Aerospike

Aerospike介绍

Aerospike（简称AS）是一个分布式，可扩展的键值存储的NoSQL**数据库**。

T级别大数据高并发的结构化**数据存储**

读写操作达微妙级，99%的响应可在1毫秒内实现

采用混合架构，索引存储在内存中，而数据可存储在机械硬盘(HDD)或固态硬盘(SSD) 上（也可存储在内存）

AS内部在访问SSD屏蔽了文件系统层级，直接访问地址，保证了数据的读取速度。

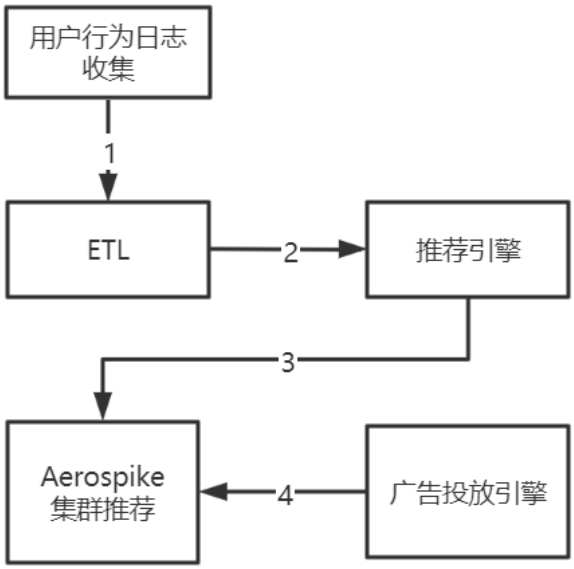
AS同时支持二级索引与Client聚合，支持简单的sql操作（aql），相比于其他nosql数据库，有一定优势。

Aerospike应用场景-广告行业

Aerospike作为一个大容量的NoSql解决方案，并未在国内厂中广泛商使用。它适合对容量要求比较大，QPS相对低一些的场景，目前主要集中于互联网广告行业使用（国外）

个性化推荐广告应用

个性化推荐广告是建立在了和掌握消费者独特的偏好和习性的基础之上，对消费者的购买需求做出准确的预 或引导，在合适的位置、合适的时间，以合适的形式向消费者呈现与其需求高度吻合的广告，以此来促进用户的消费行为。



