ALIBABA CLOUD

阿里云

云数据库 Redis 版
性能白皮书

文档版本：20220712

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.社区版（集群版-双副本）

1.1. 测试环境

本章节介绍云数据库Redis集群版-双副本的性能测试环境。

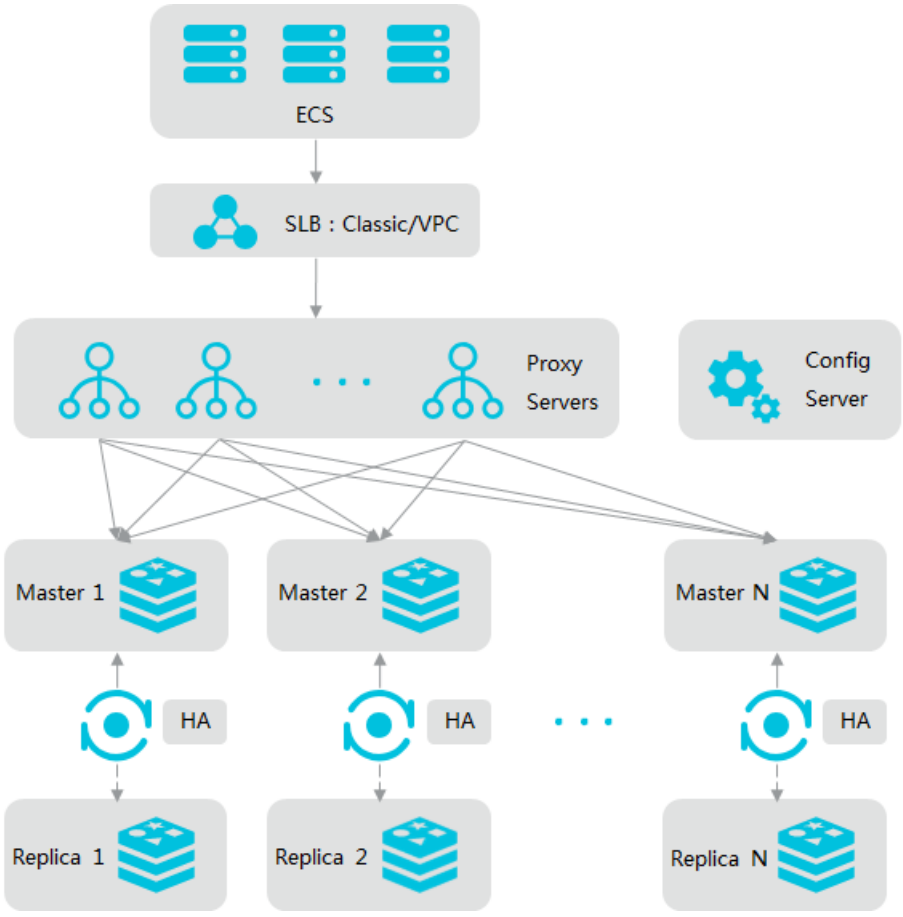
产品架构

云数据库Redis集群版-双副本实例由配置服务器、代理服务器和分片服务器组成：

- 配置服务器（Config Server）是采用双副本高可用架构的集群管理工具，用于存储集群配置信息及分区策略；
- 代理服务器（Proxy Server）为单节点架构，集群版结构中会有多个proxy，系统自动对所有proxy进行负载均衡及故障转移；
- 分片服务器（Shard Server）同样采用双副本高可用架构，主节点故障之后，系统会自动进行主备切换保证服务高可用。

Redis集群版本实例提供一个访问域名供客户端访问，您可以通过该域名进行正常的Redis访问及数据操作，代理服务器、分片服务器和配置服务器均不支持单独的直接访问。

客户端通过负载均衡（SLB）连接代理服务器，通过代理服务器对分片服务器进行访问，其架构图如下。



地域与可用区

所有测试均在华东1（杭州）地域的可用区E完成。

ECS配置

测试环境中的ECS配置根据作为测试对象的Redis版本不同而有所区别。下文以128G集群版和64G集群版的测试为例。

测试128G集群版时的ECS配置：

- 4 vCPU，8G内存的ECS 10台；
- 12 vCPU，48G内存的ECS 4台；
- 网络类型：VPC；
- 操作系统：CentOS 6.0，64位。

测试64G集群版时的ECS配置：

- 4 vCPU，8G内存的ECS 10台；
- 网络类型：VPC；
- 操作系统：CentOS 6.0，64位。

Redis实例配置

- Redis实例的规格根据测试对象决定。
- 本白皮书中使用Redis 2.8版本的实例进行基准测试，4.0版本测试结果与其相似。

1.2. 测试工具

本节介绍测试Redis集群版使用的工具。

memtier_benchmark简介

memtier_benchmark是Redis Labs推出的命令行工具，可用于在键值存储数据库中生成数据负载并进行压力测试。

下载与安装

详细步骤请参见memtier-benchmark使用方法。

1.3. 测试命令与指标

本节介绍测试云数据库Redis集群版使用的命令和结果指标。

测试命令

```
./memtier_benchmark -s r-***************.redis.rds.aliyuncs.com -p 6379 -a <password> -c 20
-d 32 --threads=10 --ratio=1:1 --test-time=1800 --select-db=10
```

> **说明** 命令说明请参见memtier_benchmark常用选项说明。

测试指标

QPS

Queries Per Second，即数据库每秒处理的请求数。

1.4. 测试结果

本节介绍云数据库Redis版的性能测试结果。

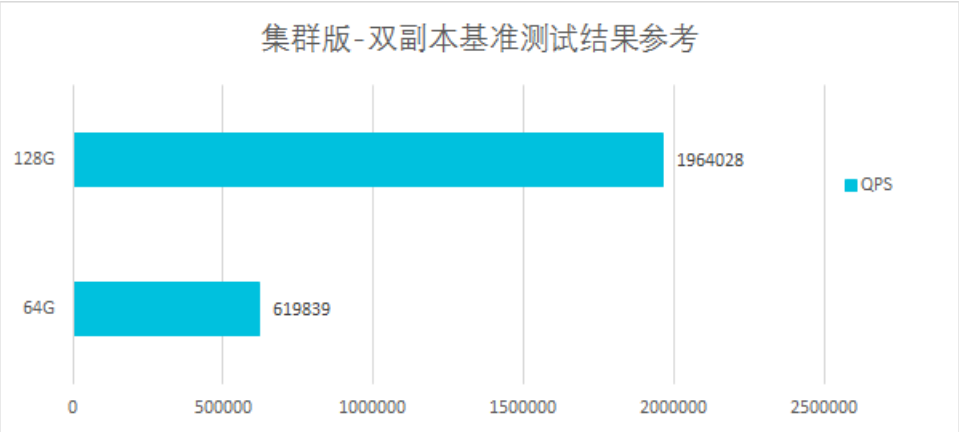
集群版-双副本规格

规格	节点数（个）	连接数上限（个）	内网带宽上限（MB）	CPU 处理能力	QPS参考值	说明
4 GB集群版	2	20000	96	2核	160000	高性能集群实例
8 GB集群版	2	20000	96	2核	160000	高性能集群实例
16 GB集群版	8	80000	384	8核	640000	高性能集群实例
32 GB集群版	8	80000	384	8核	640000	高性能集群实例
64 GB集群版	8	80000	384	8核	640000	高性能集群实例
128 GB集群版	16	160000	768	16核	1280000	高性能集群实例
256 GB集群版	16	160000	768	16核	1280000	高性能集群实例
512 GB集群版	32	320000	1536	32核	2560000	高性能集群实例
1 TB集群版	64	640000	2150	64核	5120000	高性能集群实例
2 TB 集群版	128	1280000	4300	128核	10240000	高性能集群实例

> 说明
>
> - 集群版-双副本规格中的节点数为主节点数量。
> - 其它版本的规格信息请参见云数据库Redis版规格性能。

测试结果

下图展示了云数据库Redis集群版-双副本部分规格的基准测试结果。



2.企业版（性能增强型）

2.1. 测试环境

本文介绍Redis企业版（性能增强型）性能测试所使用的具体环境。

产品概述

Redis企业版（性能增强型）适合并发量大、读写热点多、对性能要求极高的场景。相比云数据库Redis社区版，性能增强型主要在下述方面进行了优化：

- 采用多线程模型，性能约为同规格社区版实例的3倍。
- 提供多种增强型数据结构模块（modules），包括TairString（含CAS和CAD命令）、TairHash、TairGIS、TairBloom、TairDoc、TairTS、TairCpc、TairZset、TairRoaring和TairSearch，业务无需再关心存储的结构和时效性，极大提升业务开发效率。

更多介绍，请参见性能增强型。

测试环境

测试环境信息	说明
地域和可用区	所有测试均在华北2（北京）地域的可用区G中完成。
Redis实例架构	标准版（双副本）架构，详情请参见标准版-双副本。
部署压测工具的机器	云服务器ECS实例，规格为ecs.g5ne.16xlarge，详情请参见实例规格族。
性能增强型实例规格	由于测试结果受规格影响较小，本次测试以redis.amber.master.large.multithread规格为例，规格详情请参见性能增强-标准版。

