提纲

Feature Guide

- row-oriented
- 支持key-value store和document store(row-oriented对于document store的支持友好吗)
- bin中支持复杂数据类型，如List，Map，嵌套的Map和List等
- 支持基于secondary index的查询
- 支持UDF
- aggregation框架(类似于MapReduce)提供快速灵活的查询
- 支持存储，索引和查询通过GeoJSON来表述的Geospatial数据

Data Distribution

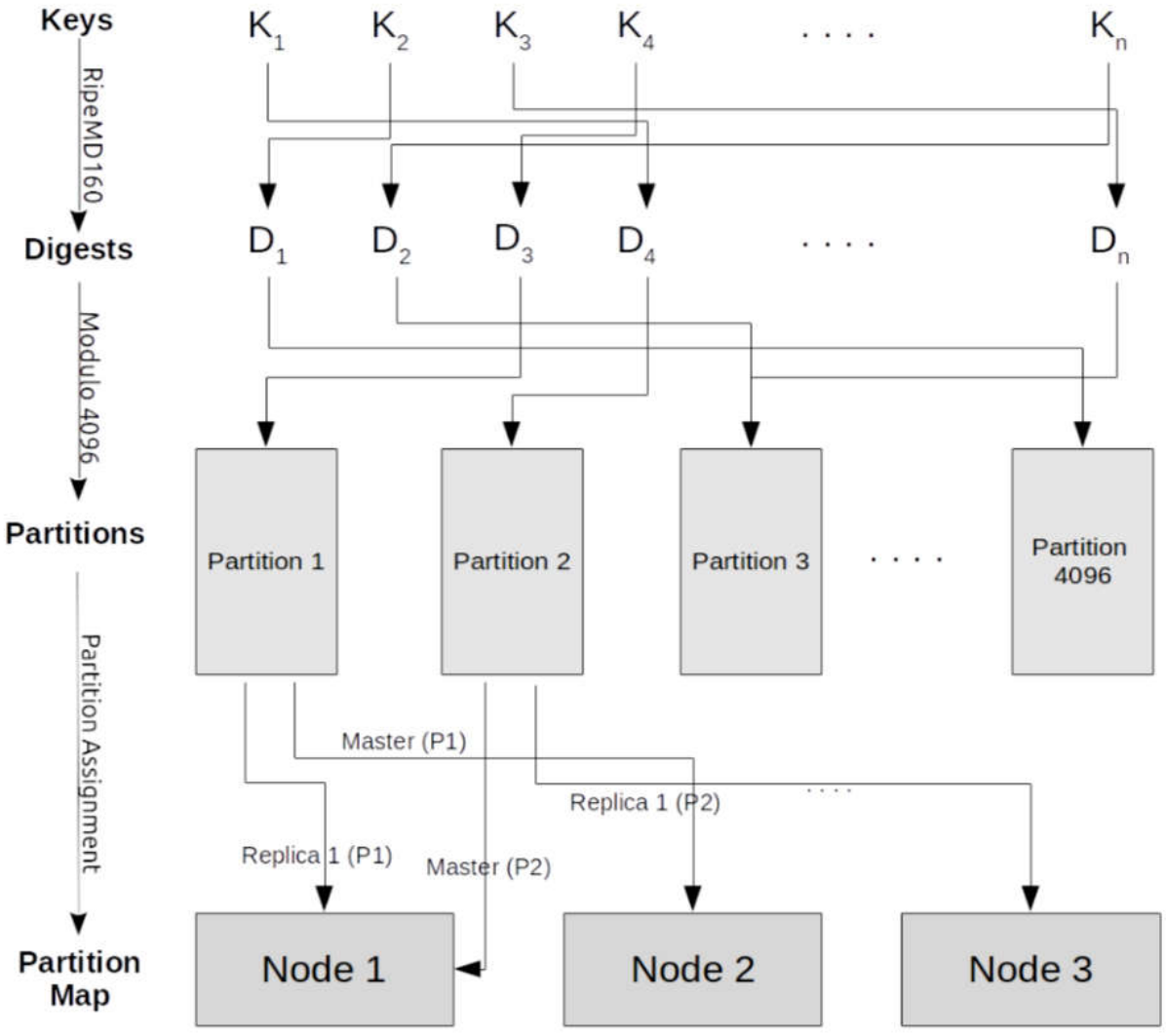


Figure 7: Data distribution

Aerospike Storage Layer - Hybrid Storage

Aerospike可以在以下任意类型的存储介质及其它们的组合上存储数据：

- DRAM
- NVME或者SSD
- PMEM(peristent memory)
- 传统的机械盘

但是并非所有类型的混合存储都是合理的，官方给出了下面的一个矩阵：

		Data Storage		
		NVMe Flash	**DRAM**	**PMEM**
Primary index storage	NVMe Flash	**All NVMe Flash**: Ultra-large records sets.	Not recommended.	Not recommended.
	DRAM	**Hybrid**: best price-performance.	High performance. No persistence.	Not common.
	PMEM	**Hybrid**: Fast restart after reboot. Very large data sets.	Not recommended.	**All PMEM**: Fast restart after reboot, with high performance.