用户行为日志收集系统收集日志之后推送到ETL做数据的清洗和转换

把ETL过后的数据发送到推荐引擎计算每个消费者的推荐结果，其中推荐逻辑包括规则和算法两部分

具体的规则有用户最近浏览、加入购物车、加入收藏等，

算法则包括商品相似性、用户相似性、文本相似性、图片相似性等算法。

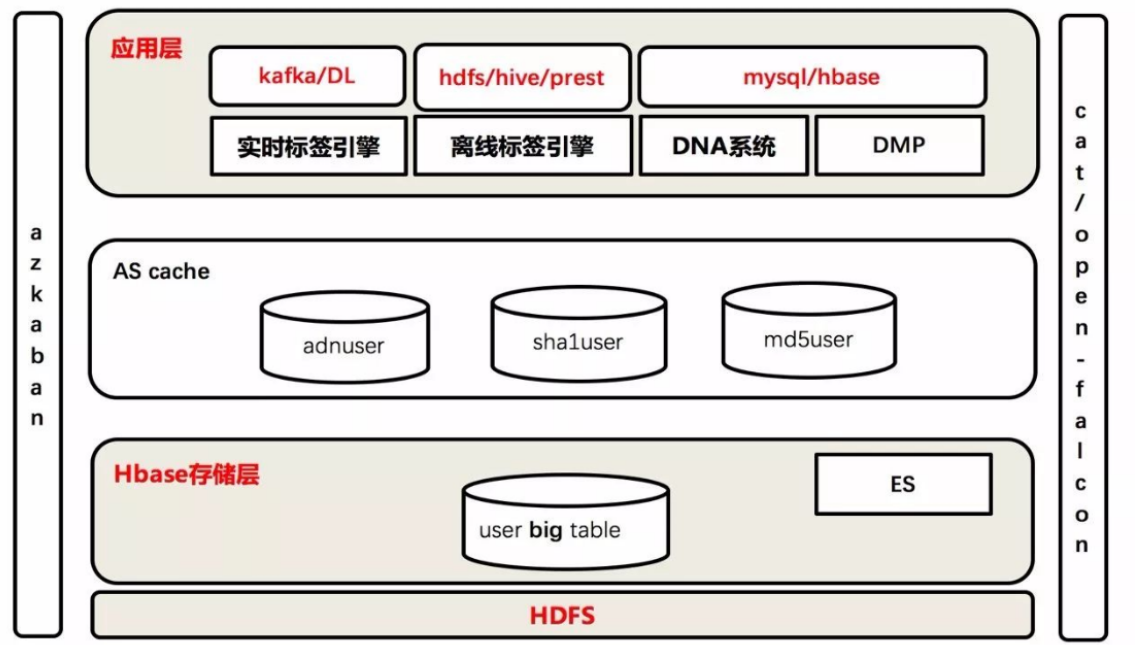
把推荐引擎的结果存入Aerospike集群中，并提供给广告投放引擎实时获取

实时竞价广告应用

当用户浏览一个加入SSP（供应方平台）的站点时，SSP会把此次请求发送到AD EXCHANGE（广告交易平台），然后ADX会把这次请求发送给多家DSP，DSP（需求方平台）根据自身的DMP（数据管理平台），通过对次用户的了解程度进行竞价，最终竞价胜出的DSP获得展现广告的机会。

DSP竞价（RTB：实时竞价）胜出的关键是DMP能够根据用户的历史浏览等数据分析和定位用户属性，其中实时竞价广告中非常重要的一个环节就是**UserProfile**（用户画像）。

分别通过HDFS和HBASE对日志进行离线和实时的分析，然后把用户画像的标签(tag : 程序猿、宅男...)结果存入高性能的Nosql数据库Aerospike中，同时把数据备份到异地数据中心。前端广告投放请求通过决策引擎（投放引擎）向用户画像数据库中读取相应的用户画像数据，然后根据竞价算法出价进行竞价。竞价成功之后就可以展现广告了。而在竞价成功之后，具体给用户展现什么样的广告，就是有上面说的个性化推荐广告来完成的。



Aerospike与Redis对比

Aerospike是NoSQL的数据存储，Redis是缓存

Aerospike是多线程的，而 Redis 是单线程的

Redis 需要开发人员自己管理分片并提供分片算法用于在各分片之间平衡数据；

client： hash 一致性hash

codis ： 代理处理sharding

RedisCluster： hash槽

而 AerospikeDB 可以自动处理相当于分片的工作；

在 Redis 中，为了增加吞吐量，需要增加 Redis 分片的数量，并重构分片算法及重新平衡数据，这通常需要停机；

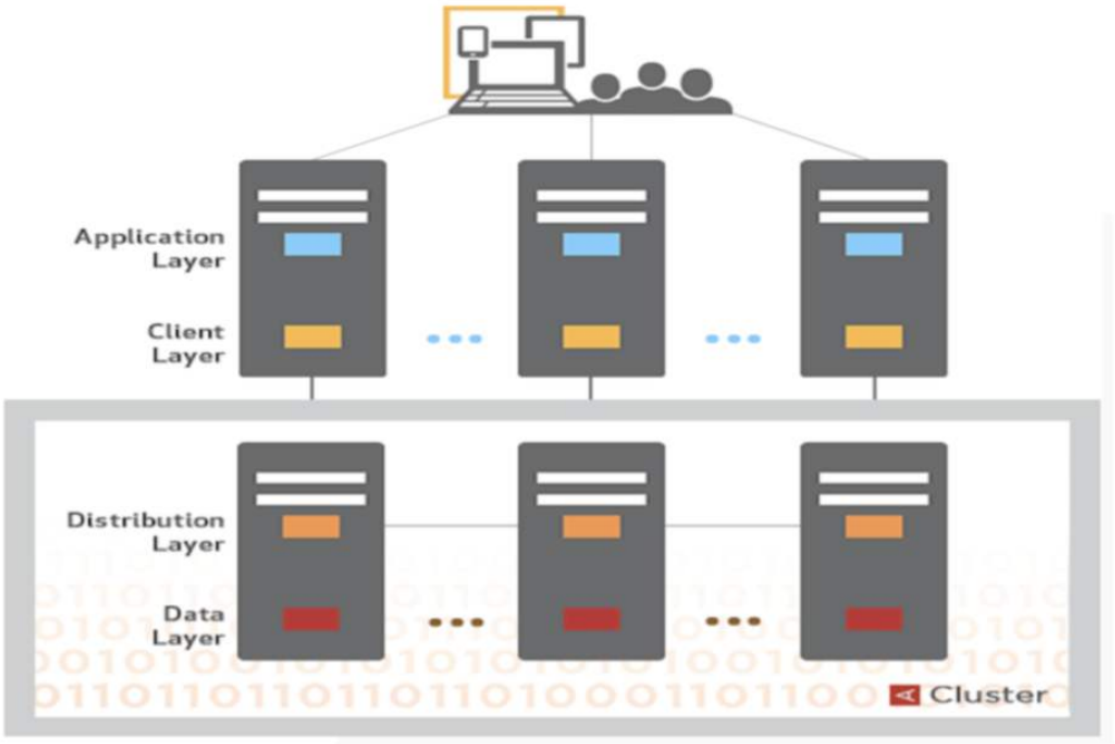
而在 AerospikeDB 中，可以动态增加数据卷和吞吐量，无需停机，并且 AerospikeDB 可以自动平衡数据和流量；

