

延云 YDB

安装与使用说明书

(2015-11-22 版本 V1.02)

大数据的未来在大索引, 大索引的未来在延云.

延云千亿大数据即席查询解决方案

微信: ycloudnet
QQ : 1820150327
电话: 024-83653716
邮箱: ycloudnet@163.com
主页: <http://www.ycloud.net.cn>

目录

延云 YDB	1
安装与使用说明书.....	1
延云简介.....	3
使用许可.....	3
大索引技术.....	4
YDB 特性.....	4
成员概览.....	5
YDB 检索过程示意图.....	5
YDB 对故障的处理	5
安装部署.....	6
依赖环境.....	6
配置文件.....	6
启动集群.....	7
启动 YDB 服务.....	8
停止 YDB 服务.....	8
停止集群.....	8
集群使用.....	8
创建表.....	8
数据导入.....	9
数据查询.....	10
过期分区的索引清理.....	10
日常清理.....	10
YDB SQL 使用说明	10
分区:	10
Worker:.....	11
Task:	11
Merger Server:	11
SQL 写法与例子.....	11
商业版 Sql 支持的额外 Sql 写法	错误！未定义书签。
经典配置参数.....	14
硬件配置.....	14
storm.yaml 配置	14
ydb_site.yaml 配置	15
内存较小情况下的配置.....	16
storm.yaml 配置	16
ydb_site.yaml 配置	16
其他注意事项.....	17
过载保护（熔断机制）	17
数据导入注意事项.....	17
数据延迟的处理.....	17

索引的管理方式与性能的关系.....	17
关于查询最多返回 前 10000 条数据的说明.....	18
如何让 YDB 性能提升 10 倍（千亿规模集群 DBA 必读）	18
系统启动阶段会碰到的问题.....	19
心跳的延迟.....	19
文件数量与磁盘 IO 负载的权衡-复合索引	20
异常关机或者结束进程导致的集群启动不了的解决办法	21
YDB 二期功能预览	21
企业版 YDB.....	22
企业版 7 天免费试用授权码生成.....	22

延云简介

延云计算技术有限公司：

国内第一家“千亿级实时多维分析”解决方案提供商。

旨在为企业提供千亿级大数据的高可靠、低成本、高性能的实时大数据解决方案。

我们的团队

延云核心开发团队成员大多出自阿里腾讯，均有超 10 年的从业经验，也有一些留英归来的硕士。

团队成员中也不乏一些开源项目的核心 committer，如 [JStorm](#) 与 [Mdrill](#)。

我们的成员曾在阿里腾讯设计出多个千亿规模的大数据系统，部分项目在业界也是大名鼎鼎。

11 月 19 日阿里巴巴 JStorm 正式成为 Apache Storm 的子项目

10 月 7 日腾讯 Hermes 集群单日数据规模突破 3600 亿，规模超过 600 台。

我们的核心产品 YDB:

YDB 是我们自主研发的一个大型分布式索引系统。旨在为数据总量为万亿级别、每天千亿级别数据增量的项目提供近似实时的数据导入，并提供近似实时响应的多维查询与统计服务。

使用许可

请您务必阅读软件主目录的 LICENSE.txt，只有您同意许可协议，才可以使用本软件。

大索引技术

为什么要使用大索引？使用后会有什么好处？

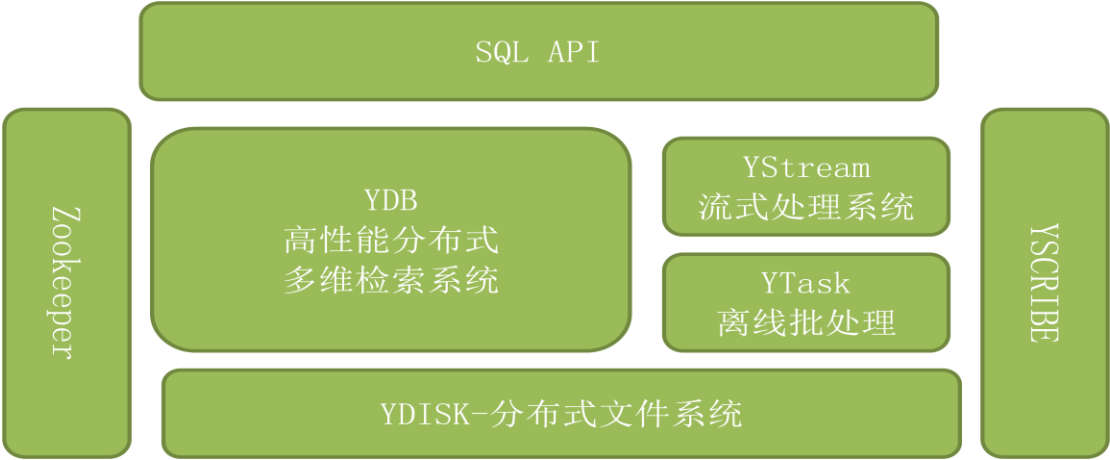
1. 索引大幅度的加快数据的检索速度。
2. 索引可以显著减少查询中分组、统计和排序的时间。
3. 索引大幅度的提高系统的性能和响应时间，从而节约资源。

