

# 时序评测工具 **TS-Benchmark** 使用手册 **V2.0**

中国人民大学信息学院

# 目 录

1 背景介绍 .....	1
1.1 风力发电数据处理流程.....	1
1.2 数据生成模型.....	2
2 测试说明 .....	2
2.1 负载说明.....	2
2.2 测试指标说明.....	3
2.3 TS-Benchmark 说明.....	3
3 运行环境说明 .....	4
3.1 运行环境.....	4
4 安装说明 .....	4
4.1 JDK 安装.....	4
4.2 maven 安装 .....	5
4.3 git 安装 .....	5
4.4 InfluxDB 安装.....	7
4.5 TimescaleDB 安装 .....	7
4.6 Druid 安装.....	11
4.7 OpenTSDB 安装.....	12
5 部署说明 .....	14
5.1 数据导入说明.....	14
5.1.1 InfluxDB 数据导入说明 .....	15
5.1.2 TimescaleDB 数据导入说明 .....	15
5.1.3 Druid 数据导入说明.....	16
5.1.4 OpenTSDB 数据导入说明 .....	16
5.2 执行说明.....	17
5.3 结果说明.....	19
6 五类查询说明 .....	19
6.1 第一类查询.....	19
6.2 第二类查询.....	20

**6.3 第三类查询.....20**

**6.4 第四类查询.....20**

**6.5 第五类查询.....21**

# 1 背景介绍

时间序列数据广泛应用于供应链，库存数据分析和智能制造等场景中，已有许多时间序列数据库系统来存储，管理和查询大量时间序列数据。我们观察到，时间序列数据库的现有基准测试集中在复杂分析的工作负载上，例如模式匹配和趋势预测，其性能可能会受到数据分析算法而不是数据管理系统的高度影响。但是，在时间序列数据库的许多实际应用中，人们对性能指标(例如数据注入吞吐量和查询处理时间)更感兴趣，仍然需要一个基准来广泛比较此类指标中时间序列数据库的性能。TS-Benchmark 便是一个时间序列数据库基准，它利用风力发电场景下的设备生成时间序列数据。工作负载分为三类：批量数据加载，流数据注入和历史数据访问(对于典型的时间序列查询)。我们利用此工具测试比较了四个具有代表性的时间序列数据库，包括 InfluxDB、TimescaleDB，Druid 和 OpenTSDB。

## 1.1 风力发电数据处理流程

该基准是基于风力发电应用中的故障监视和诊断案例建模的。基本数据处理流程如图 1-1 所示。对于风场的所有设备（风力涡轮机），每 7 秒捕获一次所有传感器的读数，每个设备的传感器数量多达数百个。一个风场中有 50 台设备（涡轮增压器），许多风场将收集到的数据分别发送到数据中心。大型风电公司（例如 GoldWind2）需要处理来自全球成千上万台设备和数千个风场的数据。流传感器数据被路由到两个不同的系统，一个是用于仪表板监视应用程序的流处理系统，另一个是用于记录时间序列数据并稍后查询以进行进一步分析的时间序列数据库，我们的基准便是第二个系统。

时间序列数据库负责持久化流数据，以便可以有效地检索它们以进行查询和进一步分析。通常需要三种主要方案来访问时间序列数据库中的历史数据：

(1)探索时间序列数据以进行问题识别。例如，当仪表板上有警报时，某些传感器的读数超过阈值，我们需要将更多数据获取到上层软件，以识别出真正的问题。在某些情况下，我们想知道异常情况发生的频率，这还需要在覆盖故障点的特定时间段内获取数据。

(2)查询数据以对设备进行操作优化。例如，用户可能对某个风电场的哪个涡轮机最近表现最好感兴趣。这可以通过获取与发电相关的传感器的主要读数进行深入分析来实现，以便找到一些理想参数来优化涡轮机，以提高发电效率。