在 Redis 中，如果需要复制及故障转移功能，则需要开发人员自己在应用程序层同步数据；

而在 AerospikeDB 中，只需设置复制因子，然后由 AerospikeDB 完成同步复制操作，保持即时一致性；而且 AerospikeDB 可以透明地完成故障转移；

Redis是在内存中运行的，AerospikeDB在内存中存储索引，在HDD、SSD中保存数据，也可以在内存中

Aerospike架构



Aerospike分为三个层次：

Client层：

对Aerospike Server中的数据进行访问。

包括CRUD、批量操作和基于二级索引的查询

是一个“智能”客户端，支持C/C++、Java、GoLang、Python、C#、Php、Ruby、JavaScript等绝大部分主流语言。

追踪节点感知数据存储位置，当节点启动或停止时立即感知集群配置变化。

具有高效性、稳定性和内部的连接池

Distribution层：

负责管理集群内部数据的平衡分布、备份、容错和不同集群之间的数据同步。主要包含三个模块：

- Cluster Management Module

  用于追踪集群节点。关键算法是确定哪些节点是集群的一部分的Paxos-like一致投票过程。

  Aerospike实现专门的心跳检测（主动与被动），用于监控节点间的连通性。

- Data Migration Module

当有节点添加或移除时，该模块保证数据的重新分布，按照系统配置的复制因子确保每个数据块跨节点和跨数据中心复制。

- Transaction Processing Module

  确保读写的一致性与隔离性，写操作先写副本在写主库。该模块包括：

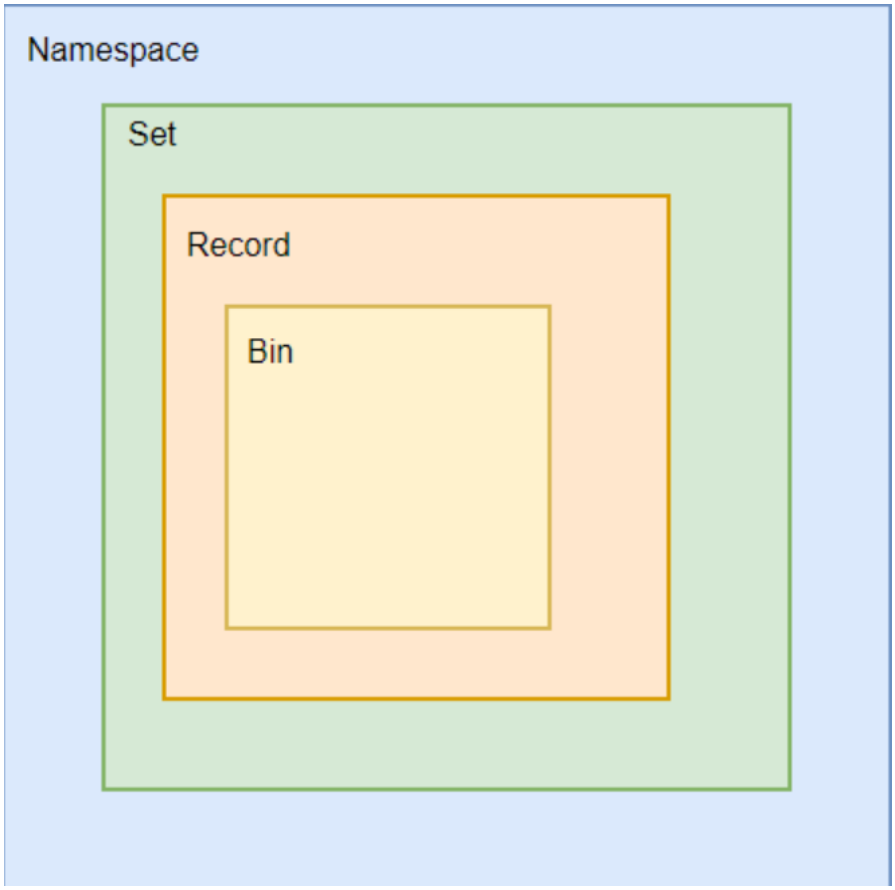
  Sync/Async Replication（同步/异步复制）：为保证写一致性，在提交数据之前向所有副本传播更新并将结果返回客户端。