正因为大索引技术可以显著的降低大数据的处理成本，显著提高大数据系统的执行效率，延云自主研发了自己的分布式大索引系统 YDB。

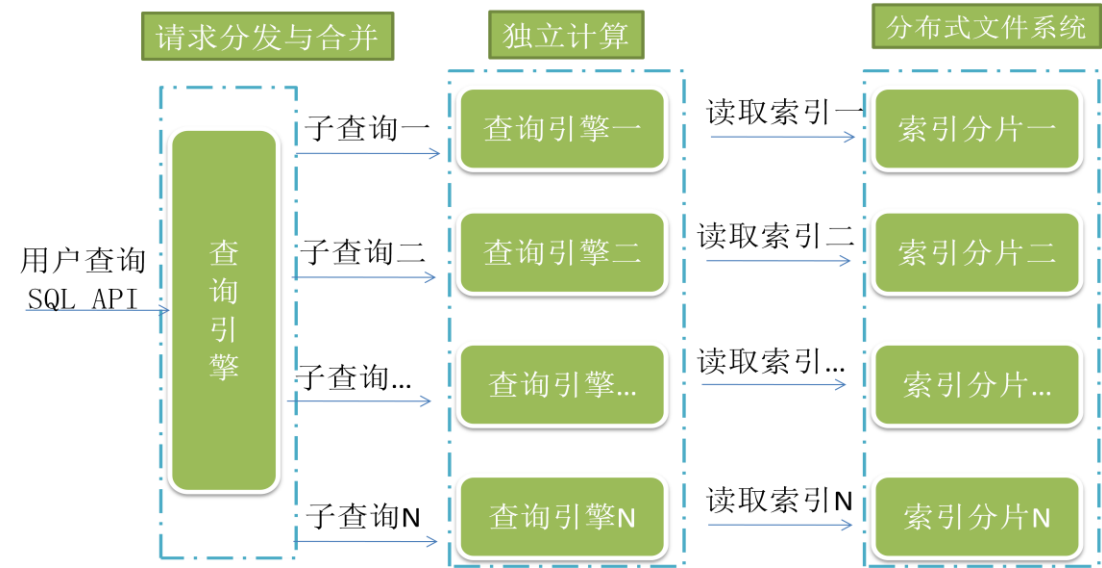
YDB 特性

1. 千亿规模
在真实业务环境上验证，每天可达千亿增量，总数据量可达几万亿。
2. 低延迟
数据从产生到能查询，根据配置的不同一般在十几秒到几分钟。
3. 查询快-高性能
常规查询毫秒级响应
常规统计秒级响应。
4. 实时搜索
长文本字段可以根据关键词进行全文检索模糊匹配，并且有较高的性能。
5. 多维钻取
支持上万个维度，任意组合查询，任意维度组合过滤、分组，统计、排序。
6. 容灾可靠
索引存储在分布式文件系统中，不因硬件的损坏或异常宕机而丢失数据。
7. **Sql Api:**
更易于上手与使用。

成员概览



YDB 检索过程示意图



YDB 对故障的处理

指标	描述
存储节点故障	数据存储于 YDISK 上，多份冗余，数据不受机器硬件故障影响
查询服务节点异常	服务异常可以感知的自动重启，硬件损坏，服务可以自动迁移到其他机器上继续服务。

安装部署

依赖环境

1. 机器时钟要校准。
2. java 1.7 及以上版本
3. python 环境（一般系统默认自带）
4. 安装 hadoop2.2.0 及以上版本
 仅需安装 hdfs。
 需独立 hdfs 集群，为保持其性能请勿与多业务共享 hdfs 集群。
5. zookeeper 3.4.5 及以上版本
6. 配置环境变量

```
echo "export LANG=zh_CN.UTF-8" >> ~/.bashrc
export LANG=zh_CN.UTF-8
cat ~/.bashrc
```
7. 调整操作系统允许同时打开的句柄数量，通过 ulimit 配置,操作系统默认的 1024 个句柄不够
8. 为每台机器配置 hostname，并配置/etc/hosts
 注意 hostname 需能正确的解析出 IP，且不要让几台机器出现重复的 hostname
9. 确保不会因为防火墙导致网络不通，关闭防火墙方法如下

```
iptables -F
```

配置文件

1. 集群配置
 配置文件分别是 conf 目录下的 defaults.yaml 与 storm.yaml 文件。
 其中 defaults.yaml 为默认配置文件，请勿修改，仅配置 storm.yaml 文件即可。

配置项	说明
Zookeeper 相关配置	
storm.zookeeper.servers	Zookeeper 的地址
storm.zookeeper.port	Zookeeper 的端口号
storm.zookeeper.root	Zookeeper 的路径
本地工作目录的配置	
storm.local.dir	请配置本地磁盘的工作路径，该路径用来存放集群运行时的一些心跳信息，配置文件，相关 jar 包等
集群 master 的配置	
nimbus.host	请指定一个机器的 IP 地址用来做集群的 master，负责任务的调度。
关键进程内存控制	
nimbus.childopts	配置 Master 进程使用的内存大小，默认配置为 1024m，如果集群规模较大，建议多配置些内存

supervisor.childopts	配置 Supervisor 进程使用的内存大小, supervisor 会在每台工作机上启动, 用来管理它自身所在机器的 worker 进程的启动, 停止, 以及心跳的发送。
http.childopts	配置查询接口服务进程使用的内存大小, 默认配置为 1024m, 如果集群规模较大, 建议多配置些内存。

2. YDB 配置

配置文件分别为 conf 目录下的 ydb_default.yaml 与 ydb_site.yaml 文件。

其中 ydb_default.yaml 为默认配置文件, 请勿修改, 仅配置 ydb_site.yaml 文件即可。

配置项	说明
ydb.hdfs.path	配置 hdfs 目录, 用来存储索引文件以及表的配置文件
ydb.reader.rawdata.hdfs.path	配置 hdfs 目录, 存储上传的文本格式的数据文件。
hadoop.conf.dir	配置 Hadoop 集群的 conf 路径
topology.worker.childopts	配置每个 ydb 的 worker 的进程使用的内存, 内存的大小直接决定性能。
ydb.reader.speed.limit.file	配置指定 hdfs 中的一个文件, 系统每隔 20 分钟会读取该文件的数值, 该数值用于控制 reader 的消费速度。否则系统默认使用“ydb.reader.speed.limit.kb”配置的值来控制 reader 的消费速度, 此配置一旦系统启动不能更改。
ydb.reader.rawdata.start.delay.secs	用于控制集群启动后 reader 等待多少秒开始消费数据, 默认为集群启动 300 秒后开始消费数据。
ydb.topoplgy.worker.count	用于控制集群启动的进程数量, 请根据实际的机器数量以及内存大小, 配置合适的进程数 (一般 64g 内存的机器配置成机器的数量乘以 4)。
ydb.topoplgy.task.count	集群启动的 task 数量, 一个 task 对应一个线程。每个线程可以管理分区的一个索引分片 (一般 64g 内存的机器配置成机器数量乘以 8)
其他配置项请参考 ydb_default.yaml 文件里的注释	

启动集群

1. 启动 nimbus(master)

nimbus 是集群的主控节点 (即 master) 用于提交任务, 分配集群任务, 集群监控等。

nimbus 必须在 master 所在的机器上启动。

启动命令如下:

```
nohup ./ydb nimbus >nimbus.log 2>&1 &
```

2. 启动 supervisor(slave)

supervisor 负责接受 nimbus 分配的任务, 管理属于自己的 worker 进程。

supervisor 必须在每台 slave 机器上启动

启动命令如下:

```
nohup ./ydb supervisor >supervisor.log 2>&1 &
```

3. 启动查询接口

用于查看集群状态以及查询数据。

在接口机上启动

启动命令如下：

```
nohup ./ydb http 8080 >http.log 2>&1 &
```

启动完毕后，打开如下地址，确保所有的 supervisor 都启动成功

<http://xx.168.112.129:8080/supervisor>

启动 YDB 服务

启动 YDB 服务就是启动 worker，worker 里运行 YDB 的具体逻辑，worker 中会启动一个或多个 task，一个 worker 表示一个进程，一个 task 表示一个线程。worker 与 task 分别由 ydb.topoplgy.worker.count 和 ydb.topoplgy.task.count 指定。