(3)检索数据以进行问题预测。当用户对预测在不久的将来会发生什么问题感兴趣时，通常需要风电场的更多数据来进行线性回归等深度分析。

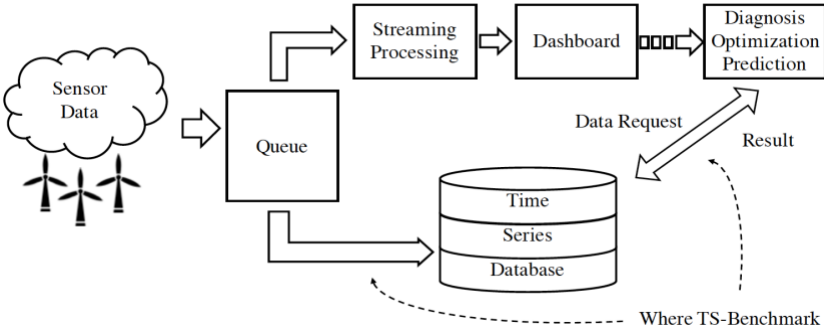


图 1-1 风力发电场景时间序列数据处理流程

1.2 数据生成模型

每个风力涡轮机上都有数百个传感器，它们可以分为两类，用于监视环境的传感器和用于监视涡轮机本身的传感器。 第一类包括用于测量温度，湿度，风向，风速等的传感器。第二类包括用于俯仰角，迎风角，角速度，电压，电流，装机功率，标称功率等的传感器。还有许多传感器可测量涡轮机内不同位置的温度，湿度和振动频率。

为了模拟应用场景并更有效地生成数据，我们从风力发电机 GoldWind 中获取了一些种子数据。基于此，我们采用文献中的通用技术来构建数据生成器。首先，我们使用 GoldWind 提供的风能数据（为期一年）训练时间序列的 ARIMA 模型。其次，我们使用训练好的 ARIMA 模型为每个涡轮机连续生成风数据。

2 测试说明

2.1 负载说明

(1)数据加载：将以文本格式生成具有基本数据集，数据集包含来自两个风场的数据，每个风场有 50 个设备，每个设备有 50 个传感器，数据集涵盖了一

周的时间序列数据，包含  $4.32 \times 10^8$  个数据点，数据集大小为 3.9GB。在数据加载测试期间，使用其导入命令或专门为其编写的数据加载程序将生成的文件导入到目标数据库。

(2)连续数据注入：将数据加载到目标时间序列数据库系统后，将应用数据注入工作负载来测试数据库的写入性能。在实际情况下，时间序列数据库需要及时处理大量并发写入请求。

数据注入任务以多线程方式实现。每个线程负责注入一个风场的数据。线程数等于风场数。每个风场有 50 个设备，每个设备有 50 个传感器。对于每个线程，每 7 秒将打包设备的所有传感器数据的快照并将其发送到目标数据库。我们逐渐将线程数从 1 增加到 512，以模拟目标系统上不断增长的工作负载。预计目标系统将在某个时候饱和，然后将无法及时保留更多数据点，我们根据每秒成功写入的数据点来衡量吞吐量。

(3)数据获取：此基准包括五类查询，会在第 6 部分详细说明。

## 2.2 测试指标说明

(1)写入吞吐量：当流数据在多线程下连续注入目标数据库时，将测量目标数据库的总体吞吐量。请注意，每个线程每 7 秒发送一次写请求，我们将附加吞吐量定义为每个时间单位成功注入数据库的数据点数。

(2)查询响应时间：当目标数据库接收查询请求并对其进行处理时，将测量目标数据库的为查询服务的响应时间。对于每种查询，我们使用不同的参数运行 10 个查询计算平均响应时间。

## 2.3 TS-Benchmark 说明

TS-Benchmark 是用 Java 实现的，该工具的整体架构如图 2-1 所示，每个模块的简要介绍如下。数据生成模块生成基本数据集（用于数据加载）并将其保存到磁盘。加载模块测试目标系统的加载能力，该目标系统将本机批处理文件导入到目标数据库。查询生成模块生成查询并将其提交到查询模块，然后查询模块对目标数据库执行查询。注入模块将数据流定向到目标系统。数据库适配器模块通过提供统一的接口来隐藏各种目标系统的差异，我们可以通过实现继承接口的驱动程序来测试新数据库。目前我们已经为 InfluxDB，TimescaleDB，Druid 和 OpenTSDB 实现了驱动程序。

