文章编号:1671-4598(2020)04-0211-06 DOI:10.16526/j.cnki.11—4762/tp.2020.04.043 中图分类号:TP311.133.1 文献标识码:A

“异地多活”分布式存储系统设计和实现

李 丹,叶廷东
(广东轻工职业技术学院 信息技术学院,广州 510300)

摘要:随着互联网进入了大数据和云计算时代,分布式存储技术近年来受到了工业界和学术界的广泛关注;为了解决服务器压力过大、异地快速容灾和用户就近访问等问题,基于异地多活、分布式CAP理论,分析了分布式存储的特性、应用场景和技术挑战,进而利用开源的Redis NoSql数据库,RabbitMQ消息队列等技术,搭建了一个满足最终一致性,可用性和分区容忍性的“异地多活”分布式存储系统;通过对系统的压力测试结果表明,基本可以满足工业级应用的吞吐量和一致性的要求。

关键词:分布式存储;异地多活;最终一致性

Design and Implementation of a “Multi—active Across Data Centers” Distributed Storage System

Li Dan, Ye Tingdong
(School of Information Technology, Guangdong Vocational College of Light Industry, Guangzhou 510300, China)

Abstract: As the Internet has entered the era of big data and cloud computing, distributed storage technology has received widespread attention in industry and academia in recent years. In order to solve the problems of excessive server pressure, rapid disaster recovery in remote locations and users'proximity access, this paper analyzes the characteristics, application scenarios and technical challenges of distributed storage based on the theory of CAP and “Multi—active across data centers”, and then uses the open source Redis cache, RabbitMQ Message Queuing to setup a " Multi—active across data centers" distributed storage system that satisfies the ultimate consistency, availability and partition tolerance. Through the stress test results of the system, it can basically meet the throughput and consistency requirements of industrial applications.

Keywords: distributed storage system; live on distant data centers; eventually consistency

0 引言

随着人类进入移动互联网(互联网+)时代,新的业务形态层出不穷,应用程序对后台性能的要求也越来越高。传统的单机、单点、关系型数据库早已不能满足业界需要,因此很多公司把数据存储在分布式NoSql数据库中[1]。最简单的分布式存储,就是在单一机房,通过多服务器组成一个集群等模式来实现。

然而,为了更进一步缩短用户访问时间,一般让用户就近接入,所以分布式后台服务都跨机房部署,通过接入层的某种路由算法,可以把用户请求分配到就近的机房。这样可以大大缩短数据的传输距离,从而提升了用户体验。

而且,异地部署可以有容灾的好处,如果整个大片区的网络不通或者机房宕机,另外一个片区的机房可以无缝的接管所有的服务请求,从而提高整个后台服务的可靠性[2]。

伴随着后台业务服务的跨机房部署,存储也需要跨机房部署,怎么能使部署在不同机房存储中的数据保持同步,且不影响分布式服务的可用性,是一项巨大的挑战。本文从分布式系统的基础理论入手,分析了分布式存储的特性、应用场景和技术挑战,利用开源的Redis存储,RabbitMQ消息队列等技术,搭建了一个满足最终一致性,可用性和分区容忍性的“异地多活”分布式存储系统。

收稿日期:2019-10-08; **修回日期**:2019-10-26。
基金项目:广东“千百十工程”人才资助项目(RC2016—005);广州市产学研协同创新重大专项(201604020049)。
作者简介:李 丹(1979-),女,吉林长春人,硕士,讲师,主要从事计算机网络、云计算、分布式技术方向的研究。
叶廷东(1976-),男,江西赣州人,博士后,教授,主要从事计算机应用技术与物联网技术方向的研究。

1 异地多活的概念

异地多活数据中心是现在传统大型数据中心的发展趋势。“异地”相对于同城而言,一般不在同一城域;“多活”是区别于一个数据中心、多个灾备中心的模式,前者是多个数据中心都在运行中,所以称为“多活”,且互为备份。后者是一个数据中心投入运行,另外一个多个数据中心备份全量数据,平时处于不工作状态,只有在主机房出现故障的时候才会切换到备用机房。冷备份的主要问题是成本高,不跑业务,当主机房出问题的时候,也不一定能成功把业务接管过来[3]。