启动 YDBworker 的命令如下：

```
./ydb start
```

执行成功后，可以通过 jps 命令看到是否有 worker 进程启动

通过查询接口可以看到启动的 worker 列表

<http://192.168.112.129:8080/worker>

停止 YDB 服务

```
./ydb kill
```

停止集群

1. 停止接口机

```
jps |grep YdbHttpServer|awk '{printf("%s\n",$1)}'|xargs kill
```
2. 停止 supervisor(需要每台机器上执行)

```
jps |grep Supervisor|awk '{printf("%s\n",$1)}'|xargs kill
```
3. 停止 numbus

```
jps |grep NimbusServer|awk '{printf("%s\n",$1)}'|xargs kill
```

集群使用

创建表

打开接口地址 <http://192.168.112.129:8080/createtable>

数据建表语句（请根据实际情况进行实际调整）

```
create table ydbexample(  
  indexnum int,
```


label string,
 usagerage int,
 clickcount long,
 paymoney double,
 price float,
 content text,
 contentcjck textckj
)

类型	解释
string	字符串类型，该类型不分词，通常用来存储比较短的字符串，如类目
int	整形 32 位
long	整形 64 位
double	Double 类型
float	Float 类型
text	字符串类型，单字分词，通常用来存储长文本
textckj	字符串类型，CJK 分词，通常用来存储长文本

数据导入

请使用如下目录导入 example 数据

~ /hadoop-2.6.2/bin/hadoop fs -put ../data_example/jsondata_example.txt /data/myntest/jsondata_example.txt

curl "http://192.168.112.129:8080/insert?taskid=0&hdfsfile=/data/myntest/jsondata_example.txt"

注：hadoop 的 put 命令，用于将本地的数据文件 put 到 hdfs 集群上

请求的 url 用于将 hdfs 集群上的数据导入到 YDB 中（导入完成到能查到数据会有几秒至几分钟的延迟）

导入数据的文件格式为文本文件，一条数据为一行，数据格式为 JSON 格式，请参考 data_example 目录下的 jsondata_example.txt 的示例文件。

```
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 0,
    "label": "1_0",
    "usage": 10,
    "clickcount": 0,
    "paymoney": 0,
    "price": 0,
    "content": "0 0 0"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 1,
    "label": "1_1",
    "usage": 11,
    "clickcount": 1,
    "paymoney": 1.033,
    "price": 1.03,
    "content": "1 1 1"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 2,
    "label": "1_2",
    "usage": 12,
    "clickcount": 2,
    "paymoney": 2.066,
    "price": 2.06,
    "content": "2 2 2"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 3,
    "label": "1_3",
    "usage": 13,
    "clickcount": 3,
    "paymoney": 3.099,
    "price": 3.09,
    "content": "3 3 3"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 4,
    "label": "1_4",
    "usage": 14,
    "clickcount": 4,
    "paymoney": 4.132,
    "price": 4.12,
    "content": "4 4 4"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 5,
    "label": "1_5",
    "usage": 15,
    "clickcount": 5,
    "paymoney": 5.165,
    "price": 5.15,
    "content": "5 5 5"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 6,
    "label": "1_6",
    "usage": 16,
    "clickcount": 6,
    "paymoney": 6.198,
    "price": 6.18,
    "content": "6 6 6"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 7,
    "label": "1_7",
    "usage": 17,
    "clickcount": 7,
    "paymoney": 7.231,
    "price": 7.21,
    "content": "7 7 7"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 8,
    "label": "1_8",
    "usage": 18,
    "clickcount": 8,
    "paymoney": 8.264,
    "price": 8.24,
    "content": "8 8 8"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 9,
    "label": "1_9",
    "usage": 19,
    "clickcount": 9,
    "paymoney": 9.297,
    "price": 9.27,
    "content": "9 9 9"
  }
}
```

如上图所示，数据里的 tablename 为表的名字，ydbpartition 为表的分区，data 则为每个字段对应的值。分区可以为任意字符串，大部分情况下，分区就是日期。

请知悉，一个表会有多个分区，一个分区下会有多个 task。不同的 task 是分散在每个机器的不同的 worker 进程里面的，查询的时候，他们一起工作。

在调用 insert 导入的时候，需要指定文件所导入的 task 的序号。如上文示例，我们通过 taskid 来指定序号，task 的个数由配置文件里的 ydb.topoplgly.task.count 配置项指定。

数据查询

打开如下的 url 即可以进行数据的查询，具体 SQL 的使用方法请参考下文

```
http://192.168.112.129:8080/sql?sql=select indexnum,label from ydbexample where ydbpartition='20151011'
limit 0,100
```

过期分区的索引清理

如果数据有生命周期，需要定期清理过期的数据。

调用如下的 url 删除指定分区的数据。

```
http://192.168.112.129:8080/droppartition?tablename=ydbexample&partitions=20151011
```

切记不要删除正在导入数据的分区，一般建议在分区停止写入后的第二天再进行清理。

日常清理

- 每台机器的 logs 目录里的日志是按照日期生成的，随着集群运行天数的增多，日志数量会越来越多，需要定期清理
- ydb_site.yaml 中的 ydb.reader.rawdata.hdfs.path 配置的目录，系统会保留已经消费过的数据文件到 finish 目录，按照日期分的目录，系统不会删除已经消费过的数据，这样做的目的是保留原始数据，一旦出现重大故障或者需要将数据导出到其他系统，这些原始数据十分有用。

但是随着时间的日期月累，这个目录会越来越大，文件数量会越来越多，如果系统每次上传的文件都很小，但文件数量很多，会对 hadoop 的 namenode 造成较大的压力，所以建议定期将里面的小文件合并成一个大文件。如做了备份或确定不再需要这些数据，也可以将其清理掉。

YDB SQL 使用说明

分区：

随着时间的日积月累，单个索引会越来越大，从而导致系统瓶颈。YDB 不会将全部的数据都完整的创建在一个索引中，YDB 会对数据进行分区，分区的方式由用户来定义，可以按照日期分区，也可以按照某些固定的 HASH 方式来分区。

一条数据的分区，在导入的时候直接指定，如下图红色部分所示。

```
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 0,
    "label": "1_0",
    "userage": 10,
    "clickcount": 0,
    "paymoney": 0,
    "price": 0,
    "content": "0 0 0"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 1,
    "label": "1_1",
    "userage": 11,
    "clickcount": 1,
    "paymoney": 1.033,
    "price": 1.03,
    "content": "1 1 1"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 2,
    "label": "1_2",
    "userage": 12,
    "clickcount": 2,
    "paymoney": 2.066,
    "price": 2.06,
    "content": "2 2 2"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 3,
    "label": "1_3",
    "userage": 13,
    "clickcount": 3,
    "paymoney": 3.099,
    "price": 3.09,
    "content": "3 3 3"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 4,
    "label": "1_4",
    "userage": 14,
    "clickcount": 4,
    "paymoney": 4.132,
    "price": 4.12,
    "content": "4 4 4"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 5,
    "label": "1_5",
    "userage": 15,
    "clickcount": 5,
    "paymoney": 5.165,
    "price": 5.15,
    "content": "5 5 5"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 6,
    "label": "1_6",
    "userage": 16,
    "clickcount": 6,
    "paymoney": 6.198,
    "price": 6.18,
    "content": "6 6 6"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 7,
    "label": "1_7",
    "userage": 17,
    "clickcount": 7,
    "paymoney": 7.231,
    "price": 7.21,
    "content": "7 7 7"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 8,
    "label": "1_8",
    "userage": 18,
    "clickcount": 8,
    "paymoney": 8.264,
    "price": 8.24,
    "content": "8 8 8"
  }
}
{
  "tablename": "ydbexample",
  "ydbpartition": "ydb_part_15",
  "data": {
    "indexnum": 9,
    "label": "1_9",
    "userage": 19,
    "clickcount": 9,
    "paymoney": 9.297,
    "price": 9.27,
    "content": "9 9 9"
  }
}
```