2.2. 测试工具

Redis企业版（性能增强型）使用redis-benchmark工具进行性能测试，本文为您使用该工具的相关注意事项。

采用开源Redis的redis-benchmark工具进行压测，它是Redis官方的性能测试工具，可以有效地测试Redis服务的性能。本次测试使用Redis官方最新的代码进行编译，详情请参见Redis开源项目。

> **说明** 为确保支持redis-benchmark工具的--threads参数，自测时请选择6.0及以上的Redis版本进行编译。

2.3. 测试方法

本文介绍Redis企业版（性能增强型）性能测试的具体方法。

测试命令

本次测试主要使用redis-benchmark进行以下两个场景的压测：

- 启动16个线程、256个连接、100000个Key的取值范围来测试SET命令执行性能。

```
./redis-benchmark -h r-bp1s02ae14mr****.redis.rds.aliyuncs.com -p 6379 -a testaccount:Rp829dlwa -n 3000000 - r 100000  -c 256 -t set -d 64 --threads 16
```

- 启动16个线程、256个连接、100000个Key的取值范围来测试GET命令执行性能。

```
./redis-benchmark -h r-bp1s02ae14mr****.redis.rds.aliyuncs.com -p 6379 -a testaccount:Rp829dlwa -n 3000000 - r 100000 -c 256 -t get -d 64 --threads 16
```

参数说明

参数	说明
-h	Redis实例的内网连接地址。具体操作，请参见查看连接地址。
-p	Redis实例的服务端口，默认为6379。
-a	Redis实例的密码。 **说明** 如果使用通过控制台创建的账号连接Redis，连接密码格式需为 `<user>:<password>` 。例如，账号名为admin，密码为Rp829dlwa，则连接Redis时的密码为 `admin:Rp829dlwa` 。
-c	并发的连接数量。
-n	测试的总请求数量，可设置较大的值以持续压测。
-t	测试的命令，例如GET、SET等。
-d	SET或GET所操作的值的数据大小，单位为字节（Byte）。
-r	使用的Key的随机范围，即使用多少个不同key。
--threads	启动多线程压测，并指定线程个数。

说明 参数的更多信息，请参见redis-benchmark。

2.4. 测试结果

本文介绍Redis企业版（性能增强型）在不同测试场景下的测试结果。

说明 由于写性能受主从同步、AOF刷新方式、审计日志是否开启等多种因素影响，一般来说测试得出的读性能会强于写性能。

测试指标

测试指标	说明
QPS	每秒处理的读写操作数，单位为次/秒。

测试指标	说明
Latency	操作的平均延迟分布，单位为毫秒（ms）。例如 `70.33% <= 0.6 milliseconds` ，表示70.33%的操作在0.6毫秒内得以处理。

GET测试结果

Value长度	QPS（次/秒）	Latency（毫秒）
64字节	444,115.47	0.00% <= 0.1 milliseconds 0.01% <= 0.2 milliseconds 0.05% <= 0.3 milliseconds 3.10% <= 0.4 milliseconds 31.02% <= 0.5 milliseconds 70.33% <= 0.6 milliseconds 99.67% <= 0.7 milliseconds
128字节	435,276.94	0.00% <= 0.1 milliseconds 0.01% <= 0.2 milliseconds 0.04% <= 0.3 milliseconds 0.67% <= 0.4 milliseconds 18.28% <= 0.5 milliseconds 57.34% <= 0.6 milliseconds 86.72% <= 0.7 milliseconds 98.58% <= 0.8 milliseconds 99.88% <= 0.9 milliseconds
256字节	427,960.03	0.00% <= 0.1 milliseconds 0.02% <= 0.2 milliseconds 0.04% <= 0.3 milliseconds 1.38% <= 0.4 milliseconds 24.10% <= 0.5 milliseconds 62.43% <= 0.6 milliseconds 86.83% <= 0.7 milliseconds 98.13% <= 0.8 milliseconds 99.86% <= 0.9 milliseconds

Value长度	QPS（次/秒）	Latency（毫秒）
1024字节	428,265.53	0.00% <= 0.1 milliseconds 0.02% <= 0.2 milliseconds 0.06% <= 0.3 milliseconds 3.92% <= 0.4 milliseconds 27.29% <= 0.5 milliseconds 62.92% <= 0.6 milliseconds 85.32% <= 0.7 milliseconds 96.47% <= 0.8 milliseconds 99.52% <= 0.9 milliseconds

SET测试结果

Value长度	QPS（次/秒）	Latency（ms）
64字节	218,086.66	0.00% <= 0.1 milliseconds 0.00% <= 0.2 milliseconds 0.01% <= 0.3 milliseconds 0.01% <= 0.4 milliseconds 0.02% <= 0.5 milliseconds 0.04% <= 0.6 milliseconds 0.07% <= 0.7 milliseconds 0.09% <= 0.8 milliseconds 0.13% <= 0.9 milliseconds 0.49% <= 1.0 milliseconds 16.20% <= 1.1 milliseconds 81.83% <= 1.2 milliseconds 98.74% <= 1.3 milliseconds 99.74% <= 1.4 milliseconds

Value长度	QPS（次/秒）	Latency（ms）
128字节	206,825.23	0.00% <= 0.1 milliseconds 0.00% <= 0.2 milliseconds 0.00% <= 0.3 milliseconds 0.01% <= 0.4 milliseconds 0.01% <= 0.5 milliseconds 0.03% <= 0.6 milliseconds 0.05% <= 0.7 milliseconds 0.08% <= 0.8 milliseconds 0.14% <= 0.9 milliseconds 0.20% <= 1.0 milliseconds 1.08% <= 1.1 milliseconds 36.09% <= 1.2 milliseconds 93.36% <= 1.3 milliseconds 99.17% <= 1.4 milliseconds
256字节	203,086.92	0.00% <= 0.1 milliseconds 0.00% <= 0.2 milliseconds 0.00% <= 0.3 milliseconds 0.01% <= 0.4 milliseconds 0.01% <= 0.5 milliseconds 0.02% <= 0.6 milliseconds 0.04% <= 0.7 milliseconds 0.05% <= 0.8 milliseconds 0.08% <= 0.9 milliseconds 0.13% <= 1.0 milliseconds 1.01% <= 1.1 milliseconds 26.12% <= 1.2 milliseconds 91.45% <= 1.3 milliseconds 99.16% <= 1.4 milliseconds

Value长度	QPS（次/秒）	Latency（ms）
1024字节	184,547.23	0.00% <= 0.1 milliseconds 0.00% <= 0.2 milliseconds 0.00% <= 0.3 milliseconds 0.00% <= 0.4 milliseconds 0.01% <= 0.5 milliseconds 0.02% <= 0.6 milliseconds 0.03% <= 0.7 milliseconds 0.05% <= 0.8 milliseconds 0.07% <= 0.9 milliseconds 0.09% <= 1.0 milliseconds 0.12% <= 1.1 milliseconds 0.30% <= 1.2 milliseconds 11.28% <= 1.3 milliseconds 75.67% <= 1.4 milliseconds 97.70% <= 1.5 milliseconds 99.41% <= 1.6 milliseconds

3.企业版（持久内存型）

3.1. 测试环境

本文介绍Redis企业版（持久内存型）性能测试所使用的具体环境。

产品概述

Redis企业版（持久内存型）基于Intel 傲腾™数据中心级持久内存，为您提供大容量、兼容Redis的内存数据库产品。单实例成本对比Redis社区版最高可降低30%，且数据持久化不依赖传统磁盘，保证每个操作持久化的同时提供近乎Redis社区版的吞吐和延时，极大提升业务数据可靠性。更多详情请参见持久内存型。

测试环境

测试环境信息	说明
地域和可用区	所有测试均在华北3（张家口）地域的可用区A中完成。
Redis实例架构	标准版（双副本）架构，详情请参见标准版-双副本。
部署压测工具的机器	云服务器ECS实例，规格为ecs.g6e.8xlarge，详情请参见实例规格族。
持久内存型实例规格	本次测试以tair.scm.standard.32m.128d规格为例，规格详情请参见持久内存型规格。

3.2. 测试工具

Redis企业版（持久内存型）使用YCSB压测工具进行性能测试，本文为您介绍测试工具及其使用方法。

采用开源社区的YCSB压测工具进行压测。YCSB是一款Java编写的支持多种数据库的性能测试工具，具体安装和使用方法请参见YCSB。

由于开源版本仅支持测试Hash结构的数据，下载YCSB工具后，需要将*YCSB/redis/src/main/java/site/ycsb/db/RedisClient.java*文件替换为RedisClient.java文件，用于测试String结构的数据。