异地多活主要有如下好处[4]:

1)服务端离用户更近,接入层可以把用户请求路由到离用户最近的机房,减少了网络传输距离,大大提升了用户访问性能和体验。

2)异地快速容灾:如果整个机房(比如整个大区)挂掉或者网络瘫痪,另外一个异地机房能无缝单独提供服务,

用户完全无感知，大大提高了服务的可靠性。

2 分布式系统的 CAP 理论

在分布式的环境下，设计和部署系统时主要考虑下述 3 个重要的核心系统需求。

1）一致性（Consistency）：所有节点在同一时间具有相同的数据。

2）可用性（Availability）：保证对每个请求的成功或者失败都有响应。

3）分区容错性（Partition Tolerance）：分布式系统在遇到某节点或网络分区故障的时候，仍然能够对外提供满足一致性或可用性的服务。

上述 3 个重要的核心系统需求又简称为 CAP 需求[5]，在理论计算机科学中，CAP 定理（CAP theorem），又被称作布鲁尔定理（Brewer's theorem），它指出对于一个分布式计算系统来说，不可能同时满足以上三点，如图 1 所示，三个核心系统特性没有交叠的区域。即在构建分布式系统的时候，一致性，可用性，分区容忍性，这三者只可以同时选择两样[6]。

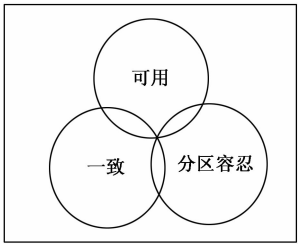

图 1 CAP 理论分布式系统的三特性

不幸的是，分区容错性是实际运营的分布式系统所必需的。设想下，谁能保证系统的各节点永远保持网络联通？一旦网络出现丢包，系统就不可用。而保证分区容错性最基本的要求是数据要跨数据中心存储。所以需要在一致性和可用性中二选其一。

为什么强一致性和高可用性不能同时满足？假如需要满足强一致性，就需要写入一条数据的时候，扩散到分布式系统里面的每一台机器，每一台机器都回复 ACK 确认后再给客户端确认，这就是强一致性。如果集群任何一台机器故障了，都回滚数据，对客户端返回失败，因此影响了可用性。如果只满足高可用性，任何一台机器写入成功都返回成功，那么有可能中途因为网络抖动或者其他原因造成了数据不同步，部分客户端独到的仍然是旧数据，因此，无法满足强一致性。

选择一致性，构建的就是强一致性系统，比如符合 ACID 特性的数据库系统。选择可用性，构建的就是最终一致性系统。前者的特点是数据落地即是一致的，但是可用性不能时时保证，这意思就是，有时系统在忙着保证一致性，无法对外界服务。后者的特点是时时刻刻都保证可用性，用户随时都可以访问，但是各个节点之间会存在不一致的时刻。

需要注意的是最终一致性的系统不是不保证一致性，而是不在保证可用性和分区容错性的同时保证一致性。最终我们还是要在最终一致性的各节点之间处理数据，使他们达到一致[7]。

3 异地多活分布式存储系统的挑战

众所周知，如果数据只是保存在单一节点，就没有一致性的问题；但是单机房存储连最基本的“分区容忍性”就保证不了。而对于“异地多活”系统而言，数据必然是跨数据中心存储的。保存在异地机房 NoSql 数据库中的数据要做到可用、并且尽可能一致，因为理论上，任何一个机房在任何时刻要给每一个用户提供服务，这就给分布式存储系统带来如下挑战[8]：

1）网络延迟 & 丢包：异地多活系统由于要跨数据中心存储，而跨数据中心的路途遥远导致的弱网络质量，数据同步是非常大的挑战；

2）一致性：用户可能几乎同时在两个机房写入数据，怎么保写入的数据不冲突并一致。

4 异地多活分布式存储系统的设计

鉴于异地多活系统的上述挑战，及分布式系统的 CAP 定理，本文设计的分布式存储系统满足了分区容错性和可用性，实现了最终一致性。

满足分区容错性，分布式数据在异地存储，任何一个机房挂掉或者跨区域网络不通，单机房可以立即提供服务。对分区错误的容忍性可以达到 100%。

满足可用性，分布式存储本地机房写成功后就返回给用户，不等待远端机房是否写成功。

为了满足最终一致性，引入消息中间件进行多地域数据分发，消息中间件可以确保消息不丢失[9]。并对写入的数据附上时间戳，通过时间戳的记录和比较机制，确保两边同时写入的数据不冲突。同时，引入一致性校验和补偿机制，数据最终一致性得到进一步的保证。