如果按照日期进行分区，每天就会生成一个分区，如需查询哪天的数据，就去对应的分区中检索，其他的分区则闲置。

YDB 的 SQL 需要通过 `ydbpartition` 来指定分区; SQL 查询必须要设置分区，而且要写在 SQL 的最外层。

设置分区例子如下：

```
ydbpartition ='20140928'  
ydbpartition in ('20140928','20140927')
```

Worker:

YDB 会在集群中启动多个 `worker`，`worker` 里运行 YDB 的具体逻辑，`worker` 中会启动一个或多个 `task`，一个 `worker` 表示一个进程，一个 `task` 表示一个线程。`worker` 与 `task` 分别由 `ydb.topology.worker.count` 和 `ydb.topology.task.count` 指定。

Task:

一个 `task` 对应一个线程。一个 `task` 可以同时管理多个分区，但是只能管理每个分区里的一个分片。

Merger Server:

每个 `worker` 会附带一个 `Merger Server`，`MergerServer` 会合并其下的每个 `Task` 的查询结果。

SQL 写法与例子

1. 查看数据例子

```
select indexnum,label from ydbexample where ydbpartition='20151011' limit 0,100
```

2. 对某个列进行排序

```
select indexnum,label from ydbexample where ydbpartition='20151011' order by indexnum desc limit 0,100
```

3. 常见过滤条件的写法

● 支持 `in` 操作，例子如下

```
indexnum in (1,2,3)
```

● `>`,`<`,`>=`,`<=`,区间查询的写法

```
clickcount >=10 and clickcount <=11
```

对于带有范围的过滤筛选，推荐使用下面这种含有 `TO` 的写法，能提升查询效率,注意两种括号的区别，前者不包含边界值，但后者则包含

```
indexnum like '{0 TO 11}'
```

```
indexnum like '([10 TO 11])'
```

● 不等于的写法

```
label<>'l_14' and label<>'l_15'
```

- **全文检索**

要使用全文检索功能，需要将数据列的格式配置成 `text` 或 `textcjk` 类型,这两个类型会进行分词处理，分词的列可以进行全文检索但是数据分词后也就不能再进行 `group by` 与 `order by` 了,通常来说分词的列的值都比较长 `group by` 与 `order by` 也失去了意义

例如:

如果有两条 `content` 的内容是 "中华人民共和国"

如果是 `text` 类型，采用单字分词，分词结果为

中-华-人-民-共-和-国

`content='中'` `content=' 中华'` `content=' 中人'` 都可以匹配到这条记录

如果是 `textcjk` 类型这个类型采用二元分词 分词结果就是

中华-华人-人民-民共-共和-和国

`contentcjk=' 中'` 匹配不到结果

`contentcjk=' 中华'` 可以匹配到结果

`contentcjk=' 中人'` 匹配不到结果

- **模糊匹配**

对于 `string` 类型的列，他的数据时没有分词的，可以使用 `like` 功能进行模糊匹配，但是性能特别低，要谨慎使用。分词的列不能使用 `like` 进行模糊匹配，但可以使用上面的全文检索。

`label like 'l*'` 注意如果该列排重后的值数量特别多，禁止将通配符写在最前面，否则查询性能很低

`label like 'l_??'` 如要要进行类似 QQ 号码的定长匹配，可以用?来代替*, *是匹配 0~多个字符，而?则表示只匹配一个字符

- **过滤条件可以进行 and 与 or 的组合（但最外层必须是 and ）且可以通过括号表达多层的逻辑关系,例子如下**

`indexnum='1' or indexnum='2' or (clickcount >=5 and clickcount <=11)`

4. 对表进行简单的 count 统计

```
select count(*),count(indexnum) from ydbexample where ydbpartion='20151011' limit 0,100
```

注意下 `count(*)`与 `count(dddwid)`的区别，前者不管列的值是否是 `null`，计数都会加 1，而且因为不需要读取字段的值，性能很高，后者需要判断改字段的值是否为 `null`，如果是 `null` 则不会进行累加计数，所以大部分场景都推荐使用 `count(*)`

5. 简单的统计函数

```
select sum(clickcount),avg(clickcount),max(clickcount),min(clickcount) from ydbexample
where ydbpartion='20151011' limit 0,100
```

6. 分类汇总的写法 group by

```
select label,count(*),sum(clickcount) from ydbexample where ydbpartion='20151011' group by
label limit 0,100
```

- **多个列进行 group by 分类汇总**

```
select label,userage,count(*),sum(clickcount) from ydbexample where ydbpartion='20151011'
group by label,userage limit 0,100
```

7. 对分类汇总的结果进行排序

```
select label,userage,count(*),sum(clickcount) from ydbexample where
```

```
ydbpartition='20151011' group by label,userage order by userage limit 0,100
select label,userage,count(*),sum(clickcount) as amt from ydbexample where
ydbpartition='20151011' group by label,userage order by amt desc,count(*) desc limit 0,100
```

8. 多列排序的写法

```
select label,indexnum from ydbexample where ydbpartition='20151011' order by label
desc,indexnum asc limit 0,1000
```

```
select label,indexnum,count(label) from ydbexample where ydbpartition='20151011'
group by label,indexnum order by label asc,indexnum asc limit 0,1000
```

```
select label,indexnum,count(label) from ydbexample where ydbpartition='20151011'
group by label,indexnum order by count(label) desc,indexnum desc limit 0,1000
```

Ydb 内置高级功能的写法

1. 导出功能

使用后导出功能，order by、limit、group by 均失效，只能对查询明细进行导出。

- **ydb_site.xml 配置每个 task 最大允许的导入条数**

ydb.export.max.return.docset.size:1000000

- **sql 中参数说明**

export.path 导出路径

export.joinchar 导出时列与列之间的分隔符默认\t

export.max.return.docset.size 用于覆盖配置文件中的参数，配置每个 task 最大允许的导入条数

Sql 写法例子

- **导出结果到/data/ycloud/ydb/001 目录**

```
select ydbsz_lyn,indexnum,label from ydbexample where ydbpartition in (20151011) and
ydbkv='export.path:/data/ycloud/ydb/001' limit 0,10
```