3.3. 测试方法

本文介绍Redis企业版（持久内存型）性能测试的具体方法。

工作负载

测试的总数据量8 GB，数据分布方法为zipfian，具体测试场景如下：

- Load：100%的写操作。
- Workload C：100%的读操作。
- Workload A：50%的更新操作与50%的读操作。

关于Workload的详细介绍，请参见Core Workloads。

> **说明** 由于应用场景不同，Set、Zset中的复杂命令无法进行通用测试，您可以根据您的业务场景自行测试。

测试命令

```
#加载数据
workload=a
./bin/ycsb load redis -s -P workloads/workload${workload} -p "redis.host=${server_ip}" -p
"redis.port=${port}" -p "redis.password=${password}" -p "recordcount=${recordcount}" -p "op
erationcount=${operationcount}" -p "redis.timeout=30000" -p "redis.command_group=${command_
group}" -p "fieldcount=${fieldcount}" -p "fieldlength=${fieldlength}" -threads ${threads}
-p "redis.password=***:*****"
#运行Workload C
workload=c
./bin/ycsb run redis -s -P workloads/workload${workload} -p "redis.host=${server_ip}" -p  "
redis.port=${port}" -p "redis.password=${password}" -p "recordcount=${recordcount}" -p "ope
rationcount=${operationcount}" -p "redis.timeout=30000" -p "redis.command_group=${command_g
roup}" -p "fieldcount=${fieldcount}" -p "fieldlength=${fieldlength}" -threads ${threads} -p
"redis.password=***:*****"
#运行Workload A
workload=a
./bin/ycsb run redis -s -P workloads/workload${workload} -p "redis.host=${server_ip}" -p  "
redis.port=${port}" -p "redis.password=${password}" -p "recordcount=${recordcount}" -p "ope
rationcount=${operationcount}" -p "redis.timeout=30000" -p "redis.command_group=${command_g
roup}" -p "fieldcount=${fieldcount}" -p "fieldlength=${fieldlength}" -threads ${threads} -p
"redis.password=***:*****"
```

参数说明

参数	说明
server_ip	Redis实例的IP地址。
port	Redis实例的服务端口。
password	根据选取账号的不同，密码的填写格式有一定区别： • • 新创建的账号：密码格式为 `<user>:<password>` 。例如自定义账号为 `testaccount` ，密码为 `Rp829dlwa` ，密码需填写为 `testaccount:Rp829dlwa` 。 > **说明** 如果忘记密码，请参见修改或重置密码。
recordcount	数据装载阶段准备的数据量。
operationcount	执行操作的数据量。

参数	说明
command_group	测试的数据结构，本案例中，依次测试下述数据结构： • string • hash • list • set • zset
fieldcount	字段或元素个数，本案例中，String数据结构配置为1，其他数据结构配置为10。
fieldlength	值长度，根据测试需求配置。
threads	YCSB线程数，根据实例规格配置。

3.4. 测试结果

本文介绍Redis企业版（持久内存型）在不同测试场景下的测试结果。

工作负载

测试的总数据量8 GB，数据分布方法为zipfian，具体测试场景如下：

- Load：100%的写操作。
- Workload C：100%的读操作。
- Workload A：50%的更新操作与50%的读操作。

关于Workload的详细介绍，请参见Core Workloads。

> **说明** 由于应用场景不同，Set、Zset中的复杂命令无法进行通用测试，您可以根据您的业务场景自行测试。

测试指标

测试指标	说明
QPS	每秒处理的读写操作数，单位为次/秒。
INSERT Average Latency	写操作平均延迟，单位为微秒（us）。
INSERT 99th Percentile Latency	处理速度最快的99%写操作中，最长的延迟时间，单位为微秒。例如该指标的值为500微秒，表示99%的请求可以在500微秒内得到处理。
READ AverageLatency	读操作平均延迟，单位为微秒。
READ 99thPercentileLatency	处理速度最快的99%读操作中，最长的延迟时间，单位为微秒。
UPDATE AverageLatency	更新操作平均延迟，单位为微秒。
UPDATE 99thPercentileLatency	处理速度最快的99%更新操作中，最长延迟时间，单位为微秒。

Load场景测试结果

String数据结构

Value长度（字节）	QPS（次/秒）	INSERT Average Latency（微秒）	INSERT 99th Percentile Latency（微秒）
128字节	134,478	473	687
256字节	126,139	504	828
1024字节	99,775	638	1,051
2048字节	77,130	826	1,157
4096字节	60,646	1,050	1,534

Hash数据结构

Key中的Field数量	Value长度（字节）	QPS（次/秒）	INSERT Average Latency（微秒）	INSERT 99th Percentile Latency（微秒）
10	128	47,353	1,348	1,885
	256	46,716	1,366	2,181
	1,024	27,759	2,297	2,873
	2,048	16,605	3,833	4,923

List数据结构

Key中的Element数量	Value长度（字节）	QPS（次/秒）	INSERT Average Latency（微秒）	INSERT 99th Percentile Latency（微秒）
10	128	64,950	979	1,310
	256	47,157	1,348	1,752
	1,024	26,719	2,386	3,457
	2,048	16,714	3,811	4,751
	4,096	10,129	6,279	7,891

Set数据结构

Key中的Member数量	Value长度（字节）	QPS（次/秒）	INSERT Average Latency（微秒）	INSERT 99th Percentile Latency（微秒）
	128	63,670	1,001	1,514

Key中的Member数量	Value长度（字节）	QPS（次/秒）	INSERT Average Latency（微秒）	INSERT 99th Percentile Latency（微秒）
10	256	44,707	1,427	2,129
	1,024	25,375	2,513	3,239
	2,048	14,318	4,451	5,619
	4,096	8,378	7,608	9,095

Sorted Set数据结构

Key中的Element数量	Value长度（字节）	QPS（次/秒）	INSERT Average Latency（微秒）	INSERT 99th Percentile Latency（微秒）
10	128	40,292	1,585	2,469
	256	34,168	1,869	2,569
	1,024	21,347	2,989	3,905
	2,048	12,868	4,956	6,255
	4,096	7,864	8,101	9,599

Workload C场景测试结果

String数据结构

Value长度（字节）	QPS（次/秒）	READ AverageLatency（微秒）	READ 99thPercentileLatency（微秒）
128字节	170,699	362	546
256字节	163,829	380	565
1024字节	161,491	386	569
2048字节	130,189	487	729
4096字节	115,433	548	808

Hash数据结构

Key中的Field数量	Value长度（字节）	QPS（次/秒）	READ AverageLatency（微秒）	READ 99thPercentileLatency（微秒）
	128	96,111	662	874

Key中的Field数量	Value长度（字节）	QPS（次/秒）	READ AverageLatency（微秒）	READ 99thPercentileLatency（微秒）
10	256	86,892	733	915
	1,024	61,608	1,030	1,293
	2,048	37,334	1,696	2,331
	4,096	25,943	2,429	3,319

List数据结构

Key中的Element数量	Value长度（字节）	QPS（次/秒）	READ AverageLatency（微秒）	READ 99thPercentileLatency（微秒）
10	128	105,296	604	889
	256	97,047	655	890
	1,024	66,384	955	1,192
	2,048	35,796	1,769	2,461
	4,096	26,314	2,392	3,271

Set数据结构

Key中的Member数量	Value长度（字节）	QPS（次/秒）	READ AverageLatency（微秒）	READ 99thPercentileLatency（微秒）
10	128	97,825	651	896
	256	80,954	787	970
	1024	59,924	1060	1313
	2048	33,356	1900	2637
	4096	23,605	2677	3723

Sorted Set数据结构

Key中的Element数量	Value长度（字节）	QPS（次/秒）	READ AverageLatency（微秒）	READ 99thPercentileLatency（微秒）
	128	58,380	1,093	1,341
	256	56,287	1,133	1,390

Key中的Element数量	Value长度（字节）	QPS（次/秒）	READ AverageLatency（微秒）	READ 99thPercentileLatency（微秒）
	1,024	47,468	1,338	1,688
	2,048	30,073	2,096	2,783
	4,096	21,850	2,880	3,765

Workload A场景测试结果

String数据结构

Value长度（字节）	QPS（次/秒）	READ AverageLatency（微秒）	READ 99thPercentile Latency（微秒）	UPDATE AverageLatency（微秒）	UPDATE 99thPercentile Latency（微秒）
128	141,120	451	616	450	618
256	137,551	463	617	461	618
1,024	124,165	516	724	508	725
2,048	92,652	695	881	678	871
4,096	78,994	819	1,042	791	1,024

Hash数据结构

Key中的Field数量	Value长度（字节）	QPS（次/秒）	READ AverageLatency（微秒）	READ 99thPercentileLatency（微秒）	UPDATE AverageLatency（微秒）	UPDATE 99thPercentileLatency（微秒）
10	128	99,495	646	831	633	820
	256	88,235	731	985	712	966
	1,024	72,013	892	1,159	863	2,049
	2,048	45,790	1,379	1,898	1,354	2,821
	4,096	32,912	1,891	2,931	1,915	7,887

List数据结构

Key中的Element数量	Value长度（字节）	QPS（次/秒）	READ AverageLatency（微秒）	READ 99thPercentileLatency（微秒）	UPDATE AverageLatency（微秒）	UPDATE 99thPercentileLatency（微秒）
10	128	71,696	591	775	1,185	1,383
	256	66,294	638	800	1,281	1,456
	1,024	53,402	791	1,006	1,581	1,865
	2,048	36,519	1,221	1,581	2,232	2,831
	4,096	28,390	1,618	2,113	2,803	3,777

Set数据结构

Key中的Member数量	Value长度（字节）	QPS（次/秒）	READ AverageLatency（微秒）	READ 99thPercentileLatency（微秒）	UPDATE AverageLatency（微秒）	UPDATE 99thPercentileLatency（微秒）
10	128	66,346	640	792	1,282	1,445
	256	60,010	707	993	1,415	1,957
	1,024	45,359	933	1,128	1,858	2,073
	2,048	29,027	1,529	2,021	2,820	4,507
	4,096	21,440	2,144	2,773	3,726	5,095