4.1 系统框架

如图 2 所示，在性能方面，由于 redis 的卓越表现，选择 redis 作为数据的承载[10]，在一个机房中部署多个 redis，组成一个集群，满足一个机房对数据的读写需求。为了解耦业务层和存储 redis，在 redis 之上引入一个 proxy 层，业务通过 proxy，按一定的 hash 策略访问 redis[11]。对于多机房分布高可用方面的需求，在 proxy 层实现数据在多机房间的互相同步机制，提供最终一致性。在多机房网络通信方面，数据同步以消息的形式发送。为了保证不丢消息，本文选择用 RabbitMQ 作为中间件发送。

4.2 实现方案

4.2.1 引入中间代理层 redisProxy 节点

业务进程通过 redisProxy 读写本地 redis。redisProxy 对 key 进行 hash，访问其对应的 redis。同时，redisProxy 节点做到无状态，按组管理，一个组内部署多个 redisProxy，组成集群。集群内的 redisProxy 可以方便地水平扩

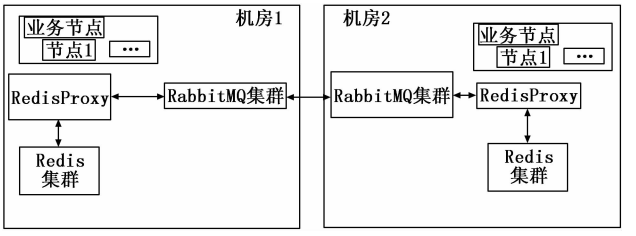

图 2　系统架构图

展，业务系统无感知。

业务进程通过配置文件指定需要访问哪个组的 redisProxy。另外，redisProxy 节点也对 redis 的读写情况、访问质量等做统计和监控。

4.2.2　写冲突问题的解决机制

本设计支持对数据的 put（写覆盖）操作，而支持对数据的写覆盖，必然带来写冲突的问题，即两个机房同时对同一个 Key 写入数据，因为时序问题，两个机房最终呈现的结果可能不一致。为了解决此问题，我们对存储在系统中的 key，维护一份元数据，目前维护的有 key 的版本，即 key 写操作的时间戳[13]。对 key 的写操作（插入、更新、删除），需要将操作发生的时间和本地记录的 key 对应的时间戳版本做比较，比版本更新（更晚）的操作将被执行，更旧（更早）的操作将被丢弃。

Redis 支持的数据类型比较丰富。除了最基本的 string 类型，通过上述方法，能够直接支持外，对 set、sorted set、hash 结构，通过做一些转换，也能够支持。转换方式如下：为 set、sorted set、hash 结构中存储的每个成员维护一份时间戳版本，对 key 做写操作时，需要对操作涉及的每个成员做时间戳比较，以决定是执行还是放弃。基于时间戳版本的写操作流程如图 3 所示。