namespaces，records and storage

不同的namespaces可以具有不同的storage engine，比如可以配置那些具有较小数据集但访问频繁的namespace存储在DRAM中，而那些具有较大数据集的namespace则存储在SSD中。

在Aerospike中：

- Record数据是存储在一起的
- 默认的每行占用1MB的存储空间
- 存储是copy-on-write模式
- 在defragmentation过程中回收空闲空间
- 每一个namespace具有固定大小的存储容量

对于数据存储在SSD/Flash时，写请求执行过程如下：

- 当接收到client的write请求的时候，会首先在行上加一个latch锁，以避免两个冲突的写操作同时操作相同的记录
  - 在发生网络分区的情况下，两个冲突的写操作会被接受，但是稍后会解决这个冲突
- 在master的内存中更新记录，要写到SSD/Flash的数据会被保存在一个write buffer中，当write buffer满的时候，该write buffer会被提交给SSD/Flash
  - write buffer的大小和行的最大存储空间一样
  - write buffer大小和写吞吐决定了未提交的数据的风险大小
    - 调整相关的配置参数可以降低数据丢失的风险
- 更新Replicas上的数据和内存索引
- 更新master中的内存索引(?官方博客中并未提到这一点)

Hybrid Memory Architecture

索引全部保存在内存中，而数据则只在持久化存储(SSD)中。

Data Model

components of data model

Component	Description
physical storage	You can choose the specific type of storage you want for each namespace: NVMEe Flash, DRAM, or PMEM. Namespaces can use different types of storage. You can also combine them as *hybrid storage*. The physical storage medium is also called the *storage engine*.
namespace	A namespace is a collection of records. A database can contain multiple namespaces. Each namespace can have its own physical storage type.
record	A record has a primary key. The primary key is also transformed into a corresponding digest.
set	Records can be optionally grouped into sets.
bin	A record also has bins. The data in the bin determines the data type of the bin. A record can have bins of varying data types. Bins can also be used to create secondary indexes.

Aerospike data model中的这些组件和传统数据库中的概念的一个**不严谨**的映射关系如下：

Aerospike component	Traditional RDBMS concept
namespace	tablespace
set	table
record	row
bin	column