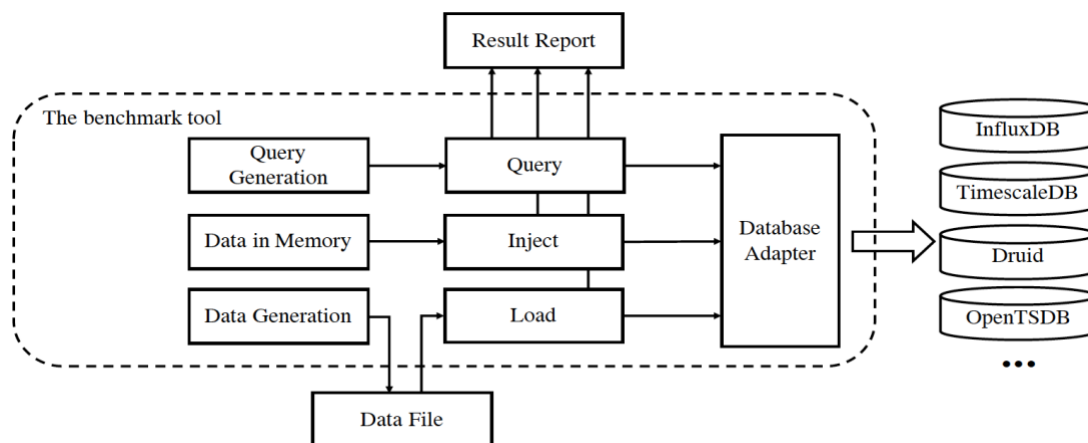


图 2-1 TS-Benchmark 整体架构

### 3 运行环境说明

#### 3.1 运行环境

Linux 系统，本文以 centos 7 为例；jdk  $\geq 1.8$ ；maven  $\geq 3$ ；git  $\geq 1.8$ 。

### 4 安装说明

#### 4.1 JDK 安装

本评测工具是基于 JDK1.8 开发，使用者需要保证运行系统内安装 JDK1.8 。使用者可在如下链接点击相应的版本下载：

<http://www.oracle.com/technetwork/java/javase/archive-139210.html>

注意要下载 linux 版本，并且根据自己系统位数选择相应的.tar.gz 文件。比如我们的 CentOS 系统为 64 位，故选择 jdk-8u144-linux-x64.tar.gz 下载。

假设下载的 jdk-8u144-linux-x64.tar.gz 文件在/usr/java 目录下。

1) 解压文件：

```
tar -zxvf jdk-8u144-linux-x64.tar.gz
```

2)配置环境变量：

```
vim /etc/profile
```

进入文本编辑状态下，光标走到文件最后一行，键盘按下 i，进入插入状态，写入：

```
JAVA_HOME=/usr/java/jdk1.8.0_144
```

```
PATH=$PATH:JAVA_HOME/bin
```

```
CLASS_PATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar:$JRE_
HOME/lib
```

然后键盘输入 ESC，wq，保存。

3)使环境变量生效：

```
source /etc/profile
```

4)查看版本号看是否安装成功：

```
java -version
```

如果版本显示为 1.8 即为安装成功。

## 4.2 maven 安装

1)通过 wget 命令下载安装包：

```
wget http://mirrors.shu.edu.cn/apache/maven/maven-3/3.5.4/binaries/apache-maven-
3.5.4-bin.tar.gz
```

2)通过 tar 命令解压到当前目录：

```
tar -zxvf apache-maven-3.5.4-bin.tar.gz
```

假设解压后的目录为/usr/share/maven，

3)在/etc/profile 中保存 Maven 的环境变量：

```
export M2_HOME=/usr/share/maven
```

```
export PATH=$PATH:$M2_HOME/bin
```

4)通过 source 使配置文件生效：

```
source /etc/profile
```

5)查看版本号看是否安装成功：

```
mvn -version
```

## 4.3 git 安装

1)查看已有 git 版本：

```
git --version
```

如果已有 git 版本<小于 1.8，则需要卸载低版本 git。

2)卸载低版本 git：

```
yum remove git
```

3) 查看 yum 源仓库的 git 信息



yum info git

```
Available Packages
Name      : git
Arch      : x86_64
Version   : 1.7.1
Release   : 9.el6_9
Size      : 4.6 M
Repo      : updates
Summary   : Fast Version Control System
URL       : http://git-scm.com/
License   : GPLv2
Description: Git is a fast, scalable, distributed revision control system with an
           : unusually rich command set that provides both high-level operations
           : and full access to internals.
           :
           : The git rpm installs the core tools with minimal dependencies. To
           : install all git packages, including tools for integrating with other
           : SCMs, install the git-all meta-package.
http://blog.csdn.net/zxy987872674
```

若如上图所示 git 版本小于 1.8，则需要下载源码进行安装，可参考第 5 步，若 yum 源仓库里的 git 版本大于 1.8，则可以用 yum 安装，可参考第 4 步。

4)yum 源安装 git:

```
yum install git
```

5)下载源码安装 git:

先安装依赖库:

```
yum install curl-devel expat-devel gettext-devel openssl-devel zlib-devel
```

```
yum install gcc perl-ExtUtils-MakeMaker
```

下载 git 源码:

```
wget https://github.com/git/git/archive/v2.9.2.tar.gz
```

解压:

```
tar -zxvf git-2.9.2.tar.gz
```

安装 git:

```
cd git-2.9.2
```

```
make prefix=/usr/local/git all
```

```
make prefix=/usr/local/git install
```

