HongKe 虹科

教程

使用Redis企业版的Active-Active功能开发应用程序

目录

使用Redis 企业版的Active-Active 功能开发应用程序 2

简介 2

地理分布式 架构样本 2

Redis CRDTs 3

如何使用Redis CRDTs构建你的解决方案? 3

使用Redis 企业版开发和测试应用程序 4

应用案例 5

计数：投票、点赞、表情符号计数 5

分布式缓存 6

使用共享会话数据协作 7

多区域数据采集 11

附录 12

附录1：Redis CRDT命令和冲突解决方案内置在Redis企业版里 13

附录2：为基于CRDT的Redis搭建docker开发环境 16

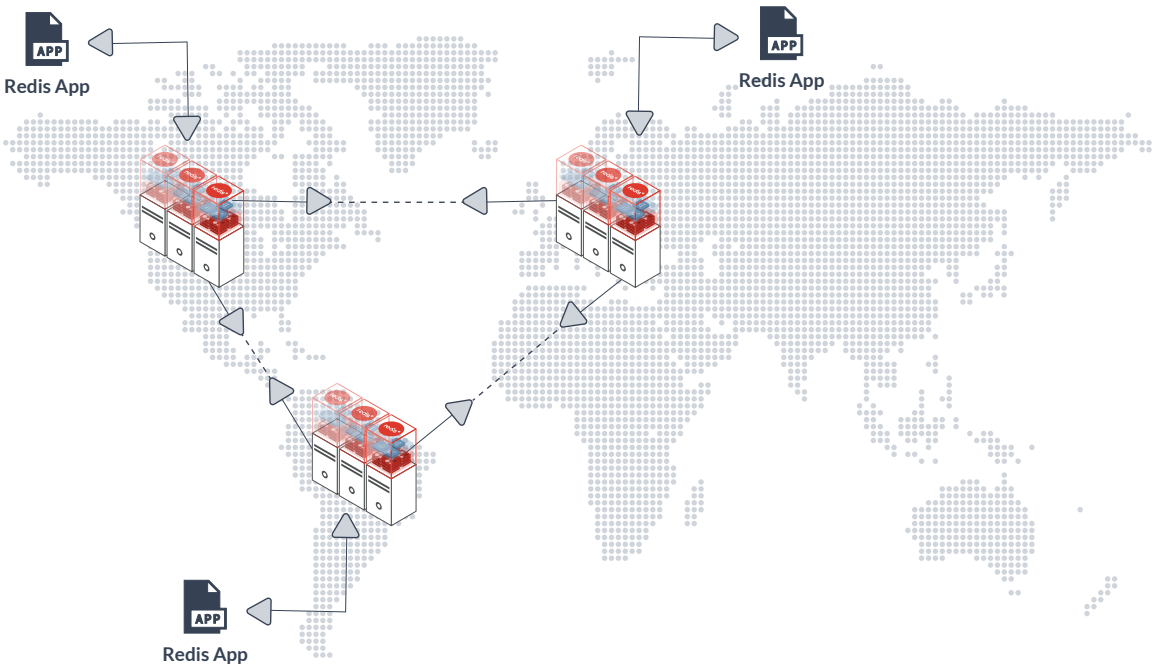
简介

CAP 定理所描述的一致性和可用性，对于地理分布式应用程序的架构师来说是一个巨大的挑战。网络分区在所难免，数据中心之间的高延迟总是会导致一些连接断开,即使断开的时间很短。基于地理分布式应用程序的传统解决方案架构遵循 CAP 定理,即要么放弃数据一致性,要么降低可用性。遗憾的是,在构建交互式用户应用程序时可用性是不能被牺牲的。最近,架构师开始探索一致性并采用“最终一致性”模型。在这个模型中,应用程序为使他们最终保持一致，则依赖于数据库管理系统来合并数据的所有本地副本,。

在出现数据冲突之前,使用最终一致性模型也可以满足应用程序的日常需求。一些弱的最终一致性模型能保证尽最大努力解决冲突,但难以保证强大的一致性。好消息是,基于无冲突复制数据类型（CRDTs)构建的模型提供了强大的最终一致性。CRDTs是通过一组预先确定的冲突解决规则和语义来实现这一点的。在基于 CRDTs的数据库之上构建的应用程序必须设计为适应冲突的解决语义。在本文中,我们将探讨如何使用 Redis CRDTs 设计、开发和测试地理分布式应用程序。我们还将通过计数器、分布式缓存、共享会话和多区域数据摄取四个案例来说明这些技术。

地理分布式架构用例

对于本文的其余部分,我们将假设一个地理分布式应用程序部署在世界各地的不同数据中心。如图所示,每个region都有一个本地的Redis企业版集群，一个跨区域部署无冲突的分布式数据库。在此设置中,每个区域中的数据库都是其他区域中数据库的副本。最重要的是,每个数据库都是一个活动数据库,这意味着它既支持读取也支持写入操作。
地理分布式应用程序的本地实例连接到最近的数据库,可以给读写操作提供本地延迟。借助 Redis CRDTs,Redis企业版提供了强大的最终一致性。



Redis CRDTs

什么是CRDTs?