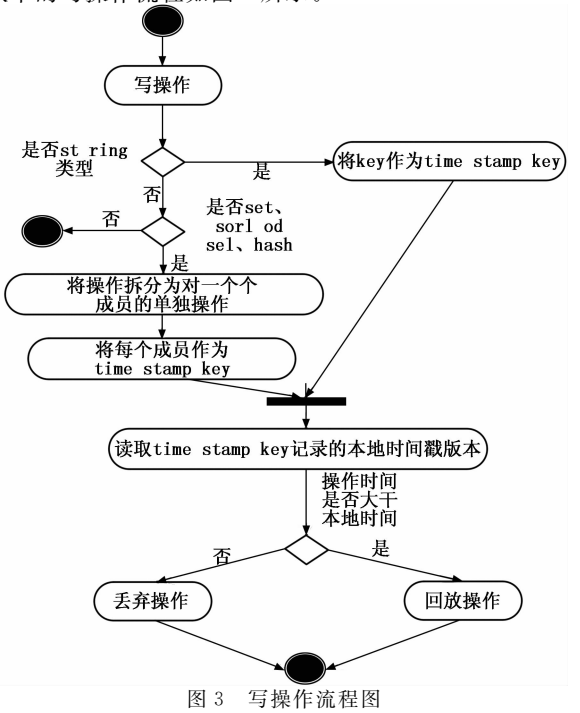

图 3　写操作流程图

时间戳机制能够工作的一个前提是服务器之间同步系统时间。一般线上服务器都有同步系统时间，机器之间系统时间误差一般不超过 1 s，为毫秒（ms）级别。这个能满足互联网生产环境对存储系统最终一致性的要求。同时，为了减少时间冲突，对一个 key 的读写，我们 hash 到一台机器上执行。

4.2.3　引入第三方消息队列，增强同步消息传递的可靠性

redisProxy 需要保证带有时间戳的写操作能够同步到其他组。为了增加同步消息的可靠性，本设计通过引入一个第三方队列来满足对同步的可靠性要求。RabbitMQ 是我们本文选定的方案，RabbitMQ 实现了高级消息队列协议（AMQP），RabbitMQ 消息中间件有着完善的可靠性机制并且使用方便[14]。通过 RabbitMQ 对同步消息的持久化、集群部署及 mirrored queue 等机制，实现写操作的可靠同步。

4.2.4　平滑的升级扩容机制

为实现升级扩容时部署一个新的组，我们利用 redis 的主从同步获取‘旧’的最近未更新的数据，利用 RabbitMQ 的同步获取‘新’的最近变化的操作。通过两者的结合，使新组的数据与已有组一致。

4.3　系统节点

4.3.1　业务节点（redis client）

App 业务进程，由配置文件指定通过哪个组的 redisProxy 访问存储系统。对 redisProxy 的访问，通过轮询的方式来均衡负载。为了接口使用方便友好，redis client 提供类似于 redis 的接口。

4.3.2　redisProxy

访问代理层，负责对 redis 读写访问。对外提供网络协议接口访问存储系统。底层存储用 redis，将数据存在内存中。内部对数据按 key 进行 hash 分片，每片存储在一个 redis 中。写操作带上时间戳，通过 RabbitMQ 同步到其他组。写操作成功发送到 RabbitMQ 即认为同步成功，返回。因此，状态系统各集群间实现的是最终一致性。

4.3.3　RabbitMQ

第三方消息队列，负责将 redisProxy 的写操作可靠地同步到其他组。通过配置，将同步队列持久化，防止 RabbitMQ 重启后消息丢失[15]。一套状态系统内，部署多个 RabbitMQ，组成集群，防止 RabbitMQ 单点失败。配置 mirrored queue，使同步队列在集群中有多个镜像，进一步提高可靠性。