- **配置分隔符**

```
select ydbsz_lyn,indexnum,label from ydbexample where ydbpartition in (20151011) and
ydbkv='export.path:/data/ycloud/ydb/002' and ydbkv='export.joinchar:;' limit 0,10
```

- **覆盖配置文件中的默认导出条数限制 ydb.export.max.return.docset.size**

```
select ydbsz_lyn,indexnum,label,content,contentcjk from ydbexample where ydbpartition in
(20151011) and ydbkv='export.path:/data/ycloud/ydb/003' and
ydbkv='export.max.return.docset.size:30' limit 0,10
```

- **导出完毕后，会在 hdfs 目录里看到如下的文件**

Browse Directory						
/data/ycloud/ydb/001						Go!
Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	yannianmu	supergroup	798.29 KB	1	256 MB	20151203113550_0_2890756250_4146418301
-rw-r--r--	yannianmu	supergroup	832.19 KB	1	256 MB	20151203113550_0_766594887_2879288672
-rw-r--r--	yannianmu	supergroup	385.5 KB	1	256 MB	20151203113551_0_3010292331_4070974009
-rw-r--r--	yannianmu	supergroup	880.11 KB	1	256 MB	20151203113551_0_7417547_744380328
-rw-r--r--	yannianmu	supergroup	362.26 KB	1	256 MB	20151203113551_1_1878947087_1983773766
-rw-r--r--	yannianmu	supergroup	684.19 KB	1	256 MB	20151203113551_1_1991454885_2896194238
-rw-r--r--	yannianmu	supergroup	832.63 KB	1	256 MB	20151203113551_2_1795892680_3729488910
-rw-r--r--	yannianmu	supergroup	78.78 KB	1	256 MB	20151203113551_2_2095116046_172026216
-rw-r--r--	yannianmu	supergroup	929.11 KB	1	256 MB	20151203113551_3_3725010600_1910517084
-rw-r--r--	yannianmu	supergroup	787.33 KB	1	256 MB	20151203113552_1_1304832843_3410492183
-rw-r--r--	yannianmu	supergroup	828.34 KB	1	256 MB	20151203113552_1_3476697707_1146293138
-rw-r--r--	yannianmu	supergroup	880.71 KB	1	256 MB	20151203113552_2_2622524323_1134225913
-rw-r--r--	yannianmu	supergroup	836.92 KB	1	256 MB	20151203113552_2_2747016799_1028870895
-rw-r--r--	yannianmu	supergroup	876.48 KB	1	256 MB	20151203113552_3_2768958343_1680909881
-rw-r--r--	yannianmu	supergroup	154.93 KB	1	256 MB	20151203113552_4_1960632392_3274844985

文本文件的内容格式如下，默认按照\t分割



经典配置参数

如果您的集群规模较大，可以考虑变更为如下的配置，性能会有显著的提升。

硬件配置

64G 内存 2T*12 的磁盘 4core 集群规模 50~100 台，如果有条件更换成 SSD 硬盘，会显著的
提升随机读取的效率

storm.yaml 配置

配置项	建议值
nimbus.childopts	分配 2g 内存
http.childopts	分配 8g 或更多的内存
supervisor.childopts	分配 512m 内存

ydb_site.yaml 配置

配置项	建议值
ydb.topology.worker.count	机器数量*4
ydb.topology.task.count	机器数量*8
ydb.directory.blockbuffer.percent: 32	如果您内存富余较多，可以使用更多的内存作为 hdfs 的 buffer
ydb.realtime.binlog.usebinlog: "true" ydb.realtime.binlog.usehsync: "true" ydb.realtime.binlog.rolltimelen.ms: 120000 ydb.realtime.binlog.sync.interval: 1	<p>如果您的业务不担心数据丢失，可以关闭 binlog（预写日志 WAL）功能。</p> <p>如果设置成 true，则 binlog 的 flush 会使用 hdfs 的 hsync 方法，否则使用 hflush 刷新数据</p> <p>日志进行切割的时间,20 分钟切割一次</p> <p>每隔几条记录，调用一次 flush，如果在 flush 调用前进程意外终止，未 flush 的数据有可能丢失,但是频繁的调用 hflush()方法会导致系统变慢建议调大此值。</p>
topology.worker.childopts	分配 8g 以上内存
ydb.reader.rawdata.start.delay.secs: 600	集群启动后延迟 600 秒开始消费数据创建索引
ydb.reader.speed.limit.kb: 512000 ydb.reader.speed.limit.file: "/hdfs/xxx"	<p>流量限速 50M</p> <p>如果觉得这个值要动态改变，可以采用此参数指向 hdfs 中的一个文件，系统会每隔 20 分钟读一次配置</p>
ydb.reader.rawdata.hdfs.monitor.secs: 120	调整为 120 秒，每 120 秒去 hdfs 上检查一次是否有新的数据过来需要消费
ydb.realtime.binlog.sync.interval: 1024	建议调整为 1024，每间隔 1024 条同步一次 binlog 到 hdfs，但是意味着如果在同步前进程异常中断，1024 条数据丢失
ydb.realtime.doclist.bufsize: 1024 ydb.realtime.doclist.flush.secs: 30 ydb.realtime.doclist.flush.thread.check.secs: 300 ydb.realtime.doclist.flush.process.threads: 8	<p>积累 1024 条数据,doclist 区域会刷新到 buffer 区</p> <p>积累 30 秒，doclist 区域会刷新到 buffer 区</p> <p>如果一直没有数据进来，间隔 300 秒启动线程检查一次，是否应该刷新到 buffer 区域</p> <p>采用 8 个线程来刷新</p>
####【buffer 区域的索引控制】#### ydb.realtime.buffer.ram.writepartition.each.percent: 1 ydb.realtime.buffer.flush.secs: 60 ydb.realtime.buffer.flush.thread.check.secs: 300 ydb.realtime.buffer.maxcount: 64 ydb.realtime.buffer.merger.factor: 16	<p>每个写入的 partition 分片会使用 1%的内存进行 buffer，如果并发写入的分区特别多，内存有可能不够</p> <p>积累 60 秒，buffer 区域的索引会刷新到 ram 区域</p> <p>如果一直没有数据进来，间隔 300 秒启动线程检查一次，是否应该刷新到 ram 区域</p> <p>Buffer 区域最多积累 64 个索引</p> <p>Buffer 区域每次合并索引的时候一次最多合并 16 个索引</p>
####【内存区域的索引控制】#### ydb.realtime.ram.ram.writepartition.each.percent: 3 ydb.realtime.ram.localflush.secs: 600 ydb.realtime.ram.ram.maxcount: 64 ydb.realtime.ram.merger.factor: 16	<p>每个写入的 partition 分片会使用 3%的内存进行 Ram</p> <p>在合并过程中如果 ydb.index.ram2disk.premerger.isuse 配置为 true 有可能占用 2 个 6%的内存</p> <p>如果并发写入的分区特别多，内存有可能不够</p> <p>积累 600 秒，Ram 区域的索引会刷新到 hdfs 上</p> <p>Ram 区域最多积累 64 个索引</p> <p>Ram 区域每次合并索引的时候一次最多合并 16 个索引</p>
####【hdfs 上的索引合并因子的控制】#### ydb.realtime.index.disk.maxcount: 64 ydb.realtime.index.disk.merger.factor: 16 ydb.index.ram2disk.premerger.isuse: "true"	<p>Hdfs 区域最多积累 64 个索引</p> <p>Hdfs 区域每次合并索引的时候一次最多合并 16 个索引</p> <p>在将 ram 区域的索引合并到磁盘上，是否先进行预合并（当内存富余的时候推荐使用,可以节省磁盘 IO）</p>
ydb.realtime.index.maxfilesize: 536870912	单个索引的最大大小为 512m