CRDTs是一种可以汇集来自所有数据库副本数据的特殊数据类型,。流行的 CRDTs 有计数器(仅增长计数器)、PN 计数器(正负计数器)、寄存器、G 集(仅增长集)和 OR 集 (观察删除集)。实际上,他们依靠以下数学特性来汇集数据:

1. 交换律: a D b = b D a
2. 结合律: a D (b D c) = (a D b) D c
3. 幂等: a D a = a

G计数器是合并运算操作 CRDTs 的完美示例。使用a +b=b +a 和 a+（b +c)= （a +b) +c。副本之间仅交换更新(添加)的数据。CRDTs将通过添加它们来实现合并更新。例如：G 集应用幂等性（{a,b,c}U c= {a,b,c) 来合并所有元素，这就避免了元素在移动时被重复添加到数据结构中并通过不同的路径汇聚。

Redis的优势

Redis在 CRDTs 方面处于领先地位，它已经拥有丰富的数据结构组合:Strings, Hashes, Lists, Sets, Sorted Sets,Bitfields, Geo, Hyperloglog 和 Streams。Redis 企业版将其中一些流行的数据结构展为 CRDTs。它还引入了一种新的，Redis 中没有的数据类型：Counter。Redis CRDTs 使用与特定用例相关的汇集以及冲突解决规则。以下是 Redis CRDTs 的列表,以及它们的冲突解决技术:

1. 计数器既可以作为G计数器也可以作为PN计数器使用
2. 字符串（寄存器）应用“last writer wins”（LWW)来解决冲突
3. 当使用hincrby设置键时，Hashes充当Strings和counters的寄存器
4. Sets 支持 G-sets 和 observed-remove sets并且在冲突期间“add”优先级高于“delete”
5. **Sorted Sets**结合了 sets 和 counters的语义 (用于计分）
6. **Lists**确保所有的副本具有相同的顺序，但最终顺序不是基于将项目推送到列表的顺序

如何使用Redis CRDTs构建你的解决方案?

如果您的应用程序已经将 Redis 用作本地非Active-Active数据库,那么您的应用程序堆栈中就拥有了所有必需的库。但是您不能只是将数据库重新配置为CRDTs就希望你的应用程序能像之前那样工作。分布式数据库中的数据生命周期是不同于集中式数据库的。因此：

1. **让你的应用程序处于无状态。**如果您的应用程序已经将 Redis 用作本地非Active-Active数据库,那么您的应用程序堆栈中就拥有了所有必需的库。但是您不能只是将数据库重新配置为CRDTs就希望你的应用程序能像之前那样工作。分布式数据库中的数据生命周期是不同于集中式数据库的。因此：
2. **选择适合您的用例 CRDTs。**Counter 是最简单的 CRDTs;它可以应用于全局投票,、跟踪活动会话、计量等用例。但是,如果要合并分布式对象的状态,则还必须考虑其他数据结构。例如,对于允许用户编辑共享文档的应用程序,您可能不仅希望它保留编辑,还希望保留执行顺序。在这种情况下，将编辑保存在Lists中是比将它们存储在Strings或 Set CRDTs中更好的解决方案。同时，了解 CRDTs强制执行的冲突解决语义,并使解决方案符合规则也是很重要的。

3. **CRDT 不是万能的解决方案。**虽然 CRDT 对于许多用例来说确实是很好的工具,但它们并不是对所有用例来说都是最好的(例如 ACID事务)。基于 CRDT 的 Redis 企业版非常适合微服务架构,您可以为每个微服务提供专用数据库。因此,您可以仅将 Redis CRDT 应用于那些需要基于 CRDT 用例的微服务。

您的应用程序应专注于逻辑，并将数据管理和同步等复杂操作委托给Redis企业版。您也可以将Redis企业版视为应用程序的 RAM 内存扩展。

使用Redis企业版开发和测试应用程序

Redis企业版旨在降低开发应用程序和将其推广到市场的复杂性。

客户端库

好消息是Redis CRDTs与 Redis数据类型兼容。因此,所有Redis 客户端都与Redis CRDTs 兼容。如果您将现有应用程序从独立的Redis 数据库移植到基于 CRDTs的 Redis企业版数据库中，则无需更改客户端库。

开发测试环境

若希望更快地进入市场,我们建议您采用一致的开发、测试、预生产和生产设置。为开发和测试配置最小型环境的最简单方法就是使用基于 Docker 的设置。
按照附录 B 中的说明，在基于 Docker 的设置的三个不同子网上部署一个三节点、分布式、基于 CRDTs 的Redis企业版数据库。

测试你的应用程序

使用分布式多主数据库测试应用程序听起来可能很复杂,但大多数情况下,您要测试的只是以下两种情况下的数据一致性和应用程序可用性:（1)当分布式数据库处于连接状态时(2) 当数据库之间有一个网络分区时。
通过在您的开发环境中设置一个三节点分布式数据库,就可以实现自动化并覆盖单元测试本身中的大部分测试场景。以下是测试您的应用程序的推荐指南:

1. 对于开启网络连接且节点之间的延迟较低时的测试用例，必须更加强调模拟冲突。您可以通过多次跨不同节点更新相同数据来实现这一点。
   暂停和验证所有节点数据。尽管 Redis企业版集群持续同步,但要测试最终一致性,您需要暂停测试并检查数据。
   对于验证,验证跨所有 Redis 节点的数据的这两件事:（1)所有节点都具有相同的数据 （2)当发生冲突时,冲突解决方案与设计一致。