添加到环境变量:

```
echo "export PATH=$PATH:/usr/local/git/bin" >> /etc/bashrc
```

```
source /etc/bashrc
```

查看版本号:

```
git --version
```

## 4.4 InfluxDB 安装

安装 influxDB 时需要 root 用户或者管理员权限。

1)InfluxDB 使用如下命令进行安装:

```
wget https://dl.influxdata.com/influxdb/releases/influxdb-1.5.3.x86_64.rpm
```

```
sudo yum localinstall influxdb-1.5.3.x86_64.rpm
```

默认情况下, InfluxDB 会使用如下的端口:

TCP8086 端口是服务器监听端口, 对 HTTP API 响应, TCP8088 端口是 RPC 服务端口, 用于数据备份和保存。

更多的端口使用细节和配置方式可以在配置文件/etc/influxdb/influxdb.conf 进行了解和设置。

2)用如下命令修改配置文件:

```
vim /etc/influxdb/influxdb.conf
```

```
reporting-disabled = true ( 这个要设置真, 关闭定时上传数据到  
influxdata.com)
```

```
#bind-address = ":8086"(这个需要自己手动添加, 指定 http 的连接操作端  
口, 默认为 8086)
```

```
[admin]
```

```
# Determines whether the admin service is enabled.
```

```
enabled = true (web 管理界面, 1.1 版本以上默认关闭。需要的话, 可以  
手动打开)
```

```
# The default bind address used by the admin service.
```

```
bind-address = ":8083" (web 服务界面的端口)
```

3)用如下命令启动:

```
sudo systemctl start influxdb
```

输入 influx 即可进入 InfluxDB 数据库。

## 4.5 TimescaleDB 安装

TimescaleDB 安装可以按照如下官网安装教程安装:

<https://docs.timescale.com/v1.2/getting-started/installation/rhel-centos/installation-yum>

TimescaleDB 是由 PostgreSQL 支持的开源时间序列数据库，作为 PG 插件的形式存在，如果要使用到相关时序的功能，需要安装上此插件。安装方法有以下两种，第一种方式为 yum 安装，第二种方式为源码安装，以下以 yum 安装为例说明。

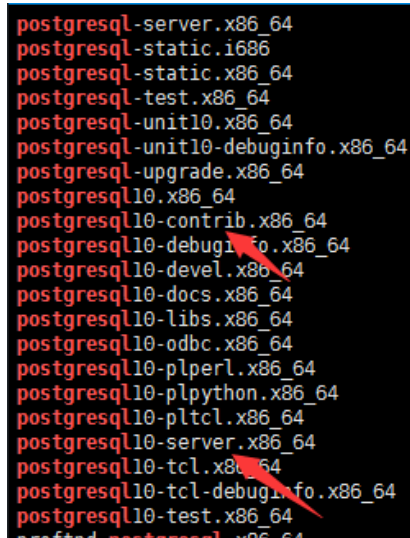
安装 TimescaleDB 需要先安装 PostgreSQL，下面给出 yum 方式安装 PostgreSQL10.5 的方法。

1)安装 PostgreSQL 的 rpm:

yum install https://download.postgresql.org/pub/repos/yum/10/redhat/rhel-7-x86\_64/pgdg-centos10-10-2.noarch.rpm -y

2)查看 PostgreSQL 源

yum list | grep postgresql



```
postgresql-server.x86_64
postgresql-static.i686
postgresql-static.x86_64
postgresql-test.x86_64
postgresql-unit10.x86_64
postgresql-unit10-debuginfo.x86_64
postgresql-upgrade.x86_64
postgresql10.x86_64
postgresql10-contrib.x86_64
postgresql10-debuginfo.x86_64
postgresql10-devel.x86_64
postgresql10-docs.x86_64
postgresql10-libs.x86_64
postgresql10-odbc.x86_64
postgresql10-plperl.x86_64
postgresql10-plpython.x86_64
postgresql10-pltcl.x86_64
postgresql10-server.x86_64
postgresql10-tcl.x86_64
postgresql10-tcl-debuginfo.x86_64
postgresql10-test.x86_64
postgresql10-test.x86_64
```

3)安装 postgresql10-contrib 和 postgresql10-server

yum install postgresql10-contrib postgresql10-server -y

4)初始化数据库

PostgreSQL 安装目录是/usr/pgsql-10,而 Postgresql 的数据目录是 /var/lib/pgsql/版本号/data 目录

在这里，如果在装系统开始分配 var 空间足够大则可以继续，如果分配 var 空间不够，我们需要更改数据目录，在这里，我们假设 var 空间足够大。直接开始初始化。