内存较小情况下的配置

在测试的情况下，需要在一个小内存的虚拟机里进行测试，那么默认的参数需要进行相应的缩小,否则内存有可能不够。

如果机器内存充盈，足够给每个 **worker** 进程分配 **6g** 以上的内存，建议使用默认配置，或者使用上面的经典配置，以下配置是在只能分配给 **jvm 200m** 到 **300m** 的内存环境下测试用的，性能并不好

storm.yaml 配置

配置项	建议值
nimbus.childopts	分配 128m 内存
http.childopts	分配 128m 内存
supervisor.childopts	分配 128m 内存

ydb_site.yaml 配置

配置项	建议值
topology.worker.childopts	根据真实的内存大小，调小
ydb.reader.speed.limit.kb: 1024 ydb.reader.speed.limit.file: "/hdfs/xxx"	流量限速 如果觉得这个值要动态改变，可以采用此参数指向 hdfs 中的一个文件，系统会每间隔 20 分钟读一次配置
ydb.index.ram2disk.premerger.isuse: "false"	内存中的索引 dump 到磁盘上的时候，不进行在内存中的预合并
ydb.index.reader.maxopen: 128 ydb.index.reader.min.open.secs: 60	允许同时打开的索引的个数（内存太小的时候打开太多内存不够） 在 reader 被使用的 60 秒内，reader 不会被释放掉（一个 reader 被使用后，至少存活 30 秒）
ydb.docvalues.cache.size:128	控制列存储的列的值的 cache，同时打开列的个数
ydb.directory.blockbuffer.percent:4	在 hdfs 读写的时候，使用 jvm 总内存的百分之多少进行 buffer
ydb.realtime.doclist.buffsize: 256	不要积累太多的文档，256 个就开始创建索引（如果单个文件体积太大，也要缩减）
ydb.realtime.buffer.ram.writepartion.each.percent:1 ydb.realtime.ram.ram.writepartion.each.percent: 2	Buffer 区域与 Ram 区域 每个正在写入的索引分片使用 jvm 总内存的百分之多少进行缓冲
ydb.hdfs.files.maxopen.count:256	同时打开的 hdfs 的文件数量的最大数量
ydb.realtime.index.lazydelete.timelen.secs: 600 ydb.realtime.index.purgerreader.timelen.secs: 60	合并成新的索引后之前的索引文件要删除掉，但由于可能有其他进程正在读取，故采用延迟删除索引的方式，这里配置延迟多久删除索引。 合并成新的索引后，之前的已经打开的索引依然会占用资源，该配置用于设置延迟多少秒释放资源。
ydb.request.stagetwo.snapshot.cache.secs: 60	由于采用两段式查询，两次查询之间要保存 snapshot，snapshot 的保留时限要缩短
ydb.fq.cache.size:128 ydb.fq.cache: "false"	禁掉 where 查询部分的 cache 或者将 cache 个数调小

其他注意事项

过载保护（熔断机制）

为了防止大量的意外请求导致的服务器宕机，YDB 的接口采用熔断机制。

我们可以用 `ydb.request.max.pral` 这个选项来控制允许多少个并发请求同时执行，如果请求数量超过并发数，则按照 FIFO 的机制在请求队列中进行排队处理。为了防止因为队列太长导致的内存溢出，如果请求队列的长度超过 128 个(可以通过 `ydb.request.max.queue.size` 修改)后续的请求会被拒绝。

数据导入注意事项

在导入数据的时候，请注意：

第一：导入数据要均匀的分配给每个分片（Task），防止出现数据倾斜的问题。

第二：同时导入的分区数量不要超过 32 个（非分片个数），如果同时导入的分区太多，打开的写索引的 writer 也太多，会导致性能不好，因为每个 writer 都会占用一定的内存 buffer，也容易导致内存用满。

目前允许同时处于写的状态的 write 数量为 512(配置项为 `ydb.index.max.writer`)个,但是实际情况下，同时产生 512 个 writer 内存是不够的，所以要严格控制写入的分区数量。如果同时写入的分区数量超过 `ydb.index.max.writer` 配置的值，就会有不经常使用的 writer 被关闭，下次有数据写入的时候会重新打开，这样就会造成索引频繁的被打开关闭，严重影响性能。

数据延迟的处理

如果配置项 `ydb.reader.rawdata.hdfs.path` 指定的 hdfs 中的目录，有较多的文件堆积，说明数据消费有延迟，如果 CPU 负载不高，可以通过修改 `ydb.reader.speed.limit.kb: 1024` 与 `ydb.reader.speed.limit.file: "/hdfs/xxx"` 这两个配置来提高消费的速度，但是如果机器负载本身很高，只能通过优化其他配置参数或者扩容的方式解决。

索引的管理方式与性能的关系

因为资源的限制，如果一个集群中的索引数量太多（比如保存了 1 年的索引），集群现有资源不能满足打开全部的索引，YDB 采用 LRU 的方式管理索引，对于那些经常使用的索引，在内存中是一直打开的，不经常使用的索引则是处于关闭状态，存放在 HDFS 中，不占用任何的内存资源。

如果要查询一个不经常使用的分区，索引需要重新打开，故查询会比较慢。

某些场景，循环扫描不同的分区索引，会导致每个分区都没有热点，会造成性能严重下降。

关于查询最多返回 前 10000 条数据的说明

因数据要经过层层 merger，最终汇总到一个节点上，merger 节点对内存，网络，性能都有要求，故无论是不同的数据检索，还是分类汇总，我们对返回的结果限制在 1W 条内。这里的 1W 条并非只是对匹配到的前 1W 条数据进行计算，而是对所有匹配的数据计算完毕后的结果只返回前 1W 条。在下个版本中，我们会提供将数据导出到 hdfs 上的功能，导出功能则没有 1W 条结果限制。