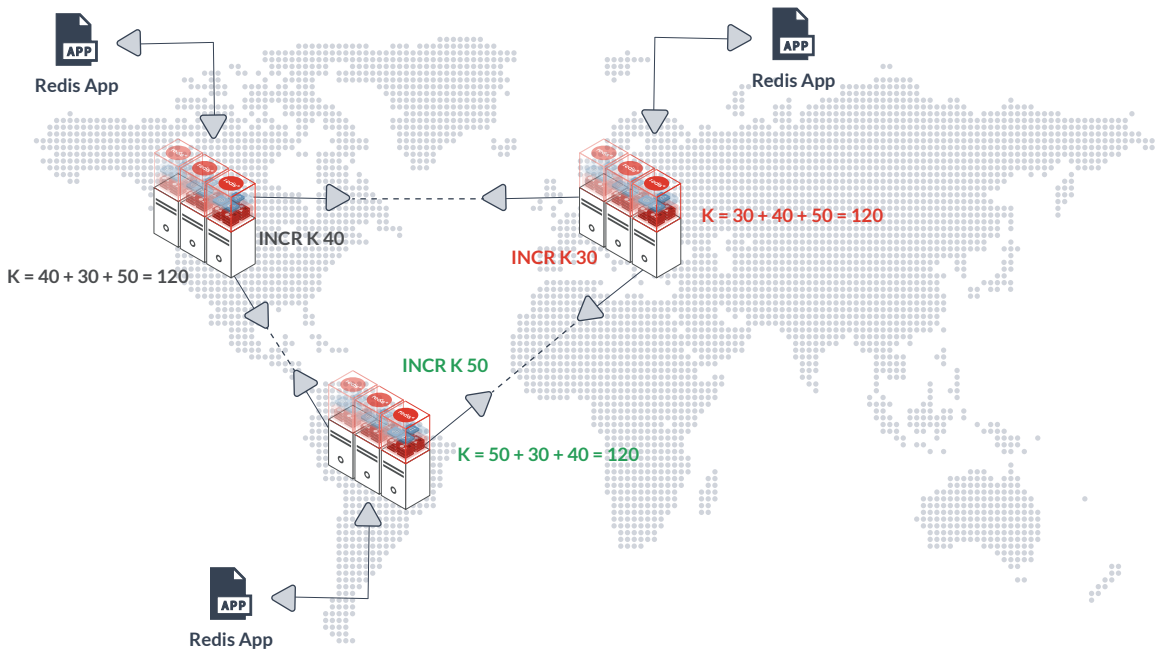
2. 分区网络的测试用例：您需要测试数据库无法相互同步的情况,所以使用分区网络执行与之前相同的测试用例。由于网络分裂时,数据库不会合并所有数据。因此,您的测试用例必须设定在您只读取数据的本地副本的情况下。

在测试的后半部分,重新连接所有网络以测试合并是如何发生的。如果您正在使用与上一节中相同的测试用例,您需要确认最终的数据是否与上一组步骤中的相同。

应用案例

1. 计数器：投票、点赞、表情符号计数

计数器有很多应用场景。也许你有一个地理分布的应用程序,用于收集选票、测量文章的点赞数量或追踪对消息的表情符号的反应数量。在此示例中,每个地理位置本地的应用程序连接到最近的Redis 企业版集群。然后它更新并读取具有本地延迟的计数器。



设计
数据类型
Counter

示例keys
Counter - poll:[pollId]:counter
Hash counter - message:[messageId]:emoji hearts

示例代码:

```
// 1. 所有用户共享计数器
 void countVote(String pollId){
// Increments the counter.  Redis Enterprise synchronizes
// the counter by merging the counts across across all locations
 //Redis 命令: INCR poll:[pollId]:counter
}
// 2. 读取全局计数
long getVoteCount(String pollId){
```

```
        // Redis command: GET poll:[pollId]:counter
}
// 3. 计算特定的消息点赞数
void incrementMessageReaction(String messageId, String emoji){
// Increment the count inside the Hash data structure associated

// with the message
// message:[messageId]:emojis is the key of the Hash
// [emoji] is a Hash item that tracks the number of reactions

// Redis command: HINCRBY message:[messageId]:emojis [emoji]
1 }
```

连接网络的测试用例：

1. **一个区域的增量计数**
   a. 在所有地区运行你的应用程序，然后在一个位置增加计数。
   b. 停止计数，留意每个地区的计数器的变化。
   c. 你的应用程序应该在所有区域保持一致，显示相同的数量。

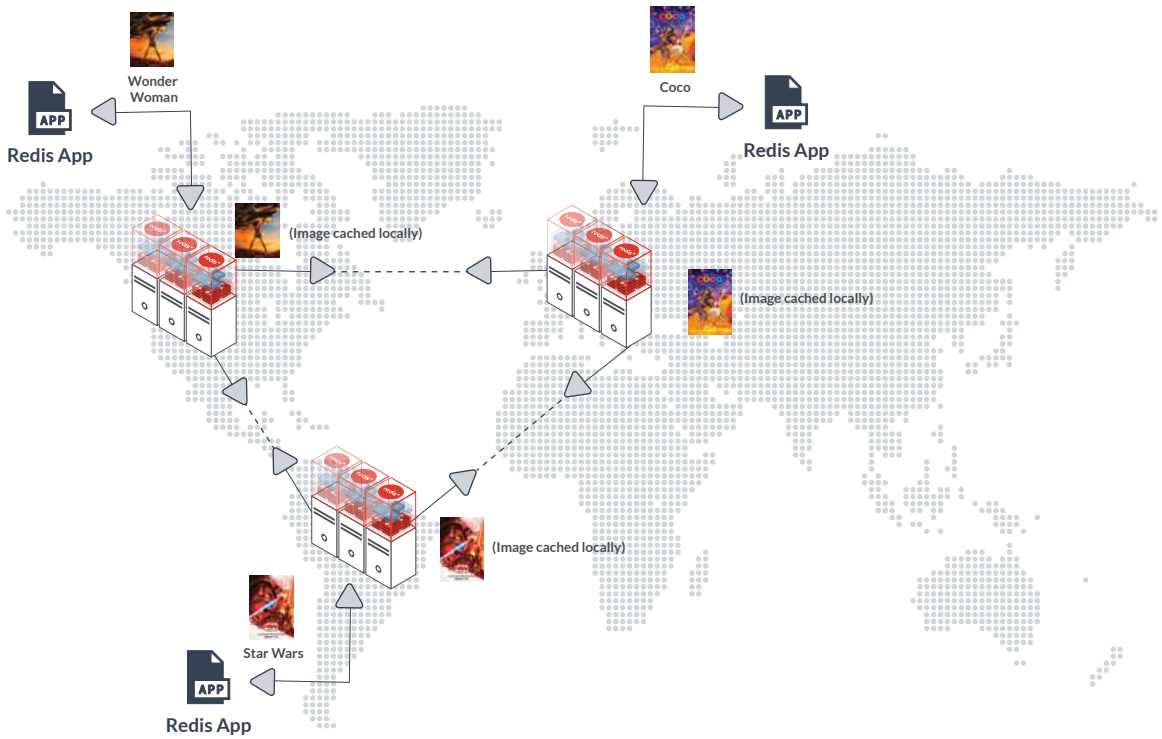
2. **多区域增量计数**
   a. 在所有区域运行你的应用程序，在所有位置增加计数。
   b. 停止计数，留意每个地区的计数器的变化。
   c. 你的应用程序应该在所有地区保持一致，显示相同的数量。