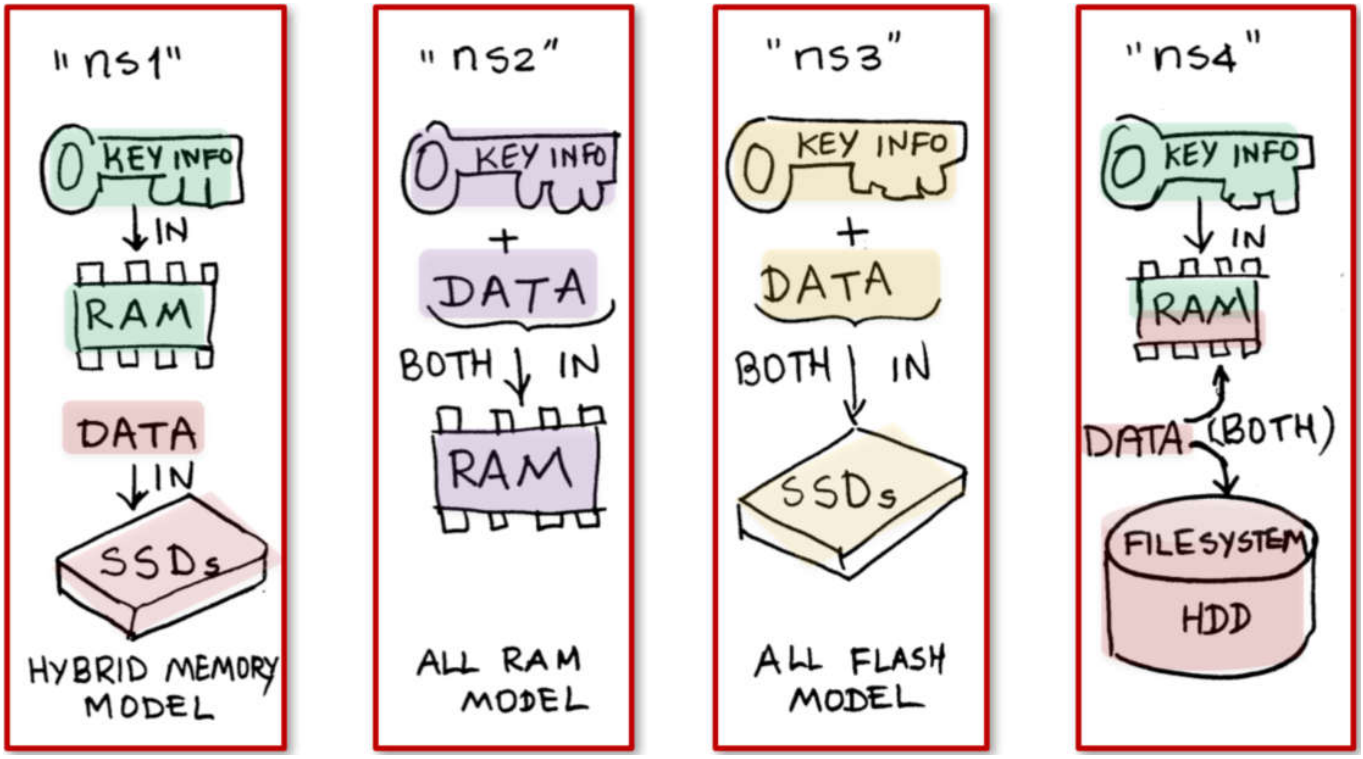
Physical Storage

请参考“Aerospike Storage Layer - Hybrid Storage”。

Namespaces

Namespace包含records，indexes和policies。policies决定了namespace的行为，包括以下policies：

- 物理上数据是如何存储的
- 每条记录多少replicas
- 记录何时过期



Sets

在namespace中，记录可选的可以被添加到set中。一个set一定是率属于一个namespace的。

默认的，所有的records都属于null set这个特殊的set。

Records

Record由以下部分组成：

Component	Description
key	Unique identifier. Records are addressable using a hash of its key, called the digest.
metadata	Record version information, called the generation count, the configured expiration date, called the time-to-live (TTL), and last update time (LUT).
bins	Bins store the data. The data type of the stored values sets the data type of the bin. Multiple data types can be stored.

Keys和Digests

用户使用Keys来读取或者写入数据，但是在Aerospike内部，Keys和它的可选的set的相关信息被用于计算出一个160-bits的digest，并以此digest来寻址到对应的记录。

Metadata

每条记录包含以下元数据：

- 版本编号(Generation count)：这个编号会在应用读取数据的时候返回给应用，用户可以借助于此编号来确保在他对该记录进行更新之前没有其它的关于该记录的更新
- TTL(Time to live)
- LUT(last update time)：上一次更新的时间

Bins和Data Types

对于每条记录而言，数据是保存在bins中的，bin由name和value组成，bin不会指定数据类型，数据类型在bin的value中定义。bin的value的数据类型可以变更，比如bin的value之前保存的是string类型，更新的时候可以将之修改为integer类型。

bin的value可以支持以下数据类型：

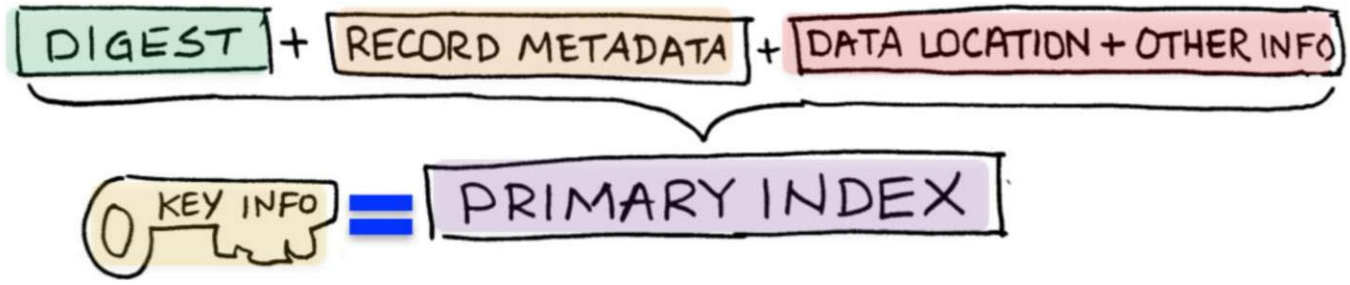
- Basic Data Types
  - Integer
  - String
  - Bytes
  - Double
- Complex Data Types
  - List
  - Map
  - Nesting Maps and Lists
- GeoJSON
- HyperLogLog and Probabilistic Data
- Language-Specific Serialized Blobs

Primary Index

每一个namespace被划分为4096个partitions，这些partitions在集群中的节点中均匀分布。每一条记录的Digest唯一决定了它所在的partition。