4.3.4　redis

＜key，value＞类型的数据存储节点。数据不持久化，只存在内存中。为配合数据分布，一个集群中部署多个 redis 节点。

4.4　交互流程

4.4.1　读数据流程

读数据流程如图 4 所示。

1）业务节点发消息给 RedisProxy 节点，此消息是基于 TCP 的网络消息，由业务自定义。

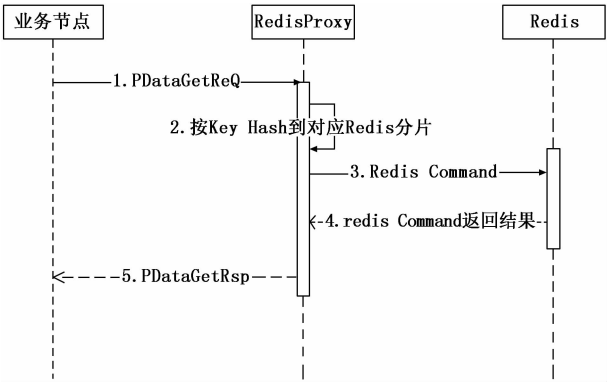

图 4 读数据流程图

2）RedisProxy 节点收到业务读请求后，按照 Key Hash 到某个 Redis 本地分片。

3）执行标准的 Redis 命令，从本地 redis 读取数据。

4）标准 Redis 命令的回包。

5）业务收到 Redis 命令的回复后，返回给业务节点一个回包（此回包也是由业务定义的基于 TCP 的网络消息）

4.4.2 写数据流程

写数据流程如图 5 所示。

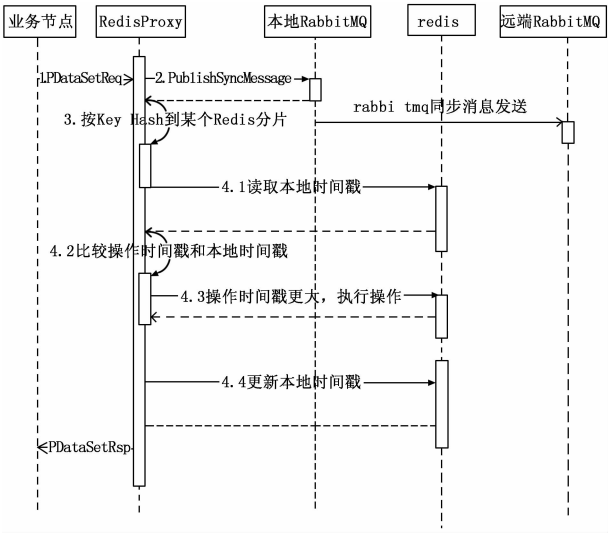

图 5 写数据时序图

1）业务节点发消息给 RedisProxy 节点，此消息是基于 TCP 的网络消息，由业务自定义。

2）RedisProxy 把写操作请求同步给 RabbitMQ；

3）RedisProxy 节点收到业务读请求后，按照 Key Hash 到某个 Redis 本地分片。

4）步骤 4.1、4.2、4.3、4.4 为一个事务操作，通过比较操作的时间戳和本地保存的时间戳来决定是否执行本次操作，以避免写冲突，确保两边数据一致。

4.4.3 同步远端数据流

同步远端数据流如图 6 所示。

1）本地 RabbitMQ 从远端 RabbitMQ 收到写同步消息；

2）推送同步消息到本机房的 RedisProxy 节点；

3）RedisProxy 节点处理推送过来的写请求，按 key Hash 到某个本地 Redis 分片（步骤 3.1）；步骤 3.2、3.3、3.4、3.5 是一个事务操作，通过比较操作的时间戳和本地保存的时间戳大小来决定是否执行本次操作，以避免写冲突，确保两边数据一致。

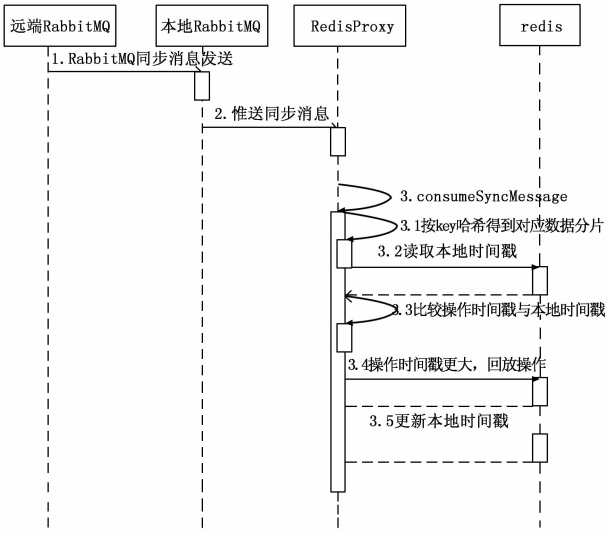

图 6 同步远端数据

4.5 容灾设计

4.5.1 redisProxy 服务器宕机

redisProxy 多台机集群化部署提高可用性。如果某台服务器宕机，其 redisProxy 服务器可以继续提供服务。

4.5.2 RabbitMQ 服务器宕机

同步消息队列持久化，同时 RabbitMQ 在多台机集群化部署，同步消息在集群中有多个镜像。如果某台服务器宕机，集群中的其他 RabbitMQ 可以继续提供服务。宕机恢复后的 RabbitMQ，追赶上其他 RabbitMQ 后可以继续提供服务。