分区网络的测试用例：

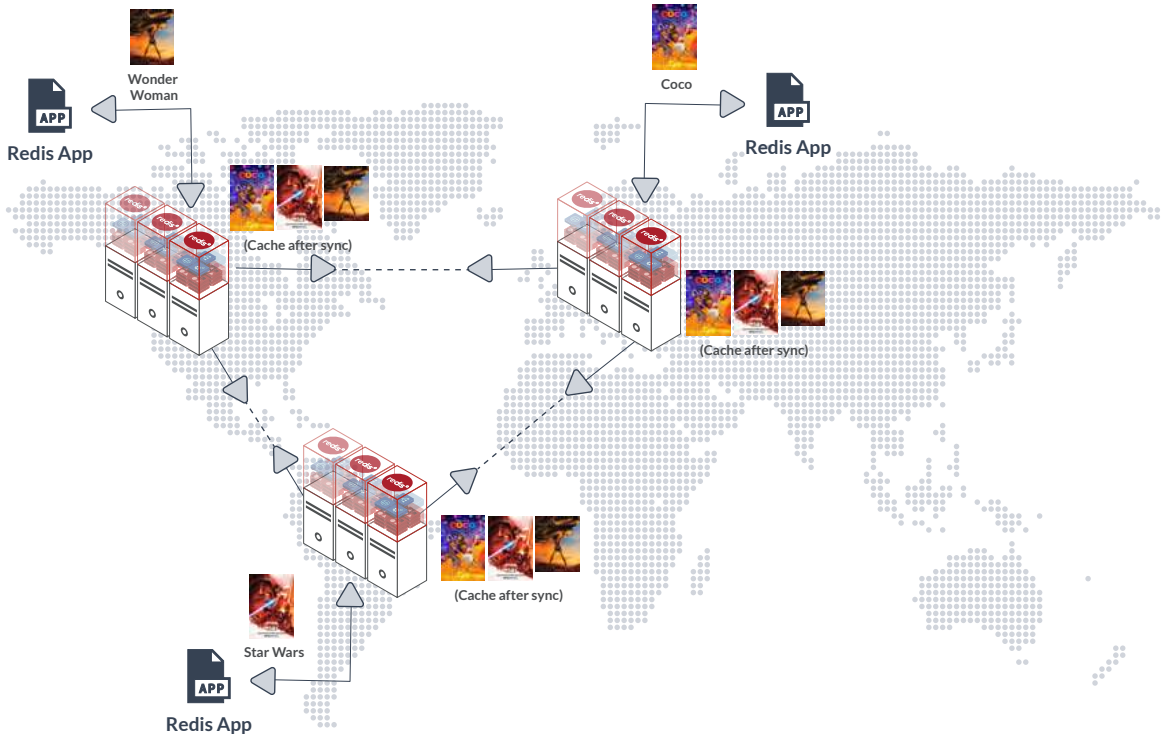
1. **多区域增量计数**
   a. 隔离Redis企业版集群。
   b. 在所有地区运行你的应用程序，在所有位置增加计数。
   c. 停止计数，留意每个地区的计数器的变化。
   d. 你的应用程序应该只显示本地增量。
   e. 重新连接网络。
   f. 所有应用程序都应显示更新的计数，并且你的应用程序可以适应这种操作。

2. 分布式缓存

分布式缓存的缓存机制与本地缓存中使用的缓存机制相同:首先您的应用程序尝试从缓存中获取对象，如果该对象不存在于缓存中,应用程序会从主存储中检索它并将其保存在具有适当过期时间的缓存中。基于CRDTs 的Redis企业版缓存的对象会自动复制到所有区域。在下面的示例中每部电影的海报图像都缓存在本地并分发到所有位置。



第一步：海报图像存储在本地缓存中。



第二步：Redis企业版跨所有区域同步缓存。

设计

数据类型: String
Redis 中的 String 是作为字节数组实现的。 String 就像一个具有“最后写入者获胜”(LWW)策略的寄存器。如果两个集群更新相同的字符串,则具有较晚时间的集群会保留在缓存中。当的缓存解决方案更新了对象的过期时间，如果与该较晚的时间存在冲突，则选择以最大的时间为准。

数据类型: Hash
为同一对象存储元数据或多个项目,则可以使用Hash数据结构作为缓存。

示例keys
String: object:[objectId]
Hash: object:[objectId][key1] [value1] [key2] [value2] [key3] [value3]

```
示例代码:
// 1.将对象缓存为字符串
void cacheString(String objectId, String cacheData, int ttl){
        // Redis command: SET object:[objectId] [cacheData] ex [ttl] }
// 2. 以字符串形式获取缓存
String getFromCache(String objectId){

                // Redis command: GET object:[objectId]
}
```