/usr/pgsql-10/bin/postgresql-10-setup initdb

## 5)启动数据库

`sudo systemctl start postgresql-10`

## 6)登录 PostgreSQL 并设置密码

PostgreSQL 在安装时默认添加用户 postgres，输入如下命令进入数据库：

`sudo -u postgres psql postgres`

会让你设置密码，输入密码即可。

然后\q 退出。

## 7)修改配置文件

默认情况下 PostgreSQL 是不用密码不支持远程登录的，我们需要修改配置文件：

`vim /var/lib/pgsql/10/data/pg_hba.conf`

原本是这样：

```
# -----  
#  
# If you want to allow non-local connections, you need to add more  
# "host" records. In that case you will also need to make PostgreSQL  
# listen on a non-local interface via the listen_addresses  
# configuration parameter, or via the -i or -h command line switches.  
  
# TYPE  DATABASE      USER      ADDRESS      METHOD  
  
# "local" is for Unix domain socket connections only  
local   all             all                                     peer  
# IPv4 local connections:  
host    all             all        127.0.0.1/32  ident  
# IPv6 local connections:  
host    all             all        ::1/128      ident  
# Allow replication connections from localhost, by a user with the  
# replication privilege.  
local   replication     all                                     peer  
host    replication     all        127.0.0.1/32  ident  
host    replication     all        ::1/128      ident
```

需要改成如下所示：

```
# "host" records. In that case you will also need to make PostgreSQL  
# listen on a non-local interface via the listen_addresses  
# configuration parameter, or via the -i or -h command line switches.  
  
# TYPE  DATABASE      USER      ADDRESS      METHOD  
  
# "local" is for Unix domain socket connections only  
local   all             all                                     md5  
# IPv4 local connections:  
host    all             all        127.0.0.1/32  md5  
# IPv6 local connections:  
host    all             all        ::1/128      md5  
# Allow replication connections from localhost, by a user with the  
# replication privilege.  
#local   replication     all                                     md5  
#host    replication     all        127.0.0.1/32  md5  
#host    replication     all        ::1/128      md5  
host    all             all        0.0.0.0/0     md5
```

修改远程访问权限：

`vim /var/lib/pgsql/10/data/postgresql.conf`

原本如下图所示：

```
#-----  
# CONNECTIONS AND AUTHENTICATION  
#-----  
  
# - Connection Settings -  
  
#listen_addresses = 'localhost'          # what IP address  
#                                     # to listen for  
#                                     # connections  
#                                     # (change requires  
#                                     # restart)  
#port = 5432                             # (change requires  
#                                     # restart)  
max_connections = 100                   # (change requires  
#                                     # restart)  
#superuser_reserved_connections = 3      # (change requires  
#                                     # restart)  
#unix_socket_directories = '/var/run/postgresql; /tmp'
```

需要修改为如下图所示：

```
#-----  
# CONNECTIONS AND AUTHENTICATION  
#-----  
  
# - Connection Settings -  
  
listen_addresses = '*'                  # what IP address  
#                                     # to listen for  
#                                     # connections  
#                                     # (change requires  
#                                     # restart)  
port = 5432                             # (change requires  
#                                     # restart)  
max_connections = 100                   # (change requires  
#                                     # restart)  
#superuser_reserved_connections = 3      # (change requires  
#                                     # restart)
```

然后重启数据库：

```
systemctl restart postgresql-10
```

如果你的 PostgreSQL 是使用 yum 网络安装，则可以同样使用该方法安装 TimescaleDB 插件，就根据官网上给出的方法，方便简洁。

1)创建 yum 源：

TimescaleDB 官方给出了一个 yum 源，添加这个 repo 文件之后即可使用：

```
# Add our repo
```

```
sudo tee /etc/yum.repos.d/timescale_timescaledb.repo <<EOL
```

```
[timescale_timescaledb]
```

```
name=timescale_timescaledb
```

```
baseurl=https://packagecloud.io/timescale/timescaledb/el/7/\$basearch
```

```
repo_gpgcheck=1
```

```
gpgcheck=0
```

```
enabled=1
```

```
gpgkey=https://packagecloud.io/timescale/timescaledb/gpgkey
```

```
sslverify=1
```

```
sslcert=/etc/pki/tls/certs/ca-bundle.crt
```

```
metadata_expire=300
```

EOL

2)更新 yum 源之后，安装插件，注意相关版本：

```
sudo yum update -y
```

```
# Now install appropriate package for PG version
```

```
sudo yum install -y timescaledb-postgresql-10
```

```
sudo timescaledb-tune (两个 y, 一个 n)
```

3)配置 TimescaleDB

每一个 TimescaleDB 都是跟一个已有的 database 绑定的，需要手动创建一下 database。

```
sudo -u postgres psql postgres
```