  Proxy （代理）：集群重配置期间客户端可能出现短暂过期，透明代理请求到其他节点。

  Duplicate Resolution（副本解析）：当集群从活动分区恢复时，解决不同数据副本之间的冲突。

Data层：

负责数据的存储，Aerospike 属于弱语法的key-value数据库。数据存储模式如下：



namespace

命名空间，数据存在命名空间中，相当于RDBMS中的database，最多可设置32个。

一个namespace可关联多块SSD，一块SSD只关联一个namespace，每个namespace下包含4096个分片

set

集合，类似于数据库表，一个namespace最多1023个set

record

记录，类似数据库中的一行记录， 采用弱语法(Schema-Less)的方式，bin可随时增加或减少

key

主键，类型于数据表中的主键

bin

类似于数据库字段，支持Java基本数据类型：List、Map、Blob, 一个namespace下最多32767个bin

索引：Aerospike Index包含主索引（Primary Index）和二级索引（Second Index），索引存储在内存中，并不保存在硬盘里

一级索引（Primary-Index）

一级索引位于主键(key)，采用hash表与红黑树的混合

二级索引（Secondary-Index）

二级索引位于非主键上，这允许一对多关系。采用哈希表和B-tree的混合

磁盘：Aerospike可以直接访问硬盘中的raw blocks（原始块）直接寻址，并特别优化了Aerospike的最小化读、大块写和并行SSD来增加响应速度和吞吐量。

Aerospike的使用

Aerospike的安装

1、从网上下载aerospike-server-community-5.0.0.7-el6.tgz

```
wget https://www.aerospike.com/download/server/latest/artifact/el6
```

下载aerospike-server-community的最新版

2、解压aerospike-server-community-5.0.0.7-el6.tgz

```
tar -zxvf aerospike-server-community-5.0.0.7-el6.tgz
mv aerospike-server-community-5.0.0.7-el6 aerospike-server
```

3、安装aerospike-server和aerospike-tools

```
cd  aerospike-server
./asinstall
```

4、验证是否安装成功

```
[root@192 aerospike-server]# yum list installed | grep aerospike
aerospike-server-community.x86_64      5.0.0.7-1.el6                   installed
aerospike-tools.x86_64                 3.26.2-1.el6                    installed

#卸载aerospike
[root@localhost ~]# rpm -e aerospike-server-community.x86_64
[root@localhost ~]# rpm -e aerospike-tools.x86_64
[root@localhost ~]# rm -rf /etc/aerospike/
```

5、aerospike-server启动、停止、重启、状态

```
systemctl start aerospike
systemctl stop aerospike
systemctl restart aerospike
systemctl status aerospike
```

6、aerospike-server管理：asadm

```
asadm  进入管理端
Admin> info
Admin> i net
```

7、aerospike-server操作： aql

```
aql> show namespaces
+------------+
| namespaces |
+------------+
| "test"     |
| "bar"      |
```

Aerospike的基本概念

• **namespace** 策略容器，类似RDBMS中的database，可以设置副本数、内存大小、有效时长、存储引擎、文件存储位置。 • **Sets** 类似RDBMS中的表。 • **Records** 类似RDBMS中的行 • **Bin** 类似RDBMS中的列，一行可以有多个bins。