连接网络的测试用例

1. 在所有地区运行您的应用程序，然后在所有区域设置一个新的缓存对象。
2. 验证您的应用程序是否可以从本地集群中提取缓存在其他区域的缓存对象。

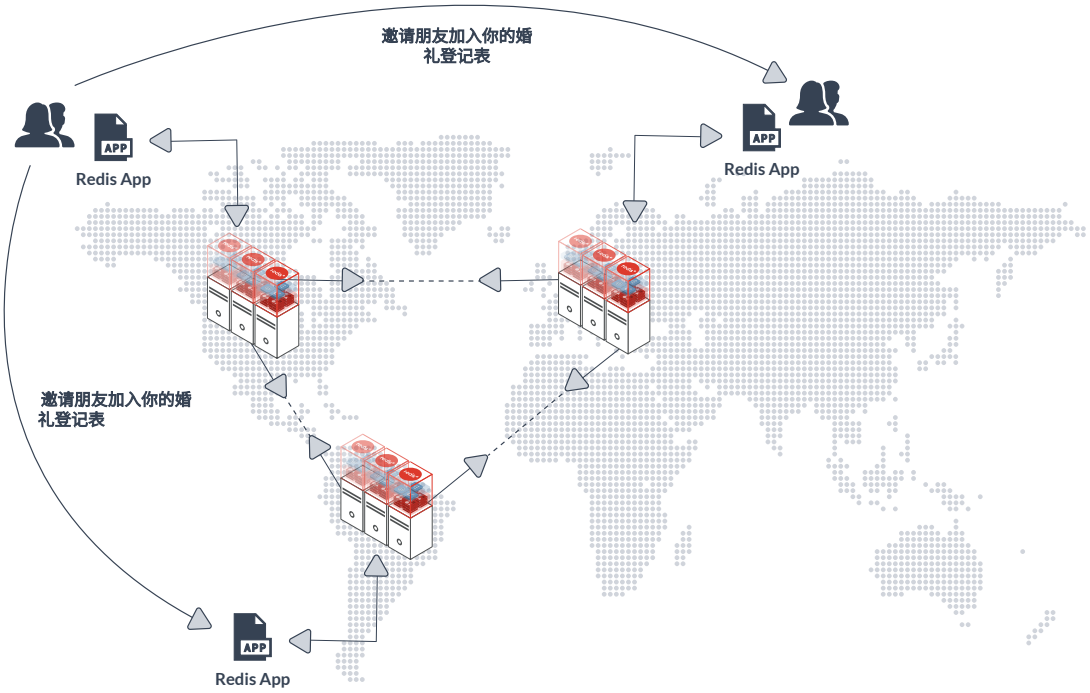
分区网络的测试用例

1. 模拟网络分区并设置值。
2. 重新连接并验证您的应用程序是否可以用于处理以下情况：
    a. 在“最后写入者获胜”中更新相同的结果
    b. 合并后，键将获得最长的TTL值

3.使用共享会话数据进行协作

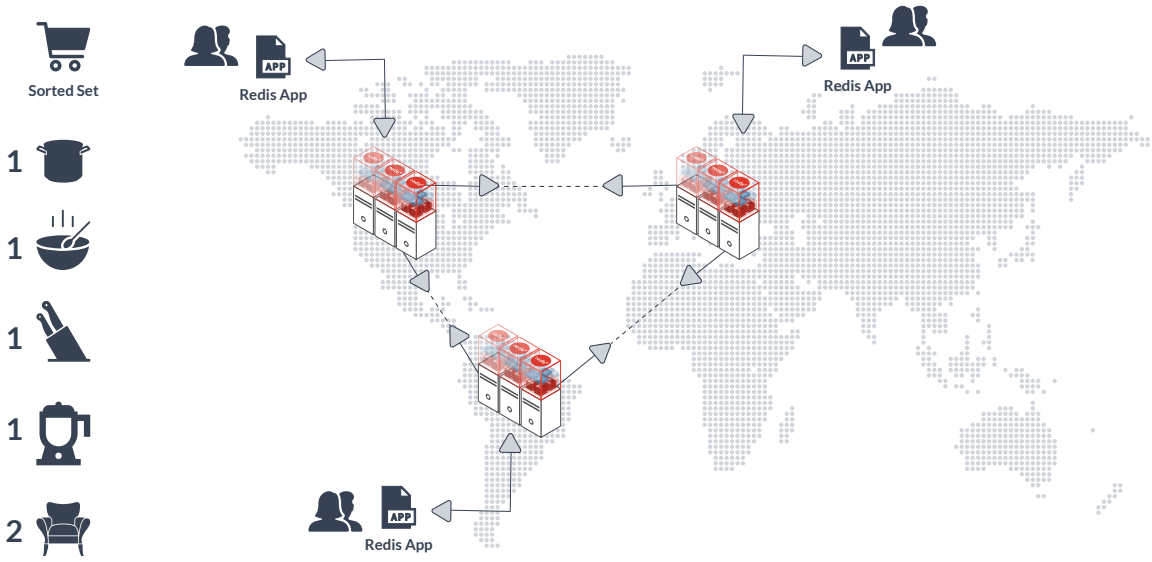
CRDT 最初是为支持多用户文档编辑而开发的。共享会话用于游戏、电子商务、社交网络,聊天、协作、紧急响应和许多其他应用程序。在下面的示例中,我们演示了如何开发一个简单的婚礼登记应用程序。在此应用程序中，一对新订婚夫妇的所有祝福者都将他们的礼物添加到作为共享会话管理的购物车中。

登记表应用程序是一个地理分布式应用程序,每个实例都连接到本地数据库。要开始会话,登记表的所有者会邀请他们在世界各地的朋友。一旦受邀者接受了邀请他们都可以访问会话对象。然后,他们开始购物并将商品添加到购物车。



第一步 :这对新人邀请他们的国际朋友加入他们的婚礼登记表。

第二步：受邀者将他们的商品添加到购物车 (有序集里面)。



第 3步:婚礼登记表的所有者现在拥有他们可以查到的所有商品。

设计

数据类型: Sorted Set

这包含购物车的所有商品。

数据类型: Set

Set数据结构包含所有的活动会话。

示例keys

SharedSession:[sessionId]

