubby -> root tablet (metadata tablet 0) -> (other) metadata tablets -> data tablets。无论metadata tablet还是data tablet都在从上。client通过缓存并直接与所需tablet服务器相连以减轻元数据片load。写时先append log然后插入memtable，读时在memtable和SSTable上合并查找。memtable在从，SSTable在GFS。memtable变大会冻结转换为SSTable并写入GFS，降低内存并加快snapshot-log recovery，称minor compaction。SSTable会越来越多，周期性将memtable和多个SSTable归并为新SSTable，顺便清除标记未删除的信息，称major compaction。删除实际上是新增记录标记为删除，通过获取最新记录判断是否被删除。可以指定locality group供程序调优。SSTable静态，可用Bloom filter过滤读取以加速。用Chubby管理host health。tablet = memtable U all SSTable

上述未包含replication与redundancy，见：<https://cloud.google.com/bigtable/docs/replication-overview>

ZooKeeper

Chubby的开源实现。一般3台，支持半数以下宕机，越多越慢。内部Paxos。

<https://www.usenix.org/legacy/event/usenix10/tech/full_papers/Hunt.pdf>

HBase

Blog：[https://mapr.com/blog/in-depth-look-hbase-architecture/#.VdMxvWSqqko](https://mapr.com/blog/in-depth-look-hbase-architecture/" \l ".VdMxvWSqqko)

BigTable的开源实现：Chubby->ZooKeeper，GFS->HDFS，换术语。没有GFS，那HDFS数据尽可能本地化。一般一个HMaster热备份，用ZooKeeper的master election。初次访问流程少一级（又不是都是Google那么大，换点效率挺好的）。+blockcache优化，读blockcache -> memstore -> HFile。持久化文件按row, column family, column, timestamp倒序排序。Split在同一台机器上，随后tablet可能被balance到别处。此时新tablet远程用HDFS，直到major compaction数据本地化。HDFS提供replication。优点consistency，built-in recovery；缺点recovery慢，major compaction慢。1 memstore / column family。HFile组织为B+ tree。数据模型table-row-(column family)-column.

Cassandra

论文：<http://www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf>

node以上DynamoDB (partitioning, consistency...) + node以下BigTable (memtable, SSTable...) + 某些变化。提供data基本类型及json且可动态扩展。写性能高。client任意连接node可中转到possessor。row主键= partition key + (prefix)。相比HBase更强调A弱化C。ScyllaDB C++版Cassandra。Generally inconsistent read性能 > HBase，consistent read则弱，毕竟HBase天生consistent。数据模型column family (相当于HBase中不使用column family的table)-row-column或super column family (相当于HBase中使用column family的table)-row-super column-column。

Redis

MongoDB

<http://horicky.blogspot.com/2012/04/mongodb-architecture.html>

document数据库，document操作atomic，支持atomic conditional modify。多副本，可变一致性。支持BTree index，根据文档内容构建索引快速定位到文档，推荐多副本轮流下线建索引。多查询条件时挑一个index用然后对结果遍历filter。index会拖慢操作。Collection Namespace下辖按2^n大小组织的free disk块，使用中的块组织为双向链表称Extent，每个Extent含多个Doc组织为双向链表。Doc中含padding减少数据移动，碎片增多需要轮流下线compact。Primary crash时leader election但可能丢数据。特殊slave replica keep master一段时间前的状态，应对误删。特殊arbiter只投票不存数据辅助election。读按策略挑一个无quorum。规模变大多服务器拆分：定义partition key，分成chunk，chunk的replica分给Shard server，config server保留map(key range -> chunk), map(chunk, shard)，Route server中介Client到Shard，Client的请求能根据partition key定位优化则优化，否则map给全部再reduce。chunk自动切分，replica自动转移以balance。Route server又称Mongos，无状态，元数据给Config server，用户数据给Shard。顺序分片则当Key操作有明显单调趋势时不利，因为写操作基本都集中在最高chunk，也容易造成huge chunk无法分裂；Hash分片查询不利因为全都要map reduce。

概论：速度：无结构数据>半结构数据>结构化数据；feature：无结构数据<半结构数据<结构化数据。MongoDB作为半结构数据代表非常典型，提供了RDBMS最重要的一些功能如Query，这是它流行的主要原因。

ElasticSearch

Doc：<https://doc.yonyoucloud.com/doc/mastering-elasticsearch/index.html>

倒排索引，搜索引擎/文档数据库，准实时搜索，RESTful。Index(~database) - Type(~table) - Document(~record) - Field(~column)。Type有Mapping(~schema)，相当于可自动建立的可动态增加但不可修改的schema用来优化查询，因此检索效率高，这是此类DB的[核心特征](https://db-engines.com/en/article/Search+Engines)。Shard分片，primary读写 - replica读，故障选举。Shard数量不可变（官方称切分和重新索引成本差不多），replica数量可变，分片基于某值哈希。聚合分析和搜索都很强，支持Geolocation，不支持但可hack关系数据库的关联查询。分data node和transport node，后者分发request-聚合result，可混合部署可分。用gateway中介持久化方式即文件系统，可用分布式文件系统拆分计算和存储。集群中有master节点负责状态管理和分片管理，选举出来的所以没有单点故障。是搜索引擎但在index中保存原始数据并可获取因此可当数据库。按功能和优先级创建连接和线程池。应用：站内搜索、document数据库、时间序数据的存储分析可视化（领先，Elastic stack）。

可能和MongoDB趋同演化。相比之下写慢，耗硬件，必须SSD，半schema，换来search强读快。

Spanner

论文：<http://delivery.acm.org/10.1145/2500000/2491245/a8-corbett.pdf?ip=54.240.196.170&id=2491245&acc=OA&key=4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E3ABADC0B30E26CFA&__acm__=1558493477_fc03f3f7ad3a35bd1f36e736629a4ee6>