Aerospike	Mysql
namespace	database
set	table
bin	column
record	row
key pk kv	pk

Aerospike的数据操作

```
                                --主键  bins 插入可以不同
        INSERT INTO <ns>[.<set>] (PK, <bins>) VALUES (<key>, <values>)
        DELETE FROM <ns>[.<set>] WHERE PK = <key>

            <ns> is the namespace for the record.
            <set> is the set name for the record.
            <key> is the record's primary key.
            <bins> is a comma-separated list of bin names.
            <values> is comma-separated list of bin values
        没有update
        当insert 同一pk时，数据为修改

        Examples:

            INSERT INTO test.demo (PK, foo, bar) VALUES ('key1', 123, 'abc')
            DELETE FROM test.demo WHERE PK = 'key1'
 insert into  test.user1 (PK,name,age,sex,address) VALUES (2,'zhaoyun',21,
'M','beijing')
 insert into test.user2(pk,name,sex,age) values(1,'zhangfei','M',23)
 -- pk都是1 则是对原纪录的修改
```

```
 insert into test.user2(pk,name,sex,age) values(1,'diaochan','F',18)

    QUERY
        SELECT <bins> FROM <ns>[.<set>]
        SELECT <bins> FROM <ns>[.<set>] WHERE <bin> = <value>
        SELECT <bins> FROM <ns>[.<set>] WHERE <bin> BETWEEN <lower> AND <upper>
        SELECT <bins> FROM <ns>[.<set>] WHERE PK = <key>
        SELECT <bins> FROM <ns>[.<set>] IN <indextype> WHERE <bin> = <value>
        SELECT <bins> FROM <ns>[.<set>] IN <indextype> WHERE <bin> BETWEEN
<lower> AND <upper>

            <ns> is the namespace for the records to be queried.
            <set> is the set name for the record to be queried.
            <key> is the record's primary key.
            <bin> is the name of a bin.
            <value> is the value of a bin.
            <indextype> is the type of a index user wants to query.
(LIST/MAPKEYS/MAPVALUES)
            <bins> can be either a wildcard (*) or a comma-separated list of bin
names.
            <lower> is the lower bound for a numeric range query.
            <upper> is the lower bound for a numeric range query.

        Examples:

            SELECT * FROM test.demo
            SELECT * FROM test.demo WHERE PK = 'key1'
            SELECT foo, bar FROM test.demo WHERE PK = 'key1'
            SELECT foo, bar FROM test.demo WHERE foo = 123
            SELECT foo, bar FROM test.demo WHERE foo BETWEEN 0 AND 999

            select * from test.user2 where name='zhaoyun'
            --没有建立索引 ，不能查询
            Error: (201) AEROSPIKE_ERR_INDEX_NOT_FOUND

            create index idx_1 on test.user2(name) string

             select * from test.user2 where name='zhaoyun'
            +-----------+-----+-----+-----------+
            | name      | sex | age | address   |
            +-----------+-----+-----+-----------+
            | "zhaoyun" | "M" | 21  | "beijing" |
            +-----------+-----+-----+-----------+


       CREATE INDEX <index> ON <ns>[.<set>] (<bin>) NUMERIC|STRING|GEO2DSPHERE
       CREATE LIST/MAPKEYS/MAPVALUES INDEX <index> ON <ns>[.<set>] (<bin>)
NUMERIC|STRING|GEO2DSPHERE
       CREATE INDEX idx_foo ON test.demo (foo) NUMERIC
       DROP INDEX test.demo idx_foo
```

Aerospike的客户端（Java）

pom.xml引入aerospike-client

```
<!-- https://mvnrepository.com/artifact/com.aerospike/aerospike-client -->
        <dependency>
            <groupId>com.aerospike</groupId>
            <artifactId>aerospike-client</artifactId>
            <version>4.4.9</version>
        </dependency>
```

aerospike-client的API使用

```
        //IP+port
        AerospikeClient client=new AerospikeClient("192.168.127.128",3000);

        //写策略
        WritePolicy wp=new WritePolicy();
        //超时时间
        wp.setTimeout(1000);

        /*
            key
         */
        Key k1=new Key("test","user1",1);
        /*
        bins
         */
        // KV
        Bin b11=new Bin("name","zhangfei");
        Bin b12=new Bin("sex","M");
        Bin b13=new Bin("age",23);

        //写值
        client.put(wp,k1,b11,b12,b13);
        //读值
        Record r1=client.get(wp,k1,"name","age","sex");

        System.out.println(r1);

        System.out.println("==================================");

        Key k2=new Key("test","user1",2);
        /*
        bins
         */
        // KV
        Bin b21=new Bin("name","diaochan");
        Bin b22=new Bin("sex","F");
        Bin b23=new Bin("age",21);

        //写值
        client.put(wp,k2,b21,b22,b23);

        /*
        取得指定key的数据
         */
        //批量执行策略
        BatchPolicy bp=new BatchPolicy(wp);
        //key的数组
```

```
        Key[] ks={k1,k2};

        //循环输出
        for(Record r:client.get(bp,ks)){
            System.out.println(r);
        }
```