4.5.3 Redis 服务器宕机

当做整个集群不可用，切换到另一个状态系统集群。当机器恢复后，按升级扩容的策略对待，重新部署该组状态系统。等追赶上其他集群后，可开始对外提供服务，将业务流量切换到本集群。

4.5.4 集群机房网络不可用

切换业务流量到其他集群，继续提供服务。等机房恢复后，通过 RabbitMQ 获取其他集群中存储着的同步消息，本地回放，追赶数据。同步队列处理完，即已追赶上其他集群，此时可将业务流量切换回，对外提供服务。

4.6 升级扩容

如图 7 所示，升级扩容的步骤如下：

1）在新机房部署 redis 从库，同步现有机房的数据。

2）在新机房部署 RabbitMQ，同步并存储更新操作。

3）在 redis 主库中写入一测试数据，测试从库是否同步上。

4）第三步中的测试数据，如果已同步到从库，将从库

提升为主库，同时启动 redisProxy，开始回放 RabbitMQ 中的同步消息。

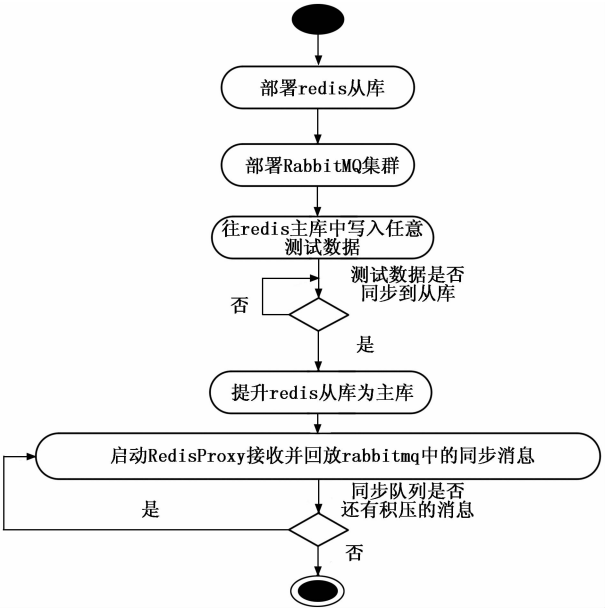

图 7　回放同步消息

4.7　一致性校验和补偿

通过脚本，定期随机抽取一批 key，比较在各集群之间的数据是否一致。如发现不一致，人工介入校正，根据业务特性做一致性补偿等措施。

5　模拟性能测试

用 C++编写模拟测试代码访问 redisProxy 对系统进行性能测试，跨机房部署，分别从两个机房对多个 key 进行读写。用 Redis－set、Redis－get 分别表示 Redis 写、Redis 读命令。通过代码分别统计出 redisProxy 的平均处理时延，吞吐量，和数据一致性时延等指标。

环境搭建：异地两个机房 RedisProxy 部署在单台服务器，CPU：E5－2620/2.10 GHz，内存：32G，操作系统 ubuntu 12.04。异地两个机房，分别部署 4 个 Redis 实例，Key－Value 通过键名 hash 到不同的 redis 实例。

实验 1：测试系统的吞吐量和系统处理时延。

分别以每秒 1 个、100 个、1 000 个，5 000 个、10 000 个读写不同的 key 请求，逐步增大系统的请求数，观察系统的 CPU 指标和系统延时。

结果分析：如图 8 和图 9 所示，系统时延和 CPU 随着请求数增加而缓慢增加。当请求数接近达到 1 万/秒时，系统延时和 CPU 利用率明显增大，所以单 redisProxy 进程部署的 QPS 接近 1 万。