Sorted Set数据结构

Key中的Element数量	Value长度（字节）	QPS（次/秒）	READ AverageLatency（微秒）	READ 99thPercentileLatency（微秒）	UPDATE AverageLatency（微秒）	UPDATE 99thPercentileLatency（微秒）
10	128	49,695	861	1,050	1,707	1,912
	256	48,036	891	1,084	1,763	1,970
	1,024	39,795	1,081	1,386	2,107	2,563
	2,048	28,415	1,597	1,981	2,855	3,589
	4,096	21,317	2,247	2,821	3,665	4,787

4.企业版（容量存储型）

4.1. 测试环境

本文介绍Redis企业版（容量存储型）性能测试所使用的具体环境。

产品概述

Redis企业版（容量存储型）基于云盘ESSD研发，兼容Redis核心数据结构与接口，可提供大容量、低成本、强持久化的数据库服务。容量存储型在降低成本和提升数据可靠性的同时，也解决了原生Redis因fork而预留部分内存的问题。适用于兼容Redis、需要大容量且较高访问性能的温冷数据存储场景。更多详情，请参见容量存储型。

测试环境

测试环境信息	说明
地域和可用区域	所有测试均在华北1（杭州）地域的可用区I中完成。
Redis实例架构	标准版（双副本）架构，详情请参见标准版-双副本。
部署压测工具的机器	云服务器ECS实例，规格为ecs.g6e.13xlarge，规格详情请参见实例规格族。
容量存储型实例规格	• tair.essd.standard.xlarge • tair.essd.standard.2xlarge • tair.essd.standard.4xlarge • tair.essd.standard.8xlarge • tair.essd.standard.13xlarge 规格详情请参见容量存储型。

4.2. 测试工具

Redis企业版（容量存储型）使用YCSB压测工具进行性能测试，本文为您介绍测试工具及其使用方法。

采用开源社区的YCSB压测工具进行压测。YCSB是一款Java编写的支持多种数据库的性能测试工具，具体安装和使用方法请参见YCSB。

本测试中，对YCSB的相关内容做了一定的修改，使其支持导入long类型的recordcount参数、支持测试Redis的String相关命令，修改后的完整源代码请参见YCSB源码。

4.3. 测试方法

本文介绍Redis企业版（容量存储型）性能测试的具体方法。

工作负载

- Load：100%的string set操作（写操作）。
- Uniform-Read：采用Workload A，100%均匀随机string get操作（读操作），主要测试严苛条件下的读性能。

- Zipfian-Read：采用Workload C，数据分布方法为zipfian，测试大部分读请求访问小部分数据的性能，符合大部分的读场景。
- Uniform-50%Read-50%Update：采用Workload A，50%的string set操作（更新操作）与50%的string get操作，主要测试随机更新的性能。

关于Workload的详细介绍，请参见Core Workloads。

测试场景

测试主要针对下述两种场景进行：

- 内存大于数据场景：绝大部分数据可以在内存中访问到，此场景下内存与数据的比例约为7:1。
- 数据大于内存场景：只有部分数据缓存在内存，绝大多数访问需要读写硬盘，此场景下内存与数据的比例约为1:4。

测试命令

下述脚本以数据大于内存场景为例。

```
#! /bin/bash
ip=192.168.0.23
port=3100
timeout=30000
command_group=string
recordcount=640000000
run_operationcount=20000000
fieldcount=1
fieldlength=100
threads=32
load_sleep_time=600
run_sleep_time=60
echo "#################################### $command_group  ################################
##############"
#Load
./bin/ycsb load redis -s -P workloads/workloada -p "redis.host=${ip}" -p  "redis.port=${por
t}" -p "recordcount=${recordcount}" -p "operationcount=${recordcount}" -p "redis.timeout=${
timeout}" -p "redis.command_group=${command_group}" -p "fieldcount=${fieldcount}" -p "field
length=${fieldlength}" -threads ${threads}
sleep ${load_sleep_time}
#Uniform-Read
./bin/ycsb run redis -s -P workloads/workloadc  -p "redis.host=${ip}" -p  "redis.port=${por
t}" -p "recordcount=${recordcount}" -p "operationcount=${run_operationcount}" -p "redis.tim
eout=${timeout}" -p "redis.command_group=${command_group}" -p "fieldcount=${fieldcount}" -p
"fieldlength=${fieldlength}" -p "requestdistribution=uniform" -threads ${threads}
sleep ${run_sleep_time}
#Zipfian-Read
./bin/ycsb run redis -s -P workloads/workloadc  -p "redis.host=${ip}" -p  "redis.port=${por
t}" -p "recordcount=${recordcount}" -p "operationcount=${run_operationcount}" -p "redis.tim
eout=${timeout}" -p "redis.command_group=${command_group}" -p "fieldcount=${fieldcount}" -p
"fieldlength=${fieldlength}" -p "requestdistribution=zipfian" -threads ${threads}
sleep ${run_sleep_time}
#Uniform-50%Read-50%Update
./bin/ycsb run redis -s -P workloads/workloada  -p "redis.host=${ip}" -p  "redis.port=${por
t}" -p "recordcount=${recordcount}" -p "operationcount=${run_operationcount}" -p "redis.tim
eout=${timeout}" -p "redis.command_group=${command_group}" -p "fieldcount=${fieldcount}" -p
"fieldlength=${fieldlength}" -p "requestdistribution=uniform" -threads ${threads}
```

参数说明

参数	说明
ip	Redis实例的IP地址。
port	Redis实例的服务端口。
timeout	测试命令的超时时间。
command_group	测试类型，配置为String。
recordcount	数据装载阶段准备的数据量。

参数	说明
run_operationcount	Run阶段操作的数据量。本测试中： • 内存大于数据场景下，配置和recordcount参数相同的值。 • 数据大于内存场景下，配置的值为recordcount参数值除以32。
fieldcount	字段个数，配置为1。
fieldlength	值长度，配置为100。
threads	YCSB线程数，根据实例规格配置。

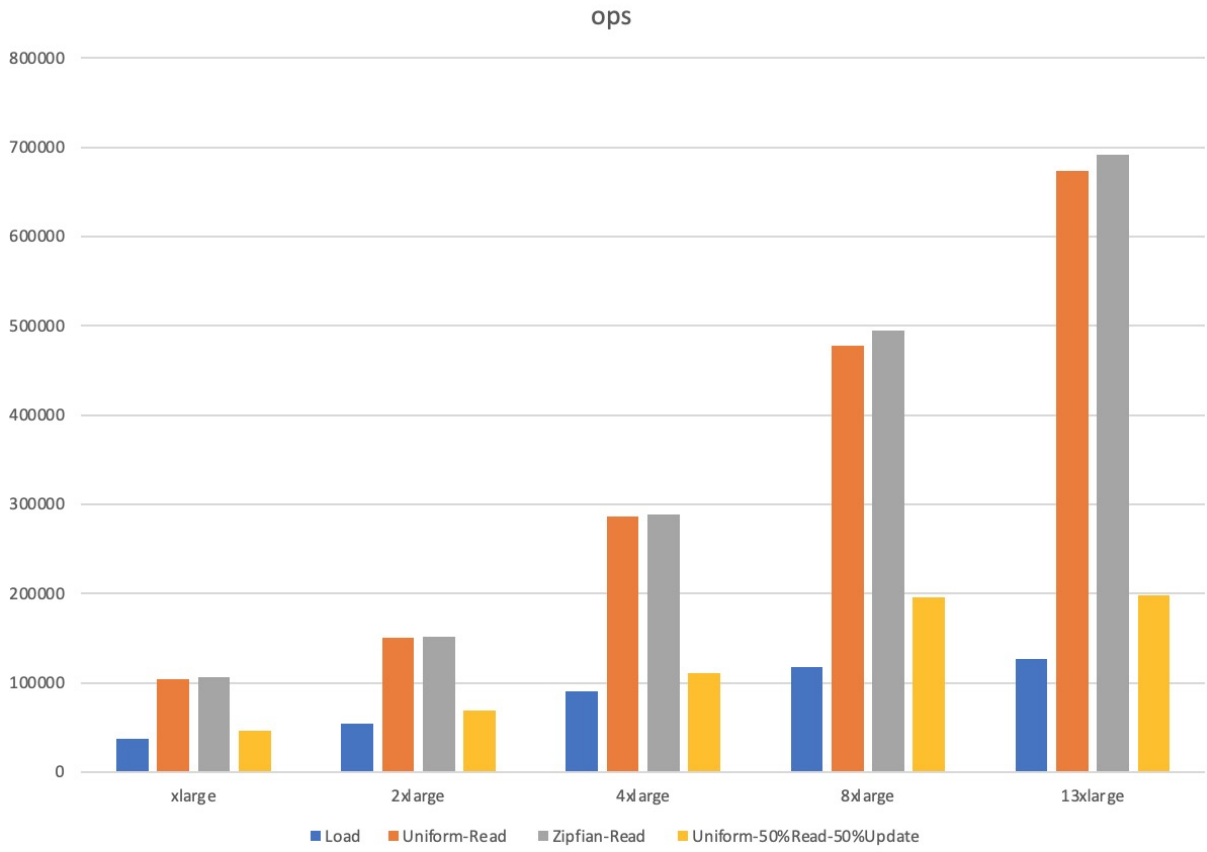
4.4. 测试结果

本文介绍Redis企业版（容量存储型）在不同测试场景下的测试结果。

测试指标

测试指标	说明
QPS	每秒处理的读写操作数，单位为次/秒。
Average Latency	读或写操作的平均延迟，单位为微秒（us）。
99th Percentile Latency	处理速度最快的99%的操作中，最长的延迟时间，单位为微秒。例如该指标的值为500微秒，表示99%的请求可以在500微秒内得到处理。