1. 普通的数据检索，只能查看前 1W 条结果，可以通过排序，选择排名最高的前 1W 条。
2. 对于分类汇总 (group by)，也只能返回前 1W 条，由于 group by 在计算的过程中需要迁移很多的 group 数据，然后集中汇总，这需要很多内存，故我们采取了近似计算的实现，每个 group 都单独计算自己的前 1W 个分组，然后按照分组进行汇总与排序，这样在某些特殊情况下会有排序误差，但是经过两段式查询，我们可以确保统计的结果是正确的，按照列名的排序也是准确，当组数超过一万组的时候，按照统计方式的排序是有误差的（但统计值是正确的）。

如何让 YDB 性能提升 10 倍（千亿规模集群 DBA 必读）

Ydb 工作的时候会将 SQL 分成几个部分 “where 后的条件过滤”、“分组与统计”、“排序”。

系统最先执行的是 “where 后的条件过滤”，先检索到与 where 条件匹配的记录。比如说全表有 10 亿条记录，经过与 where 条件匹配后，可能还剩下 500 万条记录，然后对剩下的 500 万条进行 “分组与统计”、“排序”。

由于 YDB 采用大索引技术，索引的匹配速度比较快，故后续的 “分组与统计”、“排序” 是决定计算性能的关键。

默认情况是将 “where 过滤后”，满足条件的全部匹配的数据都取出来，再进行 “分组与统计”、“排序”。如果满足条件的数据仅仅是原始数据的一小部分，不会影响性能，也能满足大部分的场景需求，但是如果有 “严重的数据倾斜”，或者直接对全表数据进行 “分组与统计”、“排序” 会严重的消耗系统的资源，这会导致系统变慢。

如果业务模式每次检索的数据 都是原始数据很小的一部分，那么可以将这个 “where 过滤” **改进一下**，不像之前那样每次都取出全部匹配的数据，而是每次仅取其中的一小部分，比如经过 “where 过滤” 后依然有几千万的数据，但我们只取其中的 1 万条，然后在这 1 万条的基础上在进一步的 “分组与统计”、“排序”。这样除了极大的提高性能，也防止因为用户的误操作（写错了 SQL），导致系统崩溃。

要做这样的配置，需要在 ydb_site.yaml 中配置限制的值，如下面的例子，我们限制每个 SQL 最多匹配 10000 条数据参与计算。

```
ydb.fq.max.return.docset.size: 10000
```

注：这里的 10000 针对一个索引片段。一个分区下会有 ydb.topoplgy.task.count 个 task，每个 task 下 hdfs 目录里最多会有 ydb.realtime.index.disk.maxcount 个索引片段，内存区域最多

会有 `ydb.realtime.ram.ram.maxcount` 个索引片段，所以一次查询最多参与计算的记录数是索引片段的个数乘以 10000。

如果个别的业务要求突破这个限制（比如说仅仅 `count` 计算，对系统性能没什么大的影响），只要 SQL 再加上如下条件即可突破。

放到 SQL 里的例子如下

```
select count(label),count(*) from ydbexample where ydbpartition='20151011' and
ydbkv='max.return.docset.size:100000000'
limit 0,100
```

系统启动阶段会碰到的问题

系统刚启动的时候，历史索引还存放在 `hdfs` 中，并没有打开，首次打开索引需要加载很多资源。故系统刚启动的瞬间，查询会比较慢。

如上次系统是因意外终止（硬件损坏，机器重启等），内存中的数据还没有来得及同步到 `HDFS` 中，所以在首次启动的时候需要用 `binlog(WAL write ahead log)` 恢复数据，这个恢复时间取决于恢复数据的量，所以首次查询的时候，有可能出现卡顿的现象或者出现数据变少的现象，出现这种现象说明系统正在恢复数据，等数据恢复完毕后这种现象会自行消失，所以建议系统启动后等待一段时间（10-15 分钟）后再开始查询。

心跳的延迟

页面的心跳信息也不是实时采集的，而是在系统启动后，采用异步的方式，进行异步的抓取。故在启动的开始阶段，有些表的心跳看不到是正常的。

因为心跳是异步采集，所以当新创建一个表或者增加一个新区时，表或分区的心跳并不能实时看到。

同样因为心跳是异步采集，实时导入数据时，数据的条数、占用空间大小、碎片个数的显示都有较大的延迟。

之所以采用异步采集，是因为我们不想因为心跳本身占用系统太多的资源。

心跳频率可控的参数是 `ydb.hb.task.interval.secs`，他只能加快心跳的频率，但是对新建表以及新增分区无效。

文件数量与磁盘 IO 负载的权衡-复合索引

在 Hadoop 运行的时候，hdfs 的元数据都是放到内存里面的，所以小文件太多，会对 hadoop 的 namenode 造成较大的压力。

针对这种问题，YDB 采用两种方式管理索引，普通索引和复合索引。

普通索引是索引的原始模式，文件数量多，但是对磁盘压力小。

复合索引是将普通索引中的文件合并成几个文件，文件数量较少，查询性能会略高，但因多一次合并的过程，比普通索引多一倍的 IO。

YDB 默认使用复合索引，可以通过如下的配置使用普通索引。
ydb-site.yaml 中，将 ydb.index.compoundfile.hdfs 配置为 false 即可。

下图为普通索引与复合索引的文件区别

普通索引的文件列表	<table><tr><th>Name</th></tr><tr><td>_0.fdt</td></tr><tr><td>_0.fdx</td></tr><tr><td>_0.fnm</td></tr><tr><td>_0.nvd</td></tr><tr><td>_0.nvm</td></tr><tr><td>_0.si</td></tr><tr><td>_0_Lucene50_0.doc</td></tr><tr><td>_0_Lucene50_0.dvd</td></tr><tr><td>_0_Lucene50_0.dvm</td></tr><tr><td>_0_Lucene50_0.tim</td></tr><tr><td>_0_Lucene50_0.tip</td></tr><tr><td>segments_1</td></tr><tr><td>txid</td></tr></table>	Name	_0.fdt	_0.fdx	_0.fnm	_0.nvd	_0.nvm	_0.si	_0_Lucene50_0.doc	_0_Lucene50_0.dvd	_0_Lucene50_0.dvm	_0_Lucene50_0.tim	_0_Lucene50_0.tip	segments_1	txid
Name															
_0.fdt															
_0.fdx															
_0.fnm															
_0.nvd															
_0.nvm															
_0.si															
_0_Lucene50_0.doc															
_0_Lucene50_0.dvd															
_0_Lucene50_0.dvm															
_0_Lucene50_0.tim															
_0_Lucene50_0.tip															
segments_1															
txid															
使用复合索引的文件列表	<table><tr><th>Name</th></tr><tr><td>_1.cfe</td></tr><tr><td>_1.cfs</td></tr><tr><td>_1.si</td></tr><tr><td>segments_1</td></tr><tr><td>txid</td></tr></table>	Name	_1.cfe	_1.cfs	_1.si	segments_1	txid								
Name															
_1.cfe															
_1.cfs															
_1.si															
segments_1															
txid															