实验 2：在保持系统 80% QPS 情况下，对某一个 key 反复进行写操作，写完之后分别在两个机房实时读，记录两个机房不同数据结构数据一致的平均时间差。

结果分析：如图 10 所示，实验结果可以看出，不同数据结构的数据同步时间有略微差别，但是都在 1s 以内，可以满足工业级数据同步要求。

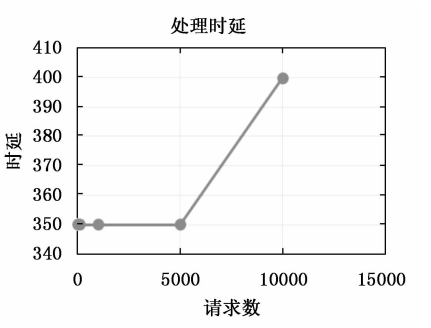

图 8　系统处理时延与请求数的关系

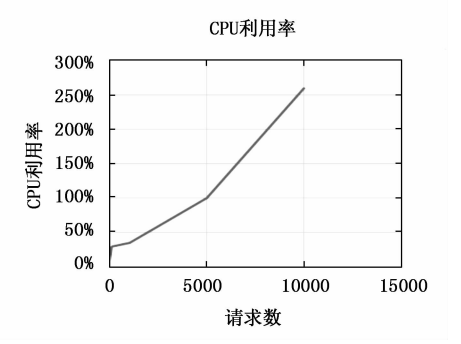

图 9　CPU 利用率与请求数的关系

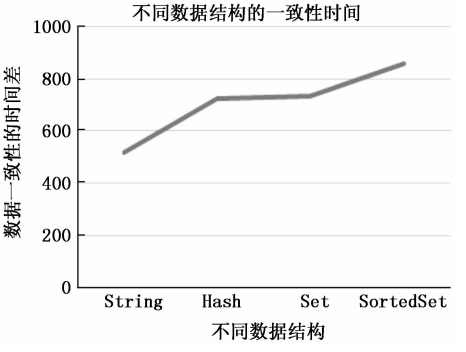

图 10　系统一致性时间

6　结束语

跨机房的存储设计是业界的难点，根据 CAP 理论，根本做不到同时满足一致性要求和可用性的系统。本文根据分布式存储应用的特点，选择了满足可用性和最终一致性。通过多机集群，异地部署等保证可用性；通过开源的可靠消息队列技术和定时一致性校验 & 补偿技术来确保最终一致性。经过压测，系统基本能满足工业级互联网应用吞吐量和可用性的要求。NoSql 技术、Web 应用的规模和使用模式的发展，为分布式存储和相关领域的发展带来了新的契机。近年来，业界开始研究 CRDT（Conflict－Free Replicated Data Type）数据结构，CRDT 是各种基础数据结构最终一致算法的理论总结，能根据一定的规则自动合并，解决冲突，达到强最终一致的效果[16]。这可以作为本课题的

以后研究方向。

参考文献：

[1] 杜丽娟. 关系型数据库与 NoSQL 数据库的性能对比 [J]. 智能计算机及应用，2017，7 (3)：132－134.

[2] Rajagopalan S，et al. SecondSite：disaster tolerance as a service [A]. Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments [C]. 2012：97－108.

[3] 丁建立，王斌强，张 超. 异地双活数据中心服务中心划分优化 [J]. 计算机应用与软件，2016，33 (2)：31－32，54.

[4] 许艺蓝. “饿了么”早餐配置系统的设计与实现 [D]. 北京：北京交通大学，2018.

[5] Brewer E. Towards robust distributed Systems [A]. Proc. 19th Ann. ACM Symp. Principles of Distributed Computing (PODC 00) [C]. ACM，2000：7－10.

[6] 陈 明. 分布系统设计的 CAP 理论 [J]. 计算机教育，2013，15：109－110.

[7] Brewer E. Towards robust distributed systems [A]. Proc. 19th Ann. ACM Symp. Principles of Distributed Computing (PODC 00) [C]. ACM，2000：7－10.

[8] Ajoux P，Bronson N，Kumar S，et al. Challenges to adopting stronger consistency at scale [A]. Proceeding of the 15th Workshop on Hot Topics in Operating Systems [C]. HotOS' 15，2015.

[9] 陈 明，潘家铭，阎保平. 消息中间件的设计与实现 [J]. 微电子学与计算机，2005，22 (4)：4－7.

[10] 柳皓亮. Redis 集群性能测试分析 [J]. 微型机与应用，2016 (10)：70－71，78.

[11] 闫 明. 高可用可扩展集群化 Redis 设计与实现 [D]. 西安：西安电子科技大学，2014.

[12] 李知杰. 基于 AMQP 的异步通信实现及其在 OpenStack 项目中的应用 [J]. 软件导刊，2013，12 (7)：35－37.