在内存中使用一种被称为sprig的结构来存储索引，每个partition都可以配置一定数目的sprigs。

primary index大小固定为64 bytes，primary index的组成如下：



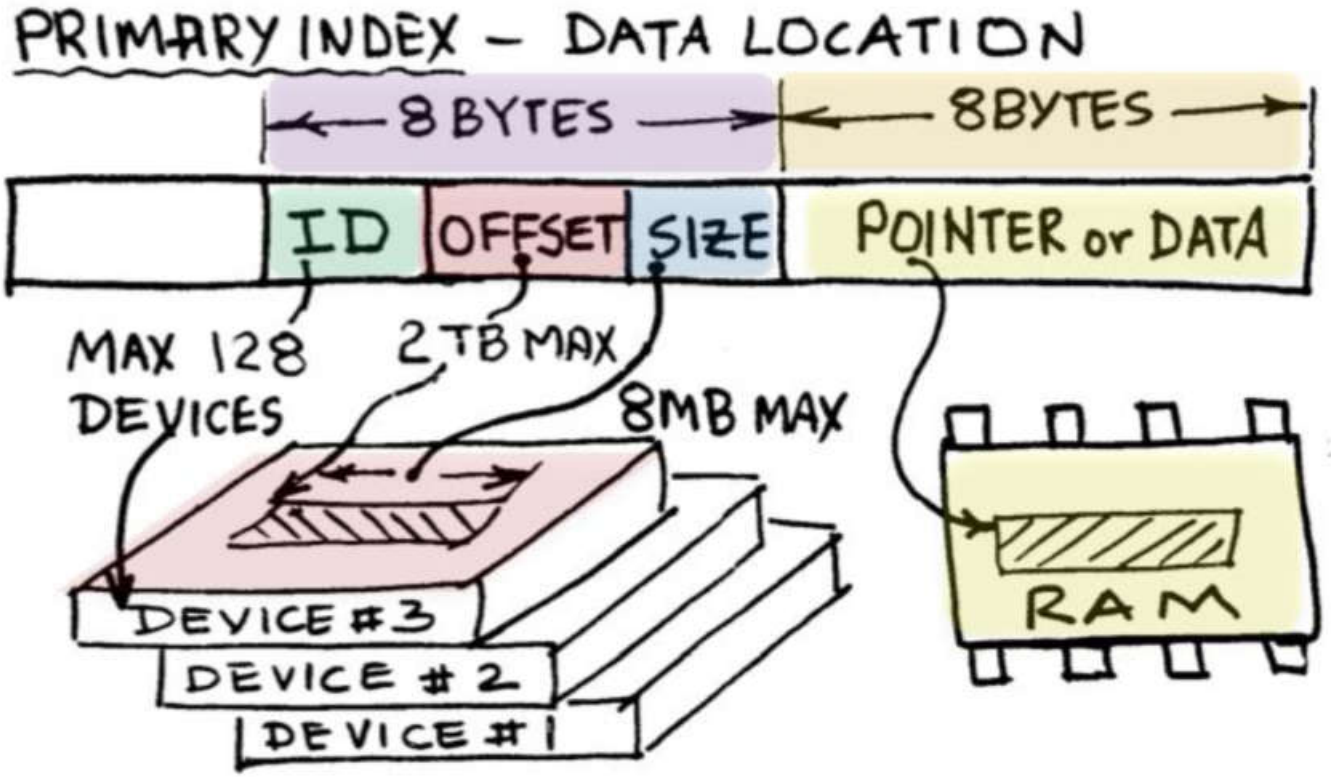
Primary Index Record Metadata

Primary index record metadata中包括以下信息：

- 版本编号(Generation count)：这个编号会在应用读取数据的时候返回给应用，用于解决冲突的更新
- TTL(Time to live)：主要被eviction subsystem使用
- LUT(last update time)：上一次更新的时间，用于冷启动过程中的冲突解决，迁移过程中的冲突解决，谓词过滤，增量备份，truncate和truncate-namespace命令等

Record Data Location

- 对于在硬盘中的记录：8 bytes Device Id(最多128个devices)，offset(最大2TB)，size(最大8MB)
- 对于在内存中的记录：8 bytes Pointer
- 特例 - 在索引中存储数据：
  - 如果记录只包含一个单一的bin，且bin类型是integer或者float，且该namespace中的数据都保存在内存中，则可以直接将它保存在primary index的memory pointer bytes中



Index Persistence

primary index可以通过数据进行重建。

Fast Restart

为了支持集群快速升级，Aerospike提供了fast restart功能。该功能从linux共享内存中分配索引内存，对于计划中的shutdown和restart，在restart的时候，Aerospike会重新连接到该共享内存，直接回复primary index。

Secondary Index

secondary index：

- 为快速查找存储于内存中
- 建立在集群的每一个节点上，每个次索引条目包含指向本节点记录位置的引用
- 可能指向master record或者replicated record(Contain pointers to both master records and replicated records in the node)

Secondary Index Metadata

Aerospike通过一个全局的数据结构 - System Metadata(SMD)来跟踪secondary index的创建情况，它的工作流程如下：

- 客户端请求create/delete/update secondary index。请求通过secondary index模块到达SMD
- SMD发送请求给paxos master
- Paxos master从集群中所有节点请求相关的元数据信息
- 当所有数据返回，它调用secondary index合并的回调函数，该函数负责分析胜出的secondary index元数据版本
- SMD发送请求给集群中所有接节点来接受新的元数据信息
- 每个节点执行create/delete secondary index
- 触发一个扫描并返回客户端