ShoppingCart:[sessionId]

示例代码

```
void joinSession(String sharedSessionID, sessionID){
        // Redis command: SADD sharedSession:[sharedSessionId] [sessionID] } void
addToCart(String sharedSessionId, String productId, int count){

        // Redis command: ZADD sharedSession:[sharedSessionId] productId
count } getCartItems(String sharedSessionId){

// Redis command: ZRANGE sharedSession:sessionSessionId 0 -1
}
```

测试你的应用程序是否反映了CRDT规则:

1. 权重就像一个计数器;两次添加相同的对象会导致 Sorted Set 中有两个对象。
2. 添加胜于删除。

连接网络的测试用例

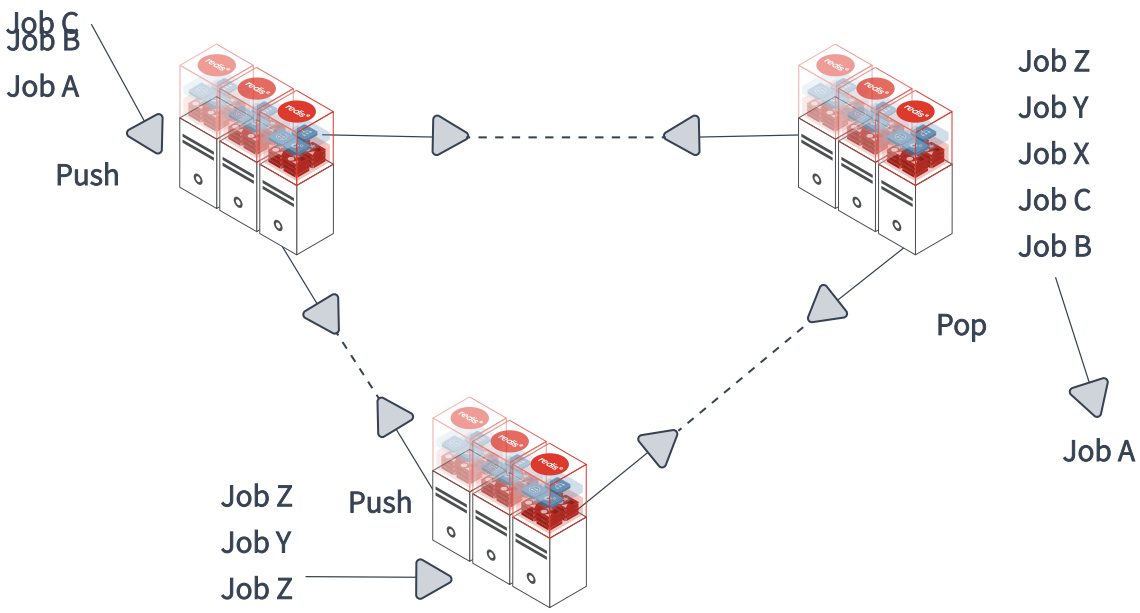
1. 在所有区域运行您的应用程序;将对象添加到分布式 Sorted Set 数据结构中。
2. 权重就像一个计数器;两次添加相同的对象会导致 Sorted Set 中有两个对象。
3. 验证您的应用程序是否符合这些 CRDT 语义。

分区网络的测试用例

1. 模拟网络分区;在本地集群中添加和删除对象。
2. 重新连接并验证您的应用程序是否可以用于处理 Sorted Set 的 CRDT 语义。

4. 多区域数据摄取

列表或队列在许多应用程序中使用。在此示例中,我们将演示如何实施分布式作业订单处理解决方案。如图所示,作业订单处理系统在基于 CRDT 的List数据结构中维护活动作业。该解决方案在不同位置收集作业。每个位置的分布式应用程序连接到最近的 Redis企业版集群。这减少了写入操作的网络延迟,进而允许应用程序支持大量作业提交。这些作业从其中一个集群的List数据结构中弹出，确保了一个作业只被处理一次。



设计

数据类型: List

List 数据结构用作 FIFO 队列。

示例keys

job:[jobQueueId]

示例代码

```
pushJob(String jobQueueId, String job){
        // Redis command: LPUSH job:[jobQueueId] [job] }
popJob(String jobQueueId){

        // Redis command: RPOP job:[jobQueueId]
}
```

连接网络的测试用例

1. 在所有区域运行您的应用程序;将对象添加到 List 数据结构中。 CRDT 合并添加到列表中的对象。
2. 验证您的应用程序是否符合 List 数据结构的 CRDT 语义。

注意:如果您的应用程序从一个集群的列表中弹出一个对象,那么您的作业处理器可以确保处理一项作业。但是 ,如果您从所有集群中弹出对象,那么两个或更多位置最终可能会处理同一个作业。如果您的应用程序要求您只能处理一次作业,那么您将需要在应用程序层中市实施自己的锁定解决方案。

分区网络的测试用例

1. 模拟网络分区;将作业添加到本地集群。
2. 重新连接并验证您的应用程序是否可以用于处理List的 CRDT 语义。

附录

关于 Redis 企业版的最后一点说明,在开始设计和开发应用程序之前,您需要了解一些事项:

1. **您不能覆盖默认的 CRDT 冲突解决语义。**一些数据库提供最终的一致性，并允许应用程序用它们自己的冲突解决语义覆盖默认的冲突解决语义。然而，Redis企业版解决了数据层下面的冲突，允许应用程序专注于实现逻辑，并将维护数据一致性的复杂性交给Redis企业版。
2. **多键命令只有在键被插入到同一个分片时才有效。**基于CRDT的Redis企业版部署在集群模式,即使你只有一个分片数据库。多键命令和 Lua 脚本要求在特定调用中访问的所有键都属于同一个分片。有一些技术可以将键映射到特定的分片。最流行的方法是标准哈希策略。
3. **Lua脚本执行遵循命令同步，而不是脚本同步。**这意味着在一个位置调用Lua脚本不会导致在其他位置调用相同的Lua脚本。相反，当本地Lua脚本执行Redis命令时，只有由这些命令引起的数据更改会复制到其他区域。