创建一个数据库

```
CREATE database ruc_test;
```

登录新数据库

```
\c ruc_test
```

安装 TimescaleDB 扩展

```
CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;
```

\q 退出以后重新登录即可。

## 4.6 Druid 安装

1) 下载 Druid 安装包

```
wget https://www.apache.org/dyn/closer.cgi?path=/incubator/druid/0.13.0-incubating/apache-druid-0.13.0-incubating-bin.tar.gz
```

2) 解压

```
tar -xzf apache-druid-0.13.0-incubating-bin.tar.gz
```

```
cd apache-druid-0.13.0-incubating
```

3) 下载 Zookeeper

```
curl https://archive.apache.org/dist/zookeeper/zookeeper-3.4.11/zookeeper-3.4.11.tar.gz -o zookeeper-3.4.11.tar.gz
```

```
tar -xzf zookeeper-3.4.11.tar.gz
```

```
mv zookeeper-3.4.11 zk
```

#### 4) 启动 druid

```
bin/supervise -c quickstart/tutorial/conf/tutorial-cluster.conf
```

之后写数据需要安装 Tranquility，所以在这里一起安装，步骤如下：

##### 1) 按 Ctrl+C 停掉 Druid 服务

##### 2) 在 Druid 根目录即 apache-druid-0.13.0-incubating 目录下下载 Tranquility

```
curl http://static.druid.io/tranquility/releases/tranquility-distribution-0.8.2.tgz -o  
tranquility-distribution-0.8.2.tgz
```

```
tar -xzf tranquility-distribution-0.8.2.tgz
```

```
mv tranquility-distribution-0.8.2 tranquility
```

##### 3) 修改 quickstart/tutorial/conf/tutorial-cluster.conf 文件，把 tranquility-server 那一行前的 # 去掉

```
# Uncomment to use Tranquility Server
```

```
!p95 tranquility-server tranquility/bin/tranquility server -configFile
```

```
quickstart/tutorial/conf/tranquility/wikipedia-server.json -
```

```
Ddruid.extensions.loadList=[]
```

##### 4) 启动 Druid + Tranquility

```
bin/supervise -c quickstart/tutorial/conf/tutorial-cluster.conf
```

## 4.7 OpenTSDB 安装

因为 OpenTSDB 的后端存储使用的是 HBase，所以我们需要先安装 HBase，HBase 需要依赖 Zookeeper。

##### 1) 安装 Zookeeper

```
curl https://archive.apache.org/dist/zookeeper/zookeeper-3.4.14/zookeeper-  
3.4.14.tar.gz -o zookeeper-3.4.14.tar.gz
```

```
tar -xzf zookeeper-3.4.14.tar.gz
```

##### 2) 安装 HBase

下载安装包：

```
wget http://apache.fayea.com/hbase/stable/hbase-1.4.9-bin.tar.gz
```

解压文件：

```
tar zxvf hbase-1.4.9-bin.tar.gz
```

解压完成后当前目录会有个 hbase-1.4.9 目录。

```
cd hbase-1.4.9
```

```
vim conf/hbase-env.sh
```

设置 export JAVA\_HOME 为 JDK 安装路径，可通过 echo \$JAVA\_HOME 查看。

```
vim hbase-1.4.9/conf/hbase-site.xml
```

设置 hbase.rootdir 为你的 Hbase 安装路径，hbase.zookeeper.property.dataDir 为你的 Zookeeper 安装路径。

启动 Hbase：

```
cd ..
```

```
./bin/start-hbase.sh
```

可用 jps 命令查看 Hbase 是否安装成功。

### 3)安装 OpenTSDB

下载安装包：

wget <https://github.com/OpenTSDB/opentsdb/releases/download/v2.2.0/opentsdb-2.2.0.tar.gz>

解压：

```
tar zxvf opentsdb-2.2.0.tar.gz
```

编译源码

```
cd opentsdb-2.2.0
```

```
./build.sh
```

```
make install
```

创建表：

第一次启动 OpenTSDB 需要 HBase 的支持，所以首先需要创建必要的 HBase 表格，命令如下：

```
env COMPRESSION=none HBASE_HOME=/usr/local/hbase-1.4.9  
/root/opentsdb-2.2.0/src/create_table.sh
```



其中 HBASE\_HOME 是本机的 hbase 的安装文件，后面脚本的路径也应该改为本机的路径，这条命令执行完会生成几张表。

修改配置文件

```
vim opentsdb.conf
```

```
tsd.core.auto_create_metrics=true
```

```
tsd.http.request.enable_chunked=true
```

```
tsd.http.request.max_chunk=65535000
```

```
tsd.storage.fix_duplicates=true
```