[13] 何远标，乐小虬，等. 基于日志的泛在个人数据同步方法研究 [J]. 现代图书情报技术，2013，29 (10)：8－14.

[14] 梁彦杰，廉东本. 基于消息中间件的数据交换平台传输框架设计 [J]. 计算机系统应用，2012，21 (4)：10－13.

[15] 唐素纯，李 宁，等. 面向海上战术云的信息资源服务架构设计 [J]. 舰船电子工程，2019 (2)：103－105.

[16] Shapiro M，Preguiça N，Baquero C，et al. Conflict－Free replicated data types [A]. 13th International Symposium on Stabilization，Safety，and Security of Distributed Systems (SSS) [C]. 2011：386－400.

(上接第 210 页)

入 Isabel 时变飓风数据集进行了案例测试，测试表明，该系统能够支持体数据的实时绘制和交互，并能够流畅地进行动画演示，帧速率可达到 50 FPS 以上。

然而本系统依然存在着不足之处，主要包括：

1) 当系统并发访问数量增加，系统的并发性难以维持，会出现系统卡顿，假死等问题；

2) 维度压缩会损失数据的精度，并且针对不同规模的时变体数据，跳跃采样的步长值无法自适应取值，缺乏灵活性。

因此在未来的工作中，将主要围绕一下内容进行研究：

1) 系统的并发性，针对大规模地访问数量，系统能够有效地保持稳定性；

2) 更加有效的体绘制方法，在体绘制方法上进行研究，改进绘制方法，提升绘制质量；

3) 针对时变体数据，研究自适应跳跃步长，根据不同的体数据，自适应进行维度压缩。

参考文献：

[1] 易玮玮，陈子轩，徐泽楷，等. 一种基于 CPU 的三维超声图像体绘制方法 [J]. 生命科学仪器，2019，17 (1)：32－36，31.

[2] 唐占红，於时才. 面绘制三维重建原理及其改进算法研究 [J]. 计算机工程与设计，2009，30 (9)：2225－2228.

[3] Krüger J，Westermann R. Acceleration techniques for GPU－based volume rendering [J]. Proc. IEEE Visualization Oct，2003：287－292.

[4] 黄文博，燕 杨. C/S 结构与 B/S 结构的分析与比较 [J]. 长春师范大学学报，2006，25 (8)：56－58.

[5] 王 娴，刘 辉，倪远平. B/S 与 C/S 体系结构的应用研究 [J]. 信息技术，2006，30 (6)：53－55.

[6] Resch B，Wohlfahrt R，Wosniok C. Web－based 4D visualization of marine geo－data using WebGL [J]. Cartography and Geographic Information Science，2014，41 (3)：235－247.

[7] Westover L. Interactive volume rendering [A]. Proc Chapel Hill Workshop On visualization [C]. 1989.

[8] 刘兆明，郭 景，柯永振. 面向 Web 的远程图像可视化系统的研制 [J]. 计算机应用与软件，2018，35 (06)：196－202，268.

[9] 张 辰. B/S 模式下的数据可视化技术研究及其应用 [D]. 北京：北京邮电大学，2014.

[10] 高 鹏，王 田. B/S 结构下基于 Socket 和 VTK 的 3D 交互方法 [J]. 中国数字医学，2012，7 (4)：75－77.

[11] 雷 辉，赵 颖，王铭军，等. 面向浏览器的医学影像可视化系统 [J]. 中国图象图形学报，2018，20 (4)：491－498.

[12] 侯晓帅. 基于 Web 架构的医学影像三维可视化处理研究 [D]. 中国科学院研究生院 (上海技术物理研究所)，2016.

[13] Cacioppo S，Frum C，Asp E，et al. A quantitative meta－analysis of functional imaging studies of social rejection [J]. Scientific reports，2013，3 (1)：1－3.

[14] Haehn D. Slice：drop：collaborative medical imaging in the browser [A]. Acm Siggraph Computer Animation Festival [C]. ACM，2013.

[15] Wu K，Knoll A，Isaac B，et al. Direct multifield volume ray casting of fiber surfaces [J]. IEEE Transactions on Visualization and Computer Graphics，2016：1－1.