4. **使用本地主从设置实现高可用性。**应用程序没有必要了解全局数据库部署拓扑(除非应用程序在设计上需要这样做)。Redis企业版的本地集群仍然在两个不同的节点上有主备实例。如果节点宕机，自动故障检测和恢复遵循与基于CRDTs的Redis企业版数据库相同的技术。

5. **如果需要持久化,快照优先于AOF。**基于 CRDT 的 Redis企业版支持快照和 AOF 配置。但是,从 AOF 恢复比快照处理需要更多的内存并且效率更低。因此持久化的推荐做法是使用快照。

基于 CRDT 的 Redis企业版使您能够构建超快速,引人入胜的地理分布式应用程序。在本文中,我们介绍了四个应用 Redis CRDT 的用例：全局计数器、分布式缓存、共享会话存储和多区域数据摄取。Redis CRDT 使您能够专注于您的业务逻辑,而不必担心区域之间的数据同步。最重要的是,它们为引人入胜的应用程序提供了本地延迟,并且即使在数据中心之间出现网络故障时也保证了强大的最终一致性。

附录1: Redis CRDT 命令以及冲突解决 Redis企业版中的语义

有关提供基于 CRDT 的Active-Active地理分布的 Redis企业版架构的详细概述,欢迎联系我们。本附录列出了基于 CRDT 的 Redis企业版中可用的 Redis 数据结构和命令。

注意：对于多键操作,键必须映射到集群中的同一个插槽。这通常是通过大括号内的标签来完成的。使用 CRDT,哈希槽映射的工作方式与 Redis企业版集群相同。

数据类型	Redis CRDT 命令	合并/冲突解决语义
Counter	incr, incrby, incrbyfloat decr, decrby,	无冲突合并
	get	返回当前本地副本
	del	“添加/更新”胜过“删除”。并发删除将保留key,但将其value递减为零
String	set, append, getset, setex, psetex, setnx, setrange, 多键操作: mset, msetnx, rename, renamenx	最后一位写入者获胜
	get, getrange, mget	返回当前本地副本
	del	“添加/更新”胜过“删除”

数据类型	Redis CRDT 命令	合并/冲突解决语义
Set	sadd 多键操作: sdiffstore, sinterstore, sunionstore, smove	无冲突合并，add ()优先于 rmv () 注意:同时进行的移动操作可能会将同一个副本移动到多个集合中。例如,smove a b 1 smove a c 1 在不同的节点上同时执行,ltem 1 将在同步后在b 和 c 上可用
	smembers, sscan, sismemberby, scard, srandmember 多键操作: sinter, sunion, sdiff	对本地副本执行操作并返回结果
	del, spop, srem	“添加/更新”胜过“删除”
Hash	hset, hmset	无冲突合并，最后写入者获胜
	hincrby, hincrybyfloat	和Counter的工作原理一样。必须使用 hincrby 命令初始化。如果使用 hset 或 hmset 初始化key,则 hincrby 将无法使用该元素
	hget, hgetall, hkeys, hlen, hmget, hscan, hkeys, hvals, hstrln	对本地副本执行操作并返回结果
	hdel, delete	“添加/更新”胜过“删除”
Sorted Set	zadd 多键操作: zinterstore, zunionstore	无冲突合并
	zincrby	分数被视为计数器
	zcard, zcount, zrange, zrank, zrevrank, zrevrange, zrangebyscore, zrevrangebyscore, zrangebylex, zrevrangebylex, zlexcount, zscan, zscore	对本地副本执行操作并返回结果
	zrem, zremrangebylex, zremrangebyrank, zremrangebyscore	“添加/更新”胜过“删除”
List	lpush, rpush, linsert	无冲突合并
	lrange, lindex	对本地副本执行操作并返回结果
	lpop, rpop, blpop, brpop	返回本地副本。 注意：你可能会在不同的节点弹出相同的元素

基于CRDT的Redis企业版不支持的Redis数据结构

Geo
Hyperloglog
Bitfields 和 Bitmaps

CRDT设置中支持的Redis命令

Del (多键操作)
Echo
Exists
Expire (发生冲突时，TTL值最大的获胜)
Expireat
Keys (不推荐用于生产环境，改用scan)
Persist
Pexpire
Pexpireat
Ping
Pttl
slot
Sort (存储多键操作结果)
Scan
Touch
Ttl
Type
Unlink
Wait (可以使用，但仅具有局部效果)

(Pub/Sub)