Writing Data with Secondary Indexes

对secondary index的更新和记录的更新是通过锁来保证原子性。

Garbage Collection

Garbage Collector会维护一个待移除的entry列表，然后由一个周期性唤醒的后台线程来执行真正的删除操作。

Transaction

ACID

Atomicity

对单条记录的读写操作，Aerospike严格保证Atomicity：

- 一条记录上的一个操作(可能涉及多个bins)会原子的被应用到该记录上。系统会在内存中创建一个副本保存该记录被更新后的最终的记录，同时在master上检查当前操作是否可能失败(因为冲突而失败？)，如果有失败发生，则新纪录不会被写入到存储中
- 当一个写操作成功执行之后，会确保后续的读操作读取到最新的数据，因此提供了immediate consistency

Aerospike支持在单个事务(一条AQL语句？因为Aerospike不支持begin，commit和abort等事务控制语句)内部同时执行多个读写操作，但是该事务中的写操作必须是一个简单的操作(如add，set和append等)，不能是一个复杂的更新逻辑。

除了支持单条记录的读写事务，Aerospike还支持分布式的多个keys的read事务。

Consistency

对于RDBMS来说，consistency意味着数据必须遵守所有的正确性规则，比如check-constraints，referential integrity constraints等，Aerospike暂不支持这些。

对于分布式系统来说，按照CAP理论中定义的Consistency，它要求集群中的多个副本之间数据是同步的。对于单个key上的操作，Aerospike通过同步写副本的方式提供immediate consistency，比如client只在replica上的记录也被成功更新的情况下才会被通知写操作成功。

Multi-key read transactions are implemented as a sequence of single key operations and do not hold record locks except for the time required to read a clean copy. Thus the multi-key read transaction provides a consistent snapshot of the data in the database (i.e., no “dirty reads” are done).这一句没搞懂？？？

Isolation

所有的写操作都由Master节点来负责。

Aerospike通过record locks提供read-committed隔离级别，因此如果关于某条记录存在多个并发的读写操作，那么在Aerospike内部这些操作会按一定的顺序执行，但是这些操作之间的顺序是不确定的。

借助应用层的CAS操作，Aerospike也支持乐观的并发控制。某条记录上的UDF操作也会通过record locks实现顺序执行。

Durability

通过以下技术实现Durability：

- 将数据存储在flash/SSD上
- 集群内多副本
- 跨集群多副本

集群内部节点间Replication是同步的，client只在replica上的记录也被成功更新的情况下才会被通知写操作成功。

跨集群的replication是异步执行的。

Aerospike也支持机架感知。

Aerospike可以配置为AP(Application Partition tolerant)模式或者SC(Strong Consistency)模式。在AP模式下存在更新丢失的问题，从Aerospike 4.0起，引入了SC模式，保证更新不丢失，且更新以正确的顺序被应用。

在AP模式下，记录的更新会被缓存在内存中的write buffer中，当该write buffer满的时候被flush到持久化存储中。在SC模式下，Aerospike可以可选的配置Commit to Device选项，如果配置了该选项，则记录在内存中更新之后会立即flush到持久化存储中，直到flush完成之后才会返回给应用。