杂项

如果只有一个备份，Optimistic lock就能解决一致性了，上一版本变化则写失败。

多数数据库写时先append log然后做各种操作，因为append持久化快，后者慢，期间更可能因故障丢失。

广泛应用的MurmerHash：<https://en.wikipedia.org/wiki/MurmurHash>

MySQL用得最多，PostgreSQL最先进。MySQL被Oracle收购了不看好，MariaDB是上升中的替代品。

book club

DDIA: Designing Data-Intensive Applications

什么是流处理（stream processing）？批处理（batch processing）？

Apache Kafka

可靠性包括防止未经授权的访问和滥用。软件错误：级联，依赖，资源，特定输入。人为错误：难犯错，犯错decouple，回滚。

写时多干活vs读时多干活

尾部延迟重要，很可能短板决定总体（尾部延迟放大），而且很可能对应最重要客户

排队延迟-头部阻塞：前面慢，大家慢

少量大机器可能比大量小机器更便宜

早期快速迭代比可扩展到假想负载更重要

stateful - distribute conflict

Sable assumptions and requirements? Requirements for Procyon?

简单性：反对额外复杂度：由具体实现中涌现，而非（从用户视角看，系统所解决的）问题本身固有的复杂度。

TLA language? Paper describing modeling system state through formal language and validate state transition in it. Problems: 1. Don't assume it IS the proof. 2. Depends on the correctness of this tool. 3. How do you know your model is correct. Use it as a supplement, not a single tool.

valuable paper:

big mud ball

the tail at scale

doccer, technical writers and good document is necessary in the sense that keeping consistent design helps you to reduce complexity coming from development.

Easy to use is not necessarily good because it gives people WRONG INCENTIVE to do.

slo vs sla

每层通过数据模型隐藏更低层次的复杂性。

阻抗不匹配：对SQL的批评。程序代码和数据库的数据模型不连贯，转换层笨拙。

就一对多关系/树状结构而言，文档模型优于关系模型，受益于局部性可能更快。关系模型使用外键处理，局部性差。

规范化/副本问题。ID对人类没有任何意义因此不需要改变；任何对人类有意义的东西都可能需要改变。冗余副本有更新开销问题和不一致风险。

文档多对一/多对多关系弱，连接支持差，连接需要的工作从数据库转移到应用程序且一般更慢。

网络模型：层次模型的推广，允许多个父节点，沿路径/搜索获取数据，复杂性交给应用层且不便更改，关系模型中此复杂性被包装。

文档数据库严格说是读时模式（隐含模式，读时解释，数据库不强制）；关系数据库为写时模式，模式迁移涉及alter和update，麻烦且慢要停机。

文档数据库建议小文档（取大文档用一小块太浪费），不变大小。

趋同演化：关系数据库试图利用局部性，提供文档支持，文档数据库试图优化连接。

声明式语言>>命令式语言：优秀的抽象，包装复杂性，提供自由度，命令式语言往往和底层couple且难以并行化。MapReduce介于二者之间，提供了一定的并行化/自由度/声明式，可能成为文档数据库版SQL，如MongoDB的聚合管道。

多对多关系最适用图模型。顺便PageRank是图算法。属性图模型/三元组模型，有声明式查询语言。灵活性好，能处理不同地区结构、数据粒度等（如出生在省住在市）。Neo4j-Cypher语言。关系数据库存图用SQL查也行但是笨，比如递归公用表表达式（数据模型之间可以模拟但不match好不了）。三元组模型对应声明式查询语言SPARQL，Datalog rule based分块构造查询可重用（语义网络和RDF的失败不妨碍三元组和相关语言本身的优点，本质上二者独立）。

图数据库vs网络模型：网络代表CODASYL有模式，必须通过路径访问无法用ID直接到达，命令式查询。

Datapath是不是基于Sable提供声明式查询语言？

数据仓库针对查询和聚合优化，常见列式存储

Hive

Teradata

Question:

Amazing, GFS and BigTable

SSD

Why "not because keep in memory" [44]? Is it because your file system view whatever data structure you input as byte stream and write in whatever format it uses?

How about Procyon?

Why Google BigTable didn't become popular? HBase is HIGH

Google provided the idea, Apache provides it free and become popular, then Google realize they should sell it.

What is BDB? You can choose between hash and BTree

write ahead log for things other than rollback? SQL lite use this: performance optimization, interesting feature.

DoryDB by Ryan in Amazon code, WOW

Redshift is a modified version of PostgreSQL

Datapath is like "materialized view" on Sable, seems like it provides 2 things: declarative query language and materialized view?

Yuval from IBM: compression may not cause much efficiency penalty because you can parallel and disk is much slower than ram, you can use this difference for de-compression.

Google spanner use atomic clock to promise consistency

Coral is trying to provide RPC?

What is the real difference between REST and RPC?

Questions:

Difference between multi master and no master? Client send to multiple or one?

Dynamo != DynamoDB?

Does sloppy quorum worth it?

Cassandra n = sum data center?

Any drawback of vector clock? Efficiency, complexity, value explode?

How does Sable handle order like publish after update?

DDB has metadata server.

Spanner: Trade off inside CAP, not outside. Use atomic clock in transaction to achieve consistency.

Talk: https://www.youtube.com/watch?v=NthK17nbpYs

Related material:

https://w.amazon.com/bin/view/BigBird/Development/MetadataTables/Partitions

https://w.amazon.com/bin/view/BigBird/Development/Split-partition