Publish
Pubsub (显示本地集群的发布和订阅)
Psubscribe
Subscribe

Lua 脚本, multi-exec -- 只要key被映射到同一个 hashslot 就可以工作。 Lua 脚本默认处于命令复制模式而不是脚本复制模式。

CRDT设置中不支持的Redis命令

Dump
Object

(Strings)
Bitcount
Bitfield
Bitop
Bitpos
Getbit
Setbit

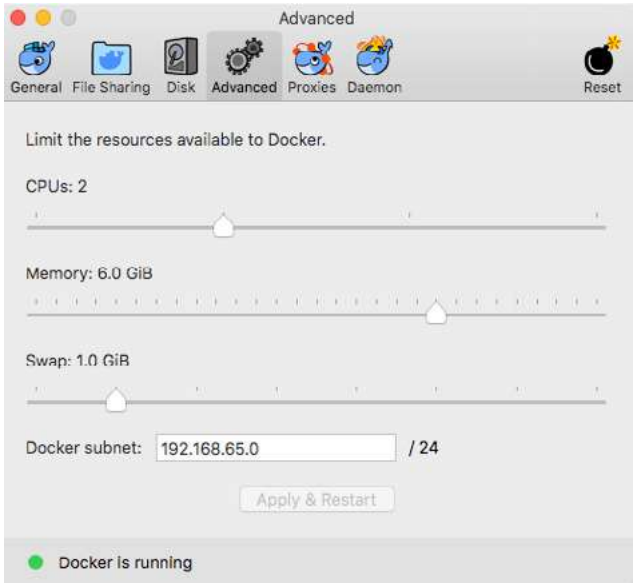
附录2：基于docker的开发
基于CRDT的Redis企业版环境

Redis企业版在Docker hub 上作为redis可用。您可以在 Redis 企业版文档页面找到有关何在 Doker 上设置 Redis企业版的详细分步说明。

作为开发人员或测试人员,您的首要工作是准备一个模拟您的生产的小型化开发环境。Docker 是开发人员中用于模拟其生产设置的流行平台。在这里,我们将引导您完成创建 Docker 环境的步骤-全部都是通过命令行来实现:

1. 创建一个数据库
    a. 创建一个三节点Redis企业版集群,每个节点位于单独的子网上
    b. 创建一个基于 CRDT的Redis企业版数据库
2. 连接到三个不同的实例
3. 验证你的设置
4. 拆分网络
5. 恢复连接

在开始之前,请确保为 docker 进程分配了足够的内存。您可以通过转到 Docker -> Preferences ->Advanced 来完成此操作。



1. 创建一个数据库

运行以下脚本以在 3 节点集群上创建基于 CRDT 的Redis企业版数据库。

```
#!/bin/bash
# 删除已经存在的桥接网络
 rm network1 2>/dev/null
```

```
docker network rm network2 2>/dev/null
docker network rm network3 2>/dev/null
# 创建新的桥接网络
echo "Creating new subnets..."
docker network create network1 --subnet=172.18.0.0/16 --gateway=172.18.0.1
docker network create network2 --subnet=172.19.0.0/16 --gateway=172.19.0.1
docker network create network3 --subnet=172.20.0.0/16 --gateway=172.20.0.1
# 启动3个docker容器。每个容器都是单独网络中的一个节点
# 这些命令从docker hub中拉取redis/redis。
# 基于redis端口映射规则, Redis 企业版实例在 12000, 12002, 12004 端口可用

echo ""
echo "Starting Redis Enterprise as Docker containers..."
docker run -d --cap-add sys_resource -h rp1 --name rp1 -p 8443:8443 -p 9443:9443 -p 12000:12000 --
network=network1 --ip=172.18.0.2 redis/redis
docker run -d --cap-add sys_resource -h rp2 --name rp2 -p 8445:8443 -p 9445:9443 -p 12002:12000 --
network=network2 --ip=172.19.0.2 redis/redis
docker run -d --cap-add sys_resource -h rp3 --name rp3 -p 8447:8443 -p 9447:9443 -p 12004:12000 --
network=network3 --ip=172.20.0.2 redis/redis

# 连接网络
docker network connect network2 rp1
docker network connect network3 rp1
docker network connect network1 rp2
docker network connect network3 rp2
docker network connect network1 rp3
docker network connect network2 rp3

# 节点启动时休眠。如果您的节点启动时间超过 60 秒,请增加休眠时间。
echo ""
echo "Waiting for the servers to start..."
sleep 60

# 创建3个 Redis企业版集群- 每个网络一个。
# 您可以以 https://localhost:8443/ (或 84458447)身份登录到集群。
#用户名为r@r.com,密码为password。
echo ""
echo "Creating clusters"
docker exec -it rp1 /opt/redis/bin/rladmin cluster create name cluster1.local user-name r@r.com password
test
docker exec -it rp2 /opt/redis/bin/rladmin cluster create name cluster2.local user-name r@r.com password
test
docker exec -it rp3 /opt/redis/bin/rladmin cluster create name cluster3.local user-name r@r.com password
test

# 创建CRDB
echo ""
echo "Creating a CRDB"
docker exec -it rp1 /opt/redis/bin/crdb-cli crdb create --name mycrdb --memory-size 512mb --port 12000 --
replication false --shards-count 1 --instance fqdn=cluster1.
```

```
local,username=r@r.com,password=test --instance fqdn=cluster2.local,username=r@r.
com,password=test --instance fqdn=cluster3.local,username=r@r.com,password=test
```

2.验证设置

在端口 12000、12002 和 12004 上运行 redis-cli 并验证您是否可以连接到所有 Redis企业版节点。

```
$ redis-cli -p 12000
127.0.0.1:12000> incr counter
(integer) 1
127.0.0.1:12000> get counter
"1"
```

3.拆分网络

在这里我们将每个节点的网络隔离开来。

```
#!/bin/bash
docker network disconnect network2 rp1
docker network disconnect network3 rp1
docker network disconnect network1 rp2
docker network disconnect network3 rp2
docker network disconnect network1 rp3
docker network disconnect network2 rp3
```

4.恢复连接

此脚本恢复节点之间的网络连接。

```
#!/bin/bash
docker network connect network2 rp1
docker network connect network3 rp1
docker network connect network1 rp2
docker network connect network3 rp2
docker network connect network1 rp3
docker network connect network2 rp3
```

HongKe 虹科

虹科电子科技有限公司
T(+86)400-999-3848
M(+86)155 2866 3362
www.hongcloudtech.com
hongcloudtech@hkaco.com

广州市黄埔区神舟路18号润慧科技园C栋6层
各分部：广州 | 成都 | 上海 | 苏州 | 西安 | 北京 | 台湾 | 香港 | 美国硅谷