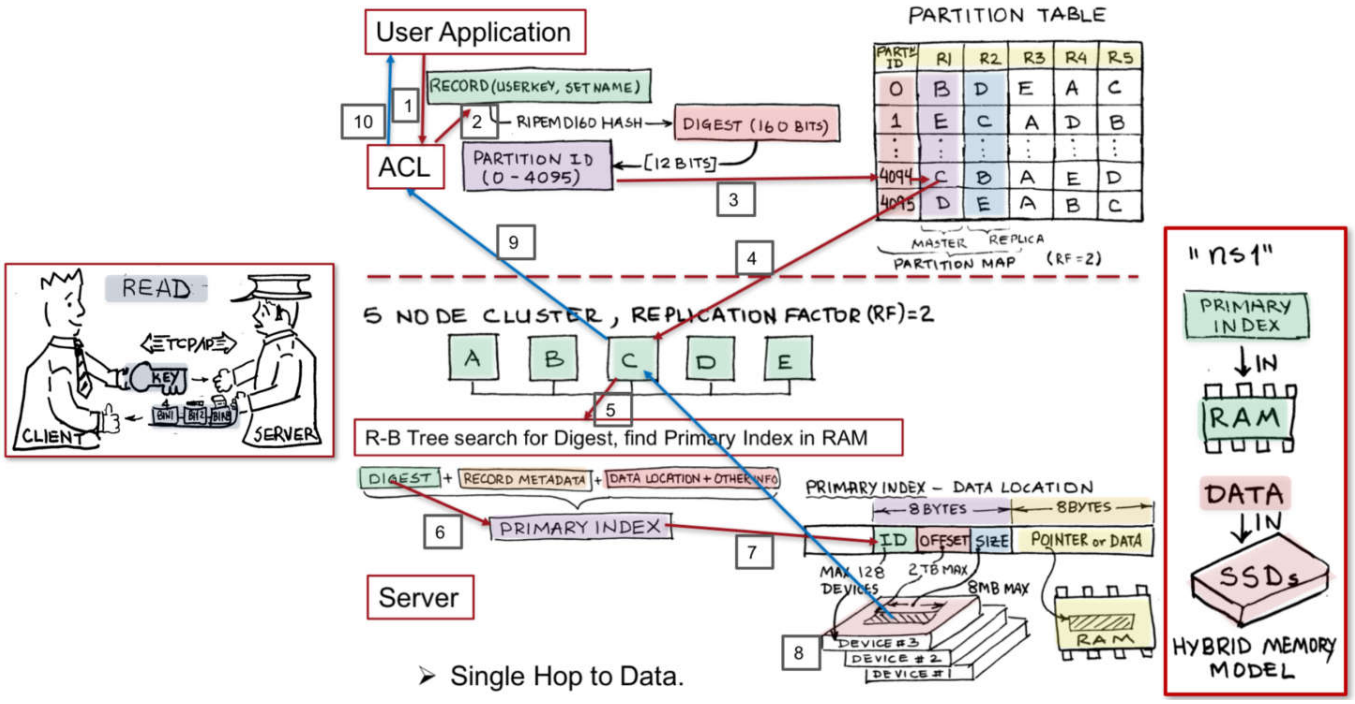
Handling conflicts

在AP模式下，如果集群发生了分区，则每个分区会继续工作。这样某些记录可能存在于某个分区内，而不存在于另一个分区内。每一个分区内部为了遵守replication factor的设定，会在分区内部的节点之间进行数据复制。每一个分区都可以接收来自用户的写请求，但是更新只在分区内部可见，不同的分区可能都针对某条记录执行了更新。当分区恢复之后，不同分区内关于相同记录的数据将被判定为inconsistent，这种情况下，Aerospike支持2中策略：

- 自动合并相同记录的不同版本(默认行为)
  - TTL based：具有较大TTL的win
  - Generation based：具有较大generation count的win
- 保留相同记录的不同版本，交给应用来进行merge(未来支持)
  - 应用会读取到关于相同记录的所有的版本，并且应用自身来解决这种inconsistency
  - 应用解决了这种inconsistency之后，重新写入解决了inconsistency之后的记录