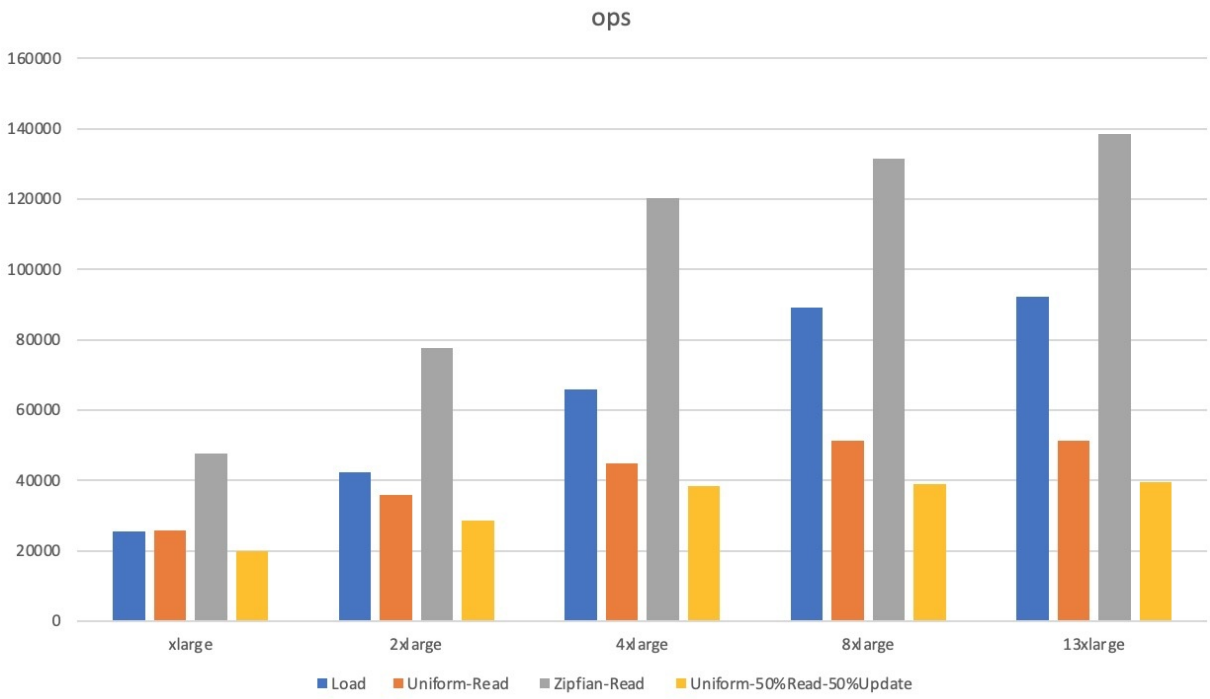
内存大于数据场景下测试结果



实例规格	YCSB配置	工作负载	QPS（次/秒）	Average Latency（微秒）	99th Percentile Latency（微秒）
tair.essd.standard.xlarge	recordcount=20000000 run_operationcount=20000000 threads=32	Load	36740	851	1595
		Uniform-Read	103890	294	907
		Zipfian-Read	106357	288	865
		Uniform-50%Read-50%Update	46610	Read：530	Read：1108
				Update：795	Update：1684
tair.essd.standard.2xlarge	recordcount=40000000 run_operationcount=40000000	Load	54670	911	1528
		Uniform-Read	150796	314	995
		Zipfian-Read	151110	314	977
				Read：537	Read：948

实例规格	threads=50 YCSB配置	Uniform-50%Read-50%Update 工作负载	69137 QPS（次/秒）	Average Latency（微秒）	99th Percentile Latency（微秒）
				Update：878	Update：1479
air.essd.standard.4xlarge	recordcount=80000000 run_operationcount=80000000 threads=100	Load	90703	1099	1697
		Uniform-Read	285833	339	1196
		Zipfian-Read	288750	335	1162
		Uniform-50%Read-50%Update	110316	Read：757	Read：1114
				Update：1041	Update：1536
tair.essd.standard.8xlarge	recordcount=160000000 run_operationcount=160000000 threads=120	Load	117581	1011	1692
		Uniform-Read	477099	242	784
		Zipfian-Read	494550	234	727
		Uniform-50%Read-50%Update	196245	Read：519	Read：829
				Update：691	Update：1096
tair.essd.standard.13xlarge	recordcount=240000000 run_operationcount=240000000 threads=160	Load	126366	1249	2281
		Uniform-Read	673183	231	637
		Zipfian-Read	691383	230	652
		Uniform-50%Read-50%Update	197803	Read：678	Read：940
				Update：935	Update：1925

数据大于内存场景



实例规格	YCSB配置	工作负载	QPS（次/秒）	Average Latency（微秒）	99th Percentile Latency（微秒）
tair.essd.standard.xlarge	recordcount=640000000 run_operationcount=20000000 threads=32	Load	25561	1245	3497
		Uniform-Read	25727	1239	2042
		Zipfian-Read	47559	667	1217
		Uniform-50%Read-50%Update	19731	Read：1576	Read：6383
				Update：1639	Update：6487
tair.essd.standard.2xlarge	recordcount=1280000000 run_operationcount=40000000 threads=50	Load	42287	1179	3465
		Uniform-Read	35794	1394	1880
		Zipfian-Read	77759	637	1219
		Uniform-50%Read-50%Update	28656	Read：1716	Read：8863
				Update：1761	Update：8951

实例规格	YCSB配置	工作负载	QPS（次/秒）	Average Latency（微秒）	99th Percentile Latency（微秒）
air.essd.standard.4xlarge	recordcount=2560000000 run_operationcount=80000000 threads=100	Load	65923	1514	6615
		Uniform-Read	44753	2232	7903
		Zipfian-Read	120337	826	1382
		Uniform-50%Read-50%Update	38470	Read：2577 Update：2617	Read：8535 Update：8583
tair.essd.standard.8xlarge	recordcount=5120000000 run_operationcount=160000000 threads=120	Load	89231	1340	9575
		Uniform-Read	51175	2343	2955
		Zipfian-Read	131317	911	1573
		Uniform-50%Read-50%Update	38930	Read：3063 Update：3097	Read：8695 Update：8735
tair.essd.standard.13xlarge	recordcount=7680000000 run_operationcount=240000000 threads=160	Load	92163	1733	9879
		Uniform-Read	51267	3510	16623
		Zipfian-Read	138522	1152	2131
		Uniform-50%Read-50%Update	39584	Read：4022 Update：4057	Read：12159 Update：12239

5.Tair扩展数据结构

5.1. TairRoaring性能白皮书

TairRoaring是基于Tair引擎的RoaringBitmap实现，提供高效的计算模块和极高的稳定性，支持使用少量的存储空间来实现海量数据的查询优化。

TairRoaring在RoaringBitmap的基础上完成大量优化：

- 通过2层索引和多种动态容器（Container），平衡了多种场景下性能和空间效率。
- 使用了包括SIMD instructions、Vectorization、PopCnt算法等多种工程优化，提升了计算效率，实现了高效的时空效率。
- 基于Tair提供的强大计算性能和极高的稳定性，为用户场景保驾护航。

> **说明** 关于TairRoaring的详情介绍，请参见TairRoaring。

测试工具

本文提供基于Go语言编写、类似redis-benchmark语法的测试工具，方便使用与修改，支持生成2个RAND随机值，便于构建更灵活的测试用例，同时支持将结果以直方图的形式输出，更多信息请参见TairRoaring测试工具。

用法：

```
Usage of ./redis:
  -a string          实例账号密码，格式为<user>:<password>。
  -batching int      管道（Pipeline）模式，并设置单次Pipeline执行的独立命令个数，该模式为MULTI-EXEC的组合命令，例如-batching 10表示一条MULTI和EXEC之间包含10条Command命令。
  -c int             运行的总测试数，该参数会屏蔽测试持续时间参数（-d）。
  -command string    测试命令。您可以将__RAND__、__RAND2__作为随机字段添加到命令或参数的后缀中，随机字段仅在后缀时生效并替换为随机值，默认命令为TR.GETBIT foo-__RAND__。
  -d int             测试持续时间，单位秒，默认30。
  -h string          实例的连接地址，默认为127.0.0.1。
  -p int             并发数，默认为4。
  -port int          实例的端口，默认为6379。
  -r int             设置__RAND__随机值的范围，默认为100000000。
  -r2 int            设置__RAND2__随机值的范围，默认为100000000。
```

示例：

```
# 通过TR.SETBIT命令进行测试（Key为10万以内的随机数，Field为1000万以内的随机数）。
./redis -h r-**********0d7f.redis.zhangbei.rds.aliyuncs.com -a user:password -d 30 -r 100000 -r2 10000000 -command "TR.SETBIT foo-__RAND__ bar-__RAND2__" -p 20 -c 1
```