Aerospike集群实现

Aerospike集群管理

集群处理节点成员身份，并确保当前成员和所有集群中的节点保持一致。包括：集群视图、节点发现和视图改变

集群视图

每个Aerospike节点都会自动分配唯一的节点标识符，是由MAC地址和监听端口组成的。包括：

cluster_key：是一个随机生成的8字节值，用于标识集群视图的实例。

succession_list:是作为集群一部分的唯一节点标识符集合。

节点发现

节点间通过心跳消息来检测节点的有效或失效

集群中的每个节点维护一个邻接表，是最近向该节点发送心跳消息的其他节点列表

如果心跳超时，则表示该节点失效，从邻接表中删除

Aerospike还采用其他信息作为备用二次心跳机制

集群中的每个节点通过计算平均消息丢失来评估每个相邻节点的运行状态评分

视图改变

领接表一旦发生改变，就会触发运行一个Paxos共识算法来确定一个新的集群视图。

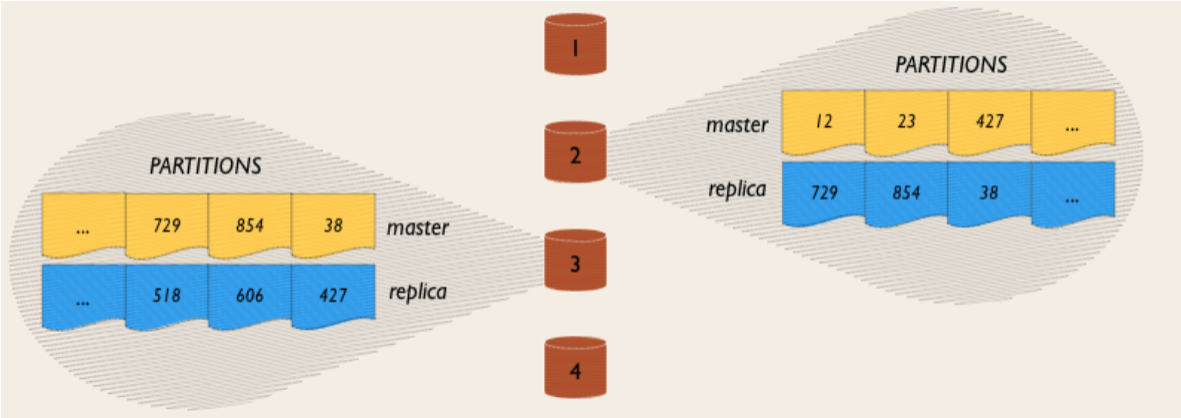
Aerospike把邻接表中节点标识符最高的节点当Proposer，并承担Proposal的角色

Proposal提成新的集群视图，如果被接受，则节点开始重新分配数据（Rebalence）

Aerospike数据分布

Aerospike有智能分区算法，即把用户输入的key在内部根据RIPEMD-160算法，重新hash出一个key并取前20位，然后相对均衡的把数据分布到各个节点之上。并且满足namespace配置文件的配置（例如保存多少个备份、是存在磁盘还是存在内存中）。

每个Digest Space（namespace）被分为4096个不重叠的分区（partitions），它是Aerospike数据存储的最小单位



如上，一个4个节点的集群，每个节点存储1/4数据的主节点，同时也存储1/4数据的副本。如果节点1不可访问，节点1的副本将被拷贝到其他节点上。

复制因子（ replication factor）是一个配置参数，不能超过集群节点数。副本越多可靠性越高。

作为必须经过所有数据副本的写请求也越高。实践中，大部分部署使用的数据因子为2（一份主数据和一个副本）。 同步复制保证即时一致性，没有数据丢失。在提交数据并返回结果给客户端之前，写事务被传播到所有副本。