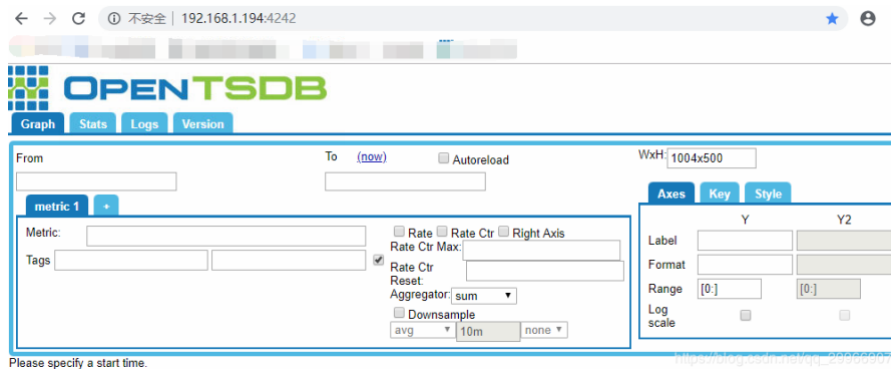
```
tsd.network.port=4242
```

启动 opentsdb 服务：

```
./build/tsdb tsd --config=./opentsdb.conf
```

或者 service opentsdb start

完成后通过 <http://192.168.1.194:4242/> 访问看到页面即为安装成功



## 5 部署说明

### 5.1 数据导入说明

1)编译项目：

```
cd ts-benchmark
```

```
sh build.sh
```

2)生成数据：

```
cd tsdb-test
```

```
sh run.sh
```

注意，这个脚本的默认测试任务是生成数据，也可以进行写和读操作，具体细节请参考 4.2 节的第 5 步骤。

运行完成后在 `tsdb-test` 目录下会有一个 `data/load` 目录，目录下有个文件 `load.data`，接下来就将这个文件的数据分别导入不同的时序数据库。

生成的 `load.data` 数据集包含来自两个风场的数据，每个风场有 50 个设备，每个设备有 50 个传感器，数据集涵盖了一周的时间序列数据，包含  $4.32 \times 10^8$  个数据点，数据集大小为 3.9GB。

### 5.1.1 InfluxDB 数据导入说明

在开始导入数据之前需要在 InfluxDB 中创建名为 `ruc_test` 的数据库。

1) 进入 `tsdb-test/data/load` 目录：

```
cd data/load
```

2) 执行 python 脚本：

```
python generate_influx.py
```

脚本执行完成后，会在当前目录下生成 `influxdb.csv` 文件，这是可以导入 InfluxDB 的文件。

3) 导入数据，在当前目录下执行如下命令：

```
sh csv_dataGen.sh 1
```

数据导入完成后可以输出数据导入的总时间消耗。

### 5.1.2 TimescaleDB 数据导入说明

1) 进入 `tsdb-test/data/load` 目录：

```
cd data/load
```

2) 执行 python 脚本：

```
python generate_timescale.py
```

脚本执行完成后，会在当前目录下生成 `timescaledb.csv` 文件，这是可以导入 TimescaleDB 的文件。

3) 进入项目根目录，即 `tsdb-test/` 目录，执行脚本，创建数据库和表：

```
cd ../..
```

```
sh init_timescaledb.sh
```

4) 导入数据，需要进入 `tsdb-test/data/load` 目录：

```
cd data/load
```

```
sh csv_dataGen.sh 2
```

数据导入完成后可以输出数据导入的总时间消耗。

### 5.1.3 Druid 数据导入说明

1) 进入 tsdb-test/data/load 目录：

```
cd data/load
```

2) 执行脚本：

```
sh druid_dataParse2Json.sh
```

会在当前目录下生成 load\_druid\_json.txt 文件，将 csv 文件转换成可导入 Druid 数据库的 json 文件。

3) 将 tsdb-test/data/load 目录下的 tsbm-druid\_index.json 拷贝到您安装的 druid 目录下，即之前的 apache-druid-0.13.0-incubating/quickstart/tutorial/目录下，在文件中找到如下内容：

```
"ioConfig" : {  
    "type" : "index",  
    "firehose" : {  
        "type" : "local",  
        "baseDir" : "/home/tsbm/tsbm_app/transform/bin",  
        "filter" : "load_druid_json.txt"  
    },  
    "appendToExisting" : false  
},
```

将其中的"baseDir"改为您系统的真正路径。

4) 进入 apache-druid-0.13.0-incubating 根目录，执行如下导入命令，导入之前确保 Druid 服务启动成功：

```
bin/post-index-task --file quickstart/tutorial/tsbm-druid_index.json --url  
http://localhost:8081
```