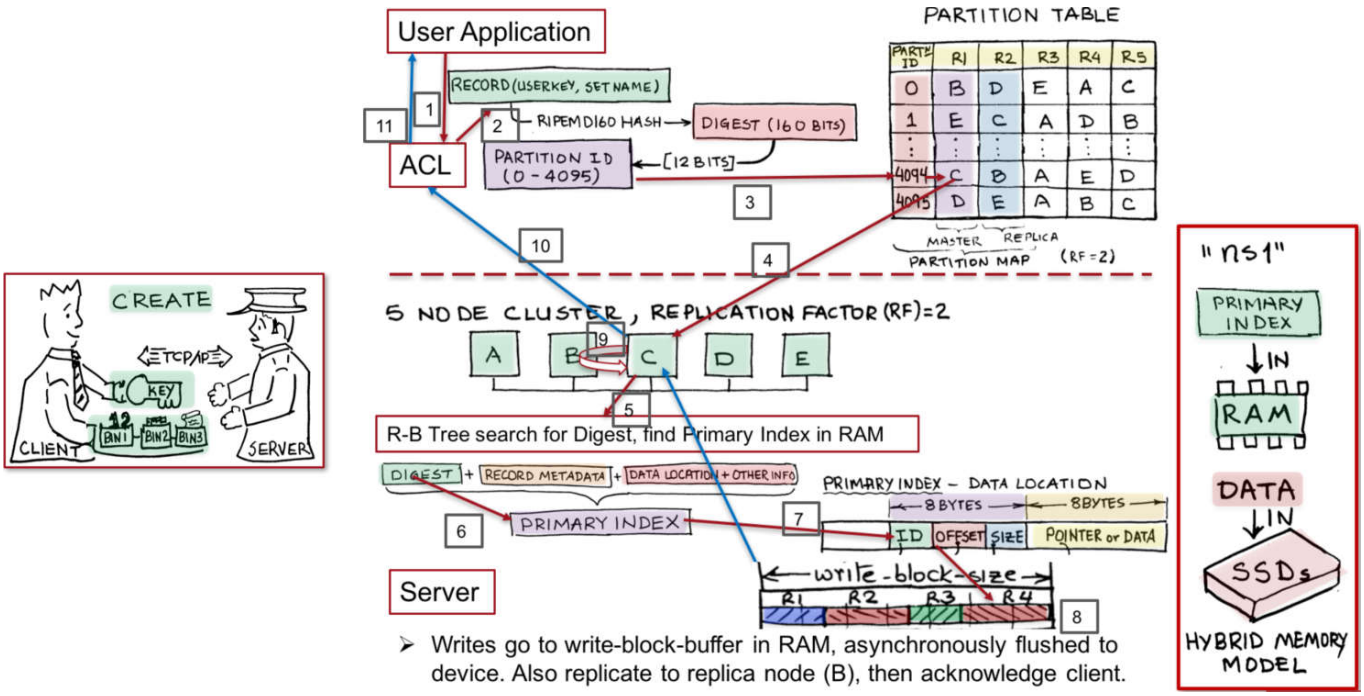
Read and Write Transactions

Read Transaction – Hybrid Memory Storage Example



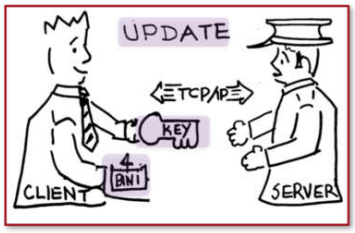
➢ Single Hop to Data.

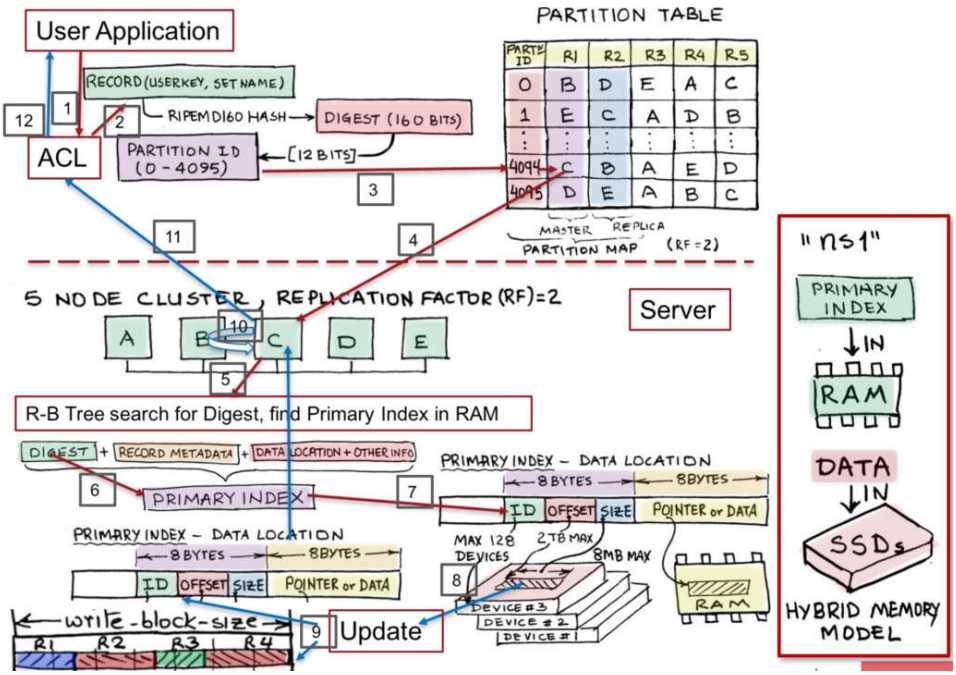
Create Transaction – Hybrid Memory Storage Example



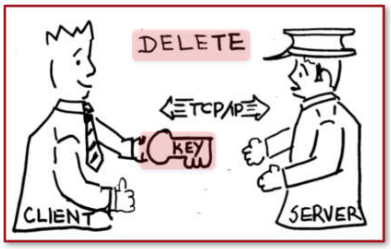
➢ Writes go to write-block-buffer in RAM, asynchronously flushed to device. Also replicate to replica node (B), then acknowledge client.

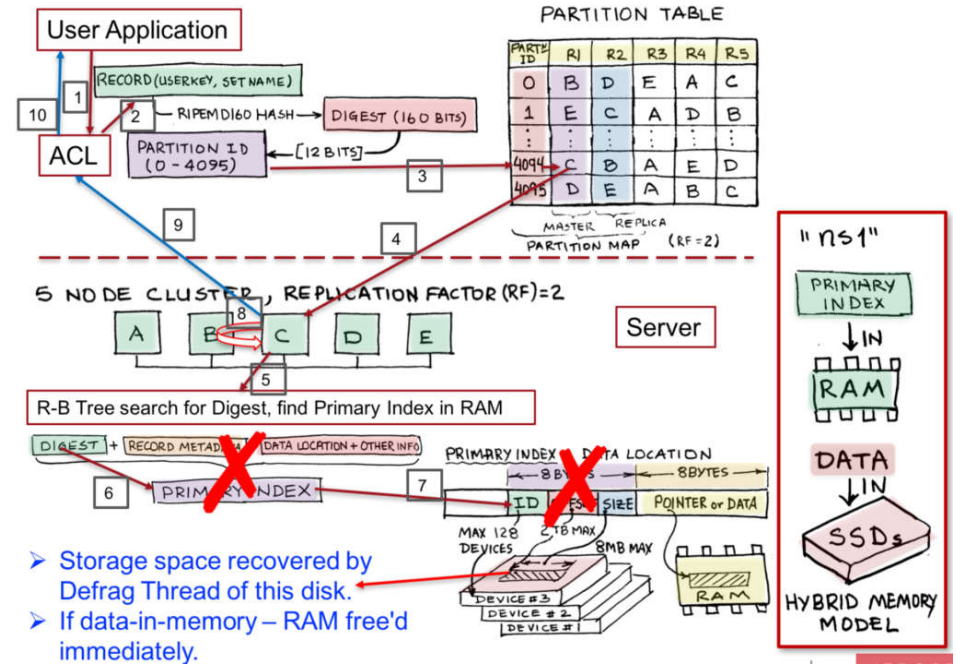
Update Transaction – Hybrid Memory Storage Example