主成功同时备成功后，客户端认为是成功

在集群重新配置期间，当Aerospike智能终端发送请求到那些短暂过时的错误节点时，Aerospike智能集群会透明的代理请求至正确的节点。

Aerospike集群配置和部署

有两种方式可以搭建集群：Multicast组播方式（UDP）和Mesh网格方式（TCP）

Multicast组播方式（UDP）

```
heartbeat {
        mode multicast
        multicast-group 239.1.139.1
        port 3000
        address 192.168.127.131
        interval 150
        timeout 10
    }
```

udp是不可靠协议，所以有可能会造成节点脱落，而且网络有可能不支持组播

Mesh网格方式（TCP）

```
heartbeat {
        mode mesh
        # add current node address here
        address 192.168.127.131
        port 3000
        # add all cluster node address here
        mesh-seed-address-port 192.168.127.131 3002
        mesh-seed-address-port 192.168.127.128 3002

        interval 150
        timeout 10
    }
```

集群部署

在192.168.127.128上安装Aerospike后，修改配置文件/etc/aerospike/aerospike.conf

```
vim /etc/aerospike/aerospike.conf

service {
        user root
        group root
        paxos-single-replica-limit 1 # Number of nodes where the replica count
is automatically reduced to 1.
        pidfile /var/run/aerospike/asd.pid
        proto-fd-max 15000
}

logging {
        # Log file must be an absolute path.
        file /var/log/aerospike/aerospike.log {
                context any info
        }
}

network {
        service {
                address any
                port 3000
                access-address 192.168.127.128 3002
        }

        heartbeat {
                mode mesh
                address 192.168.127.128
                port 3002
                #all cluster
                mesh-seed-address-port 192.168.127.128 3002
                mesh-seed-address-port 192.168.127.131 3002
                # To use unicast-mesh heartbeats, remove the 3 lines above, and
see
                # aerospike_mesh.conf for alternative.

                interval 150
                timeout 10
```

```
        }

        fabric {
                address any
                port 3001
        }

        info {
                address any
                port 3003
        }
}

namespace test {
        replication-factor 2
        memory-size 256M

        storage-engine memory
}

namespace bar {
        replication-factor 2
        memory-size 256M

        storage-engine memory

}
```

在192.168.127.131上安装Aerospike后，修改配置文件

```
service {
        user root
        group root
        paxos-single-replica-limit 1 # Number of nodes where the replica count
is automatically reduced to 1.
        pidfile /var/run/aerospike/asd.pid
        proto-fd-max 15000
}

logging {
        # Log file must be an absolute path.
        file /var/log/aerospike/aerospike.log {
                context any info
        }
}

network {
        service {
                address any
                port 3000
                access-address 192.168.127.131 3002
        }

        heartbeat {
                mode mesh
```

```
                address 192.168.127.131
                port 3002
                #all cluster
                mesh-seed-address-port 192.168.127.131 3002
                mesh-seed-address-port 192.168.127.128 3002
                # To use unicast-mesh heartbeats, remove the 3 lines above, and
see
                # aerospike_mesh.conf for alternative.

                interval 150
                timeout 10
        }

        fabric {
                address any
                port 3001
        }

        info {
                address any
                port 3003
        }
}

namespace test {
        replication-factor 2
        memory-size 256M

        storage-engine memory
}

namespace bar {
        replication-factor 2
        memory-size 256M

        storage-engine memory

}
```

Aerospike集群的访问

```
Host[] hosts = new Host[]{
                new Host("192.168.127.128", 3000),
                new Host("192.168.127.131", 3000)
        };
        ClientPolicy policy = new ClientPolicy();

        AerospikeClient client = new AerospikeClient(policy, hosts);


        //写策略
        WritePolicy wp = new WritePolicy();
        //超时时间
        wp.setTimeout(500);
```

```
        Key key1 = new Key("test", "SUser", "11");
        Bin bin11 = new Bin("name", "zhangfei-c");
        Bin bin12 = new Bin("age", 25);
        Bin bin13 = new Bin("sex", "M-c");
        client.put(wp, key1, bin11, bin12, bin13);

        Key key2 = new Key("test", "SUser", "22");
        Bin bin21 = new Bin("name", "zhaoyun-c");
        Bin bin22 = new Bin("age", 24);
        Bin bin23 = new Bin("sex", "M-c");
        client.put(wp, key2, bin21, bin22, bin23);


        Record r1 = client.get(wp, key1, "name", "age", "sex");
        System.out.println(r1);
```