标准版测试方法与测试结果

测试实例规格为Tair（Redis企业版）性能增强型16 GB，标准版，网络时延低于0.1ms。

单Key测试

- 性能测试

对同一个Key并发写入，通过调整并发取得最大QPS值。

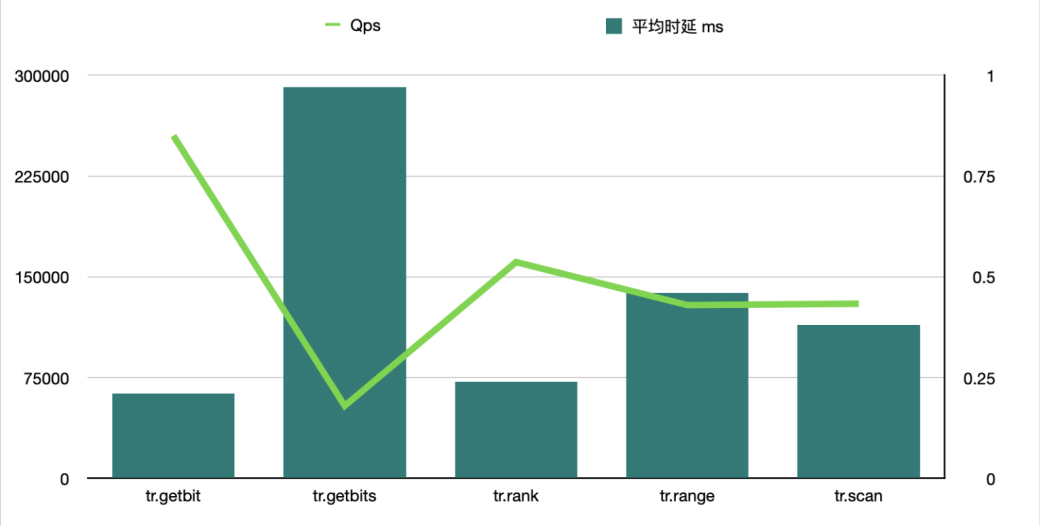
- 读命令测试

  测试的TairRoaring命令：TR.GETBIT、TR.GETBITS、TR.RANGE、TR.SCAN、TR.RANK，测试命令示例：

```
./redis -h r-**********0d7f.redis.zhangbei.rds.aliyuncs.com -a user:password -d 30 -r 100000 -r2 10000000 -command "TR.GETBIT foo-__RAND__ bar-__RAND2__" -p 20 -c 1
```

  测试结果：

压测命令	Key数量	读取数量	参数范围	QPS参考值	平均时延（ms）
TR.GETBIT	1	1	1~10000000	255000	0.21
TR.GETBITS	1	100	1~10000000	54000	0.97
TR.RANK	1	1	1~10000000	161000	0.24
TR.RANGE	1	100	0~100	129000	0.46
TR.SCAN	1	100	1~10000000	130000	0.38



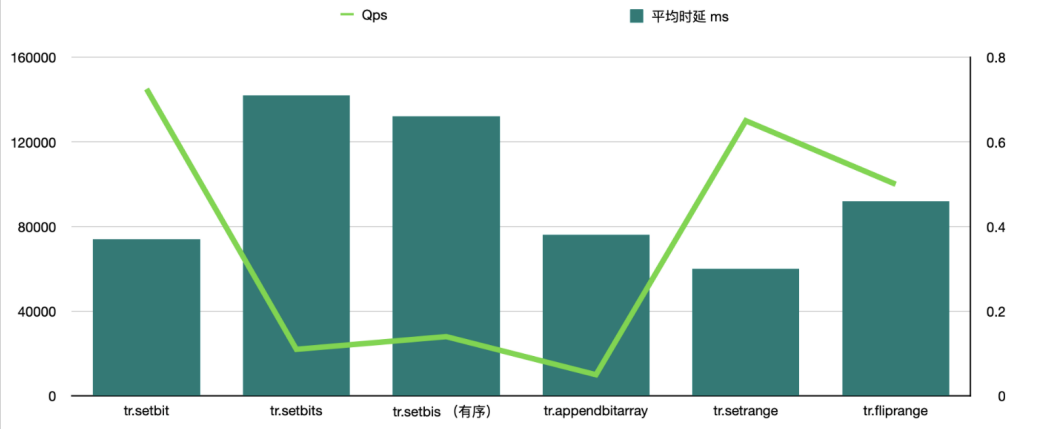
- 写命令测试

  测试的TairRoaring命令：TR.SETBIT、TR.SETBITS、TR.APPENDBITARRAY、TR.SETRANGE、TR.FLIPRANGE，测试命令示例：

```
./redis -h r-**********0d7f.redis.zhangbei.rds.aliyuncs.com -a user:password -d 30 -r 100000 -r2 10000000 -command "TR.SETBIT foo-__RAND__ bar-__RAND2__" -p 20 -c 1
```

  测试结果：

压测命令	Key数量	读取数量	参数范围	QPS参考值	平均时延（ms）
TR.SETBIT	1	1	1~10000000	145000	0.37
TR.SETBITS	1	100	1~10000000	22000	0.71
TR.SETBITS（有序）	1	100（bit的最大偏移量为2的32次方）	1~6000	28000	0.66
TR.APPENDBITARRAY	1	1000（其中500个bit为1）	1~10000000	10000	0.38
TR.SETRANGE	1	-	1000	130000	0.3
TR.FLIPRANGE	1	-	1~10000000	100000	0.46



- batch测试

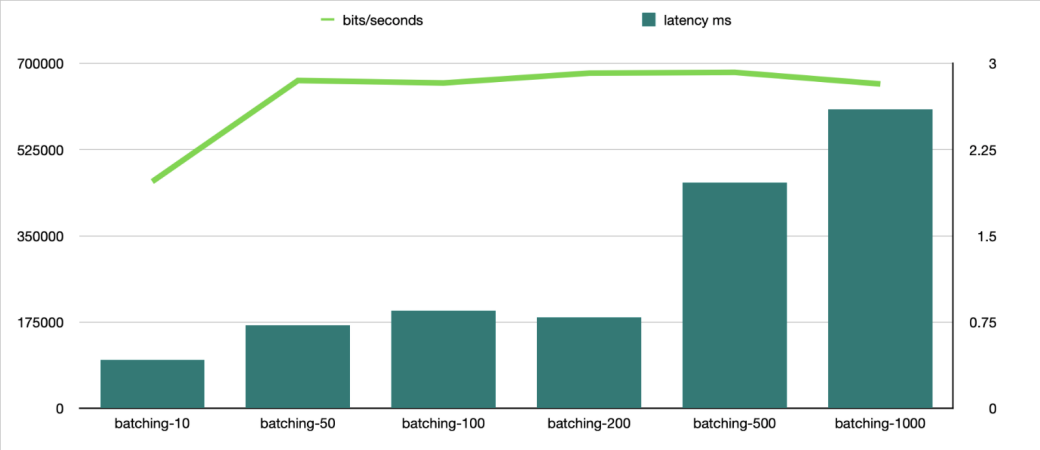
  测试不同命令在Pipeline模式下不同batching大小的性能情况。

- TR.SETBIT

  测试命令示例：

```
./redis -h r-**********0d7f.redis.zhangbei.rds.aliyuncs.com -a user:password -d 30 -r 10000000 -command "TR.SETBIT foo-__RAND__ 1" -batching 10 -p 20 -c 1
```

parallel	batching	bps（bit per second）	平均时延（ms）
20	10	460000	0.42
10	50	665000	0.72
6	100	660000	0.85
3	200	680000	0.79
3	500	681500	1.96
2	100	658000	2.6



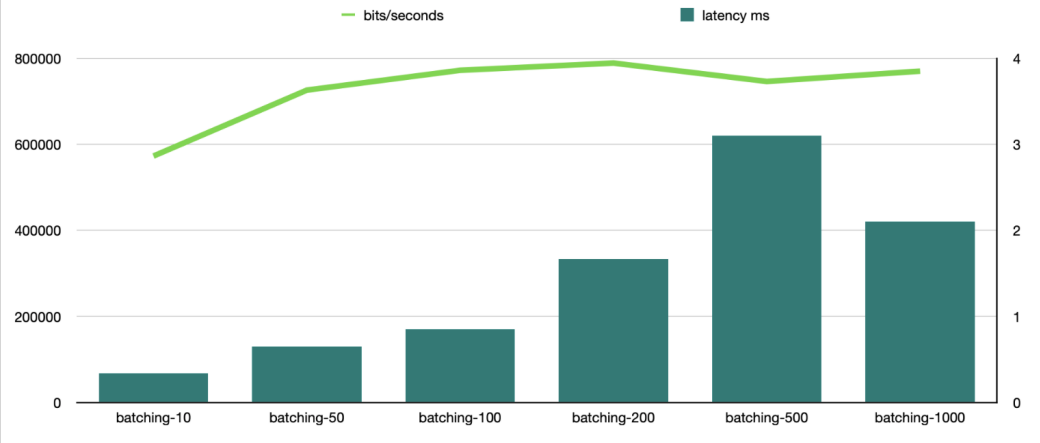
- TR.GETBIT

  测试命令示例：

```
./redis -h r-**********0d7f.redis.zhangbei.rds.aliyuncs.com -a user:password -d 30 -r 100000 -r2 10000000 -command "TR.GETBIT foo-__RAND__ __RAND2__" -batching 10 -p 20 -c 1
```