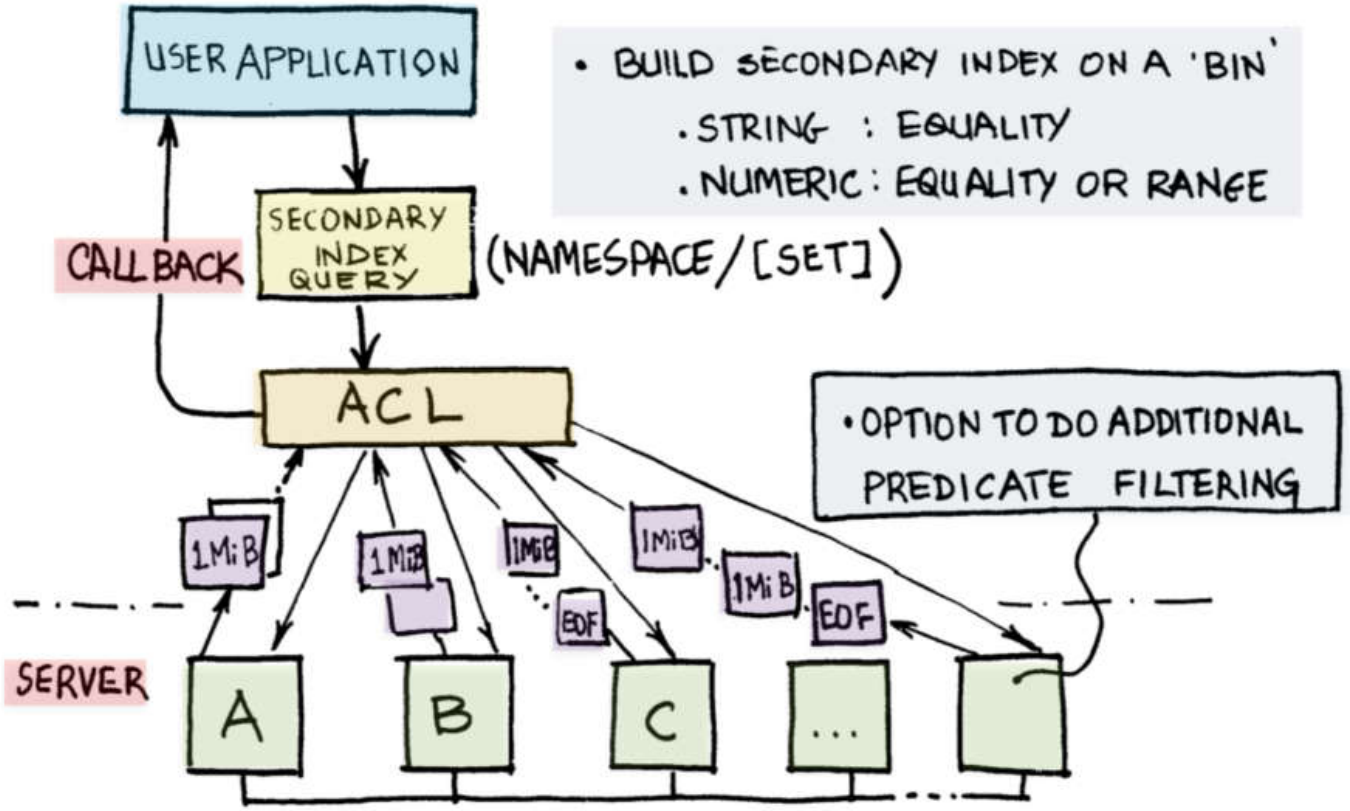
Delete Transaction – Hybrid Memory Storage Example





- Storage space recovered by Defrag Thread of this disk.
- If data-in-memory – RAM free'd immediately.

Secondary Index Query



Aerospike数据组织

这里讨论的是Aerospike数据和索引均保存在内存的情况下的数据组织。

Aerospike的索引在内存中是按照红黑树进行组织，数据在内存中没有特殊的组织，就是一个个单独的记录，红黑树中的索引中会记录数据所在的内存地址。