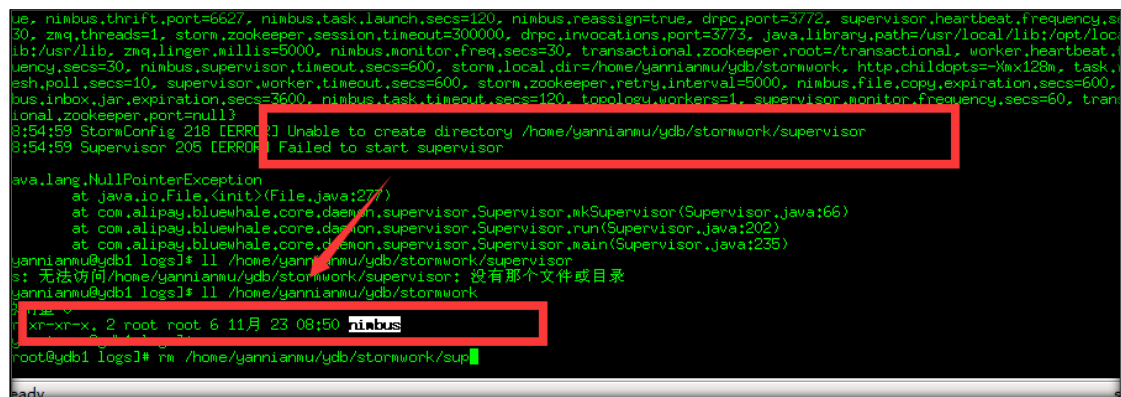
异常关机或者结束进程导致的集群启动不了的解决办法

1. 停掉所有进程，然后清理 `storm.local.dir` 所指向的本地目录
2. 检查目录权限，确保本地与 `hdfs` 有读写权限。

注：

有时因误用了 `root` 账户启动会导致本地目录没有写权限，这时应杀掉所有的进程并清理 `storm.local.dir` 指向的本地目录，修正所有的目录权限后，重新启动。

下图为典型因目录权限导致的不能启动的日志截图。



```
... nimbus.thrift.port=6627, nimbus.task.launch.secs=120, nimbus.reassign=true, drpc.port=3772, supervisor.heartbeat.frequency.s
30, zmq.threads=1, storm.zookeeper.session.timeout=300000, drpc.invocations.port=3773, java.library.path=/usr/local/lib:/opt/loc
lib:/usr/lib, zmq.linger.millis=5000, nimbus.monitor.freq.secs=30, transactional.zookeeper.root=/transactional, worker.heartbeat.f
ency.secs=30, nimbus.supervisor.timeout.secs=600, storm.local.dir=/home/yannianmu/ydb/stormwork, http.childopts=-Xmx128m, task.l
ash.poll.secs=10, supervisor.worker.timeout.secs=600, storm.zookeeper.retry.interval=5000, nimbus.file.copy.expiration.secs=600,
bus.inbox.jar.expiration.secs=3600, nimbus.task.timeout.secs=120, topology.workers=1, supervisor.monitor.frequency.secs=60, tran
sactional.zookeeper.port=null
8:54:59 StormConfig 218 [ERROR] Unable to create directory /home/yannianmu/ydb/stormwork/supervisor
8:54:59 Supervisor 205 [ERROR] Failed to start supervisor

java.lang.NullPointerException
    at java.io.File.<init>(File.java:277)
    at com.alipay.bluewhale.core.daemon.supervisor.Supervisor.mkSupervisor(Supervisor.java:66)
    at com.alipay.bluewhale.core.daemon.supervisor.Supervisor.run(Supervisor.java:202)
    at com.alipay.bluewhale.core.daemon.supervisor.Supervisor.main(Supervisor.java:235)
yannianmu@ydb1 logs]$ ll /home/yannianmu/ydb/stormwork/supervisor
ls: 无法访问/home/yannianmu/ydb/stormwork/supervisor: 没有那个文件或目录
yannianmu@ydb1 logs]$ ll /home/yannianmu/ydb/stormwork
total 4
-rwxr-xr-x 2 root root 6 11月 23 08:50 nimbus
root@ydb1 logs]$ rm /home/yannianmu/ydb/stormwork/sup
```

YDB 二期功能预览

免责声明：二期功能是延云 YDB 未来开发的重点方向，但延云并不保证二期会发布全部的功能，延云 YDB 保留对二期上线功能做变更的权利。

1. 全量数据导出
当前版本只能查询前 10000 条记录，二期可以将数据导出到 `hdfs` 上，全量数据导出则没有前 10000 条的限制。该功能在广告系统的人群推送，人群导出等场景可广泛应用。
2. 基于文件的过滤
当前版本只能根据 `SQL` 进行过滤，过滤的条件个数有限（200 个），在某些场景下，用户可能会有一个几千万条记录的客户名单，不想开放给其他人使用（比如说淘宝某个店铺的客户名单），但是想与我们的系统关联，分析此客户名单的特征。该功能如果与上面的数据导出功能组合使用，在有多表关联的业务场景上有非常大的使用意义。
3. 近似 `count(distinct)`
可以对某一列的值进行排重后计算，但是由于其中涉及大量的数据迁移以及内存，这里采用抽样计算。
4. 快速连续匹配
当前版本分词的列不支持连续目录匹配，部分的列可以使用 `like` 进行连续匹配，但效率太低。高性能的联系匹配功能在二期会开放。

5. 支持 update
二期开放。
6. 更多的数据源
目前仅支持从 **hdfs** 上导入数据，二期可以支持更多的数据源，包括 **kafka**、**metaq** 以及自定义数据源。
7. 不同的表不同的分片资源
目前所有的表分片资源均统一，下个版本，可以针对不同的表配置不同的分片资源。
有些业务的表确实比较小，都统一分配统一的 **task** 资源，并不是很合理。
注：当前版本虽然不支持，但是可以借助导出数据的时候只给部分的分片分配数据来达到分配资源的目的。

企业版 YDB

默认的 YDB 版本为免费使用的基础版，基础版支持运行 5 个 Task(每个 Task 对应一个线程)。如果您想运行更大规模的集群，请联系沈阳延云云计算技术有限公司，届时我们会有专业的团队为您提供大数据解决方案以及优质的技术支持。

联系方式：

网址：www.ycloud.net.cn

电话：024-83653716

手机：17099831107

微信：

qq：

企业版 7 天免费试用授权码生成

打开 <http://xx.168.112.129:8080/au>

会输出授权码，将其贴到 **ydb_site.yaml** 中，然后重启 Ydb 即可

输出的授权码示例如下

```
ydb.au.sn: "1964218139-1762173003-2069092389-3512374086"
ydb.au.company: "ycloud.net.cn"
ydb.au.mail: "myn@163.com"
ydb.au.start: "20151127"
ydb.au.end: "20151205"
ydb.au.pral: 10000
```