> **说明** 若foo-__RAND__指定的Key不存在，则会按照bit为0返回，因此需要预先写入数据才能较好地测试不同负载下的表现。

parallel	batching	bps（bit per second）	平均时延（ms）
20	10	572700	0.34
10	50	725900	0.65
7	100	772000	0.85
7	200	788800	1.67
5	500	746000	3.1
2	100	770000	2.1



- 单Key内存测试

  通过TR.STAT命令分析不同容量场景下RoaringBitmap的bit分布情况，测试命令示例：

```
TR.STAT foo JSON
```

- 稀疏场景

  稀疏场景指bit分布较为分散，bit分布密度低于6.25%。该场景下RoaringBitmap基本以array容器为主，容量约为cardinality * 2 Byte。

cardinality	rle-c	array-c	bitset-c	heap mem (byte)
37700484	-	65536	-	75400968
75011384	-	65536	-	150022768
100403264	-	65536	-	200806528
163090592	-	65536	-	326181184

- 常规随机分布场景

  该场景下RoaringBitmap基本以bitset容器为主，当array容器内元素超过4096时，会陆续转换成bitset容量，符合推算逻辑。

cardinality	rle-c	array-c	bitset-c	heap mem (byte)
253104088	-	65534	2	506208102
261169659	-	63273	2263	522227634
267974804	-	35932	29604	533159296
273694253	-	6607	58929	536491922
343504134	-	0	65536	536870912
535589835	-	0	65536	536870912

- 连续随机bit场景

  连续随机bit场景的存储容量与bit分布逻辑强相关，该场景下测试结果参考意义不大。

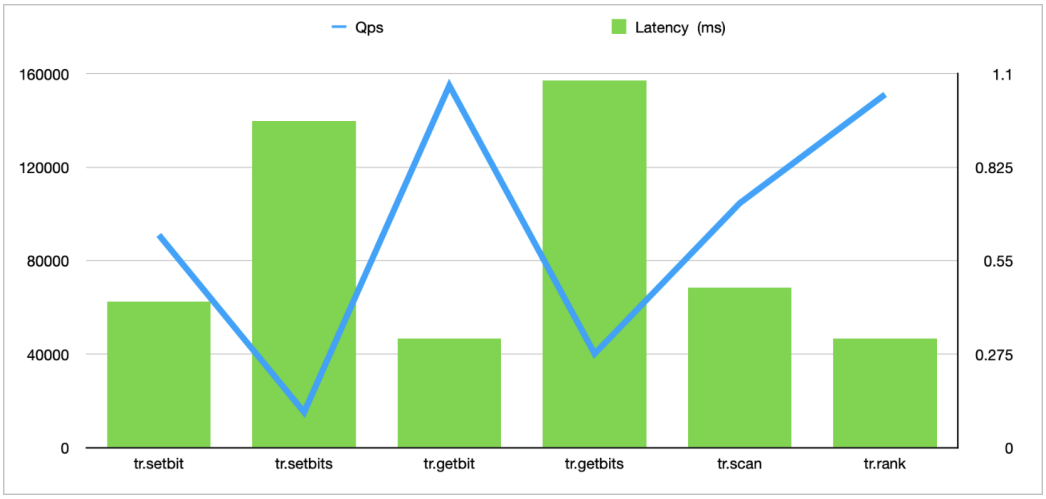
多Key模型测试

- 多Key写入测试

  测试多个Key并发写入时引擎的总体性能，测试命令示例（TR.SETBITS每次创建100个随机bit）：

```
./redis -h r-**********0d7f.redis.zhangbei.rds.aliyuncs.com -a user:password -d 30 -r 100
00000 -r2 100000 -command "TR.SETBITS foo-__RAND2__ __RAND__ __RAND__ __RAND__ __RAND__ _
_RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__
__RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__
__RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__
__RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__
__RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__
__RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__
__RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__
__RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__
__RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__
__RAND__ __RAND__ __RAND__ __RAND__ __RAND__ __RAND__" -p 20
```

压测命令	FIELDS/RANGE	QPS参考值	平均时延（ms）
TR.SETBIT	1	91003	0.43
TR.SETBITS	100	15127	0.96
TR.GETBIT	1	154941	0.32
TR.GETBITS	100	40166	1.08
TR.SCAN	100	104637	0.47
TR.RANK	-	151161	0.32



- 集合运算

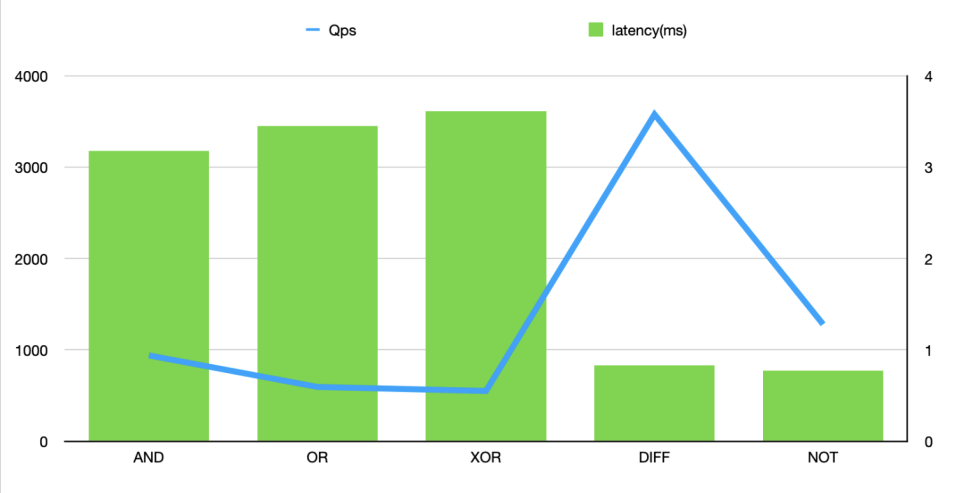
- TR.BITOP

  对多个RoaringBitmap执行集合运算逻辑操作，并将运算结果存储至新的Key中，支持AND、OR、NOT、DIFF、XOR运算逻辑。

  测试命令示例：

```
./redis -h r-**********0d7f.redis.zhangbei.rds.aliyuncs.com -a user:password -d 30 -r 100000 -r2 10000000 -command "TR.BITOP dest-__RAND__ AND foo-__RAND__ foo-__RAND__" -p 3 -c 1
```

sub命令	parallel	QPS参考值	平均时延（ms）
AND	3	940	3.18
OR	2	595	3.45
XOR	2	551	3.61
DIFF	3	3577	0.83
NOT	1	1281	0.77



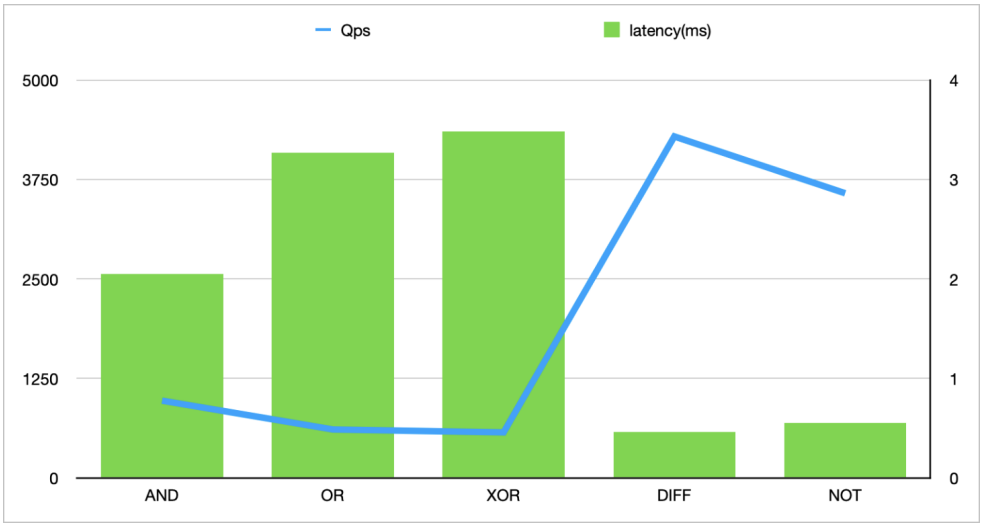
- TR.BITOPCARD

  对多个RoaringBitmap执行集合运算逻辑操作，仅返回运算结果中bit值为1的数量，支持AND、OR、NOT、DIFF、XOR运算逻辑。

  测试命令示例：

```
./redis -h r-**********0d7f.redis.zhangbei.rds.aliyuncs.com -a user:password -d 30 -r 100000 -r2 10000000 -command "TR.BITOPCARD AND foo-__RAND__ foo-__RAND__" -p 2 -c 1
```

sub命令	parallel	QPS参考值	平均时延（ms）
AND	2	971	2.05
OR	2	609	3.27
XOR	2	572	3.48
DIFF	2	4290	0.46
NOT	2	3577	0.55



集群版测试方法与测试结果

测试4个不同分片数的性能增强集群版实例，每个分片均为4 GB，规格如下：

- 8 GB集群性能增强版（2分片）
- 16 GB集群性能增强版（4分片）
- 32 GB集群性能增强版（8分片）
- 64 GB集群性能增强版（16分片）

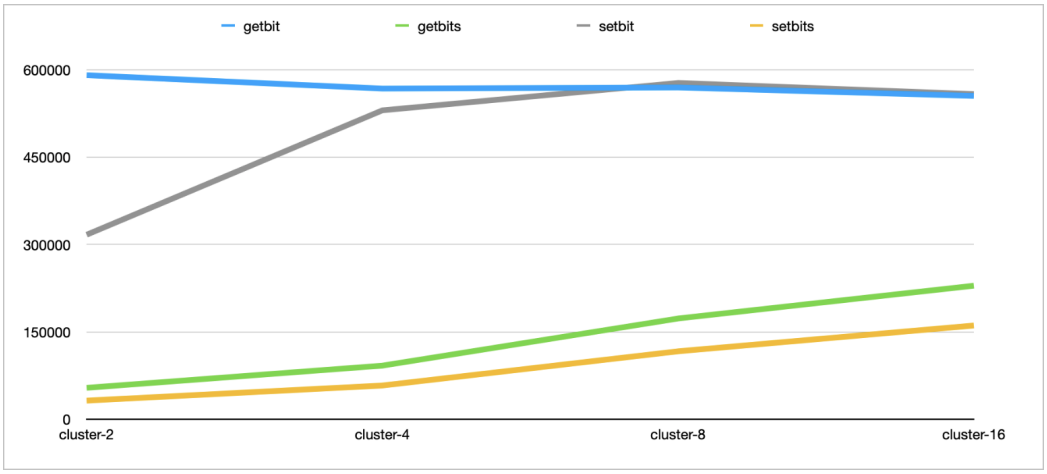
为避免测试场景存在网络带宽干扰，将所有集群的带宽均升级至最大（2048 MB/s）。压力测试客户端部署在Tair集群实例相同的可用区，并通过Proxy（代理模式）访问Tair集群实例。根据不同的测试命令分别测试各集群实例执行单Key命令的平均QPS值。

测试的TairRoaring命令：TR.GITBIT、TR.GITBITS、TR.SETBIT、TR.SETBITS，测试命令示例：

```
./redis -h r-**********0d7f.redis.zhangbei.rds.aliyuncs.com -a user:password -d 30 -r 10000
0 -r2 10000000 -command "TR.SETBIT foo-__RAND__ bar-__RAND2__" -p 20 -c 1
```

测试结果：

压测命令	2分片集群（QPS值）	4分片集群（QPS值）	8分片集群（QPS值）	16分片集群（QPS值）
TR.GITBIT	590742	567738	569610	555178
TR.GITBITS	53900	91991	172969	229214
TR.SETBIT	316753	530367	577406	558301
TR.SETBITS	31917	57843	116614	160891



5.2. TairSearch性能白皮书

本文介绍TairSearch、ElaticSearch数据写入和搜索性能的测试方法及测试结果。

TairSearch是基于Redis module全自研（不基于Lucene等开源搜索库）的全文搜索数据结构，采用和Elasticsearch相似（ES-LIKE）的查询语法，可实现高效的全文搜索，更多信息请参见TairSearch。

测试说明

测试环境

执行测试的服务器规格：

- 架构：x86_x64
- 内存：251 GB
- CPU：32 Core，64 Threads
- 操作系统版本：Linux 4.19.91-011.ali4000.alios7.x86_64

数据库内核：

TairSearch	ElaticSearch
• 版本：性能增强型1.7.28 • 配置：单进程无副本，4个Search线程 • 内存：10 GB • CPU：分配5个CPU（1 worker + 4 io/search）	• 版本：8.1.1 • 配置：单节点，单shard无副本，indices.queries.cache.size和index.queries.cache.enabled都是默认值。 • 内存：-Xms10g、-Xmx10g • CPU：node.processors设置为5

测试数据

本次测试数据为维基百科公开的英文文章摘要。

压测工具

下载TairSearch测试工具，解压后，选择对应操作系统（Linux、Windows、Darwin）的二进制可执行文件。

以Linux为例，执行 `./TairSearchBench.linux --help` 可查看使用方法，用法如下：

```
Usage of ./TairSearchBench.linux:
  -a string
        The address of network to connect
        # 实例连接地址。
  -c int
        Benchmark concurrency (default 30)
        # 并发数，默认为30。
  -e string
        The engine backend to run [tairsearch/elasticsearch]
        # 指定运行引擎为TairSearch或ElaticSearch。
  -f string
        Input file to ingest data from (wikipedia abstracts)
        # 执行数据文件路径。
  -h string
        Print usage (default "help")
        # 显示使用方法。
  -n uint
        Specify the number of times to benchmark (default 10000)
        # 指定总测试数，默认为10000。
  -q string
        Search query string to benchmark
        # 指定压测的查询语句。
  -t string
        Specify the type of benchmark [write/search]
        # 指定测试类型为write或search。
```

测试步骤与测试用例

1. 创建Schema（索引）。

   由于TairSearch和ElasticSearch DSL部分兼容，可共用Schema。

```
schemaByte, _ := json.Marshal(JsonObject{
    "settings": JsonObject{
    "number_of_shards":            1,
    "number_of_replicas":          1,
    "refresh_interval":            "1s",
    "index.queries.cache.enabled": false,
    },
    "mappings": JsonObject{
    "properties": JsonObject{
    "id":       JsonObject{"type": "keyword", "index": false},
    "url":      JsonObject{"type": "text", "index": false},
    "title":    JsonObject{"type": "text", "index": false},
    "abstract": JsonObject{"type": "text", "analyzer": "standard"},
    },
  },
})
```

> **说明** 当前TairSearch创建Schema时无需配置 `settings` ，会自动忽略。

2. 数据写入（单Schema）。

   在一个索引中写入500万个文档，文档内存为1.5 GB，压测并发数为30，命令示例：

   - TairSearch: `TairSearchBench -t write -e tairsearch -f ./enwiki-latest-abstract.xml -c 30 -n 5000000 -a 192.168.0.1:6379`
   - ElasticSearch（Refresh频率为1s）：`TairSearchBench -t write -e elasticsearch -f ./enwiki-latest-abstract.xml -c 30 -n 5000000 -a http://192.168.0.1:9200`

3. 关键字查询。

   搜索关键字，查询次数为10万，压测并发数为30，命令示例：

   - TairSearch: `TairSearchBench -t search -e tairsearch -c 30 -n 100000 -q '{"query":{"term":{"abstract":"hello"}}}' -a 192.168.0.1:6379`
   - ElasticSearch: `TairSearchBench -t search -e elasticsearch -c 30 -n 100000 -q '{"track_total_hits": true, "query":{ "term":{"abstract":"world"}}}' -a http://192.168.0.1:9200`

4. 聚合查询。

   在 `"abstract"="hello"` 的文档中，统计ID字段的value的去重个数，类似SQL: `SELECT distinct_count(ID) FROM table WHERE abstract="hello" ORDER BY ID ASC LIMIT 10` 。

   查询次数为10万，压测并发数为30，命令示例：

   - TairSearch: `TairSearchBench -t search -e tairsearch -c 30 -n 100000 -q '{"size":0, "query":{"term":{"abstract":"hello"}}, "aggs":{"a1":{"terms":{"field":"id", "order":{"_key":"asc"} }} } }' -a 192.168.0.1:6379`
   - ElasticSearch: `TairSearchBench -t search -e elasticsearch -c 30 -n 100000 -q '{"size":0, "query":{"term":{"abstract":"hello"}}, "aggs":{"a1":{"terms":{"field":"id", "order":{"_key":"asc"} }} } }' -a http://192.168.0.1:9200`

> **说明** 同时在开启缓存（默认）和关闭缓存情况下，分别测试TairSearch、ElasticSearch的性能。

测试结果

- 数据写入

类别	TairSearch	ElasticSearch（Refresh频率为1s）
使用内存（GB）	3.89	5 GB内存与1 GB磁盘
总耗时（s）	257.33	1995.75
QPS值	19430.09	2505.34
平均时延（ms）	1.54	11.97

> **说明** 同时测试了ElasticSearch（Refresh为实时），由于写入时间较慢，仅测试写入个1000个文档数，压测并发数为30，测试结果：总耗时为5.16s、QPS值为199.54、平均时延为148.22ms。

- 关键字查询

类别	TairSearch	ElasticSearch
总耗时（s）	7.04	39.56
QPS值	14209.48	2528.23
平均时延（ms）	2.11	11.86

- 聚合查询

类别	TairSearch（默认）	TairSearch（关闭Query cache）	ElasticSearch（默认）	ElasticSearch（关闭缓存）
总耗时（s）	1.75	124.98	12.86	121.95
QPS值	57025.44	800.37	7780.01	82.24
平均时延（ms）	0.53	37.48	3.86	364.60

> **说明** ElasticSearch（关闭缓存）由于查询较慢，仅查询10000次，其余均为100000次。

总结

TairSearch经过精巧的倒排索引设计和算法实现，充分利用了内存运算速度快的优势，提升数据写入与查询性能；通过并发结构的设计实现无锁并发搜索，提升搜索吞吐量；同时在实现时尽量保证存储的局部性，提升缓存命中（Cache hit）概率。