### 5.1.4 OpenTSDB 数据导入说明

1) 进入 tsdb-test/data/load 目录：

```
cd data/load
```

2)执行脚本:

```
sh opents_dataParse2format.sh
```

会在当前目录下生成 load\_opentsdb\_format.txt 文件, 将 csv 文件转换成可导入 OpenTSDB 数据库的 json 文件。

将生成的 load\_opentsdb\_format.txt 文件拷贝到 OpenTSDB 数据库安装目录下。

3)进入 OpenTSDB 数据库安装目录, 执行如下命令

```
build/tsdb import --config=./opentsdb.conf load_opentsdb_format.txt
```

数据导入完成后会关闭 OpenTSDB 服务, 需要重启。

## 5.2 执行说明

1)下载 ts-benchmark 源码:

```
git clone git@github.com:dbiir/ts-benchmark.git
```

下载完成后会在当前目录下多一个 ts-benchmark 目录。

2)编译项目:

```
cd ts-benchmark
```

```
sh build.sh
```

3)下载 tsdb-test 源码:

```
git clone git@github.com:sunape/tsdb-test.git
```

下载完成后会在当前目录下多一个 tsdb-test 目录。

```
cd tsdb-test
```

```
vim pom.xml
```

会看到如下依赖:

```
<dependency>
    <groupId>cn.edu.ruc</groupId>
    <artifactId>TS-BM</artifactId>
    <version>1.0</version>
</dependency>
```

如果您想新建自己的项目, 一定要在 pom.xml 文件中加入上述依赖。

```
sh run.sh
```

注意，测试任务默认运行的是生成数据，可在 `run.sh` 中修改测试模式，将 `TEST_METHOD` 改为其他值即可。

### 5)设置测试的数据库、测试任务:

```
vim run.sh
```

找到如下两行:

# 1:influxdb ;2:timescaledb ;3:iotdb ;4 opentsdb;5 druid

DB\_CODE=1

# 0: generate, 1: i, w, r, 2 w, r

TEST\_METHOD=0

可以看到 DB\_CODE=1 表示测试的是 InfluxDB 数据库，DB\_CODE=2 表示测试的是 TimescaleDB 数据库，DB\_CODE=3 表示测试的是 IotDB 数据库，DB\_CODE=4 表示测试的是 OpentsDB 数据库，DB\_CODE=5 表示测试的是 Druid 数据库。

TEST\_METHOD=0 表示测试任务为生成数据，TEST\_METHOD=1 表示测试任务为向数据库写入和读取数据，2 和 1 的功能相同，您可以直接选择 1。

执行结束以后，可以根据提示查看执行结果：

```
#query result  
query1 94  
query2 101  
query3 4601  
query4 14  
query5 111  
  
<<<<<<<<query end finished<<<<<<<<<<  
test finished  
test result in file /home/tsbm/V2/tsdb-test/result//1571206344308.txt
```

## 5.3 结果说明

```
##append result
###append farm++ result
farm 1 33515
farm 2 79125
farm 4 135266
farm 8 265396
farm 16 542186
farm 32 1045095
farm 64 1982747
farm 128 3643917
farm 256 5267693
farm 512 5530504
###append device++ result
device 50 273930
device 100 378499
device 150 424440
device 200 474051
device 250 514489
device 300 542541

##query result
query1 19
query2 2206
query3 515
query4 568
query5 3234
```

结果文件示例如上图，`##append result` 是写入的结果，`###append farm++ result` 是不同线程数下的写入吞吐量，如图中所示，风场数即线程数为 1 时，吞吐量为 33515points/s，线程数为 2 时，吞吐量为 79125points/s，线程数为 4 时，吞吐量为 135266points/s，可以此类推查看每个线程数对应的写入吞吐量。

`###append device++ result` 是把线程数固定为 8，改变每个线程数的写入数据量也即改变每个风场的设备数时的写入吞吐量，如图中所示，风场数为 8，设备数为 50 时，吞吐量为 273930points/s；风场数为 8，设备数为 100 时，吞吐量为 378499points/s。

`##query result` 是五类查询的结果响应时间，每类查询的结果都是测试了 10 次以后取平均值。如图中所示，第一类查询的响应时间为 19ms，第二类查询的响应时间为 2206ms，第三类查询的响应时间为 515ms，第四类查询的响应时间为 568ms，第五类查询的响应时间为 3234ms。

## 6 五类查询说明

### 6.1 第一类查询

当仪表板上有警报时，例如某个传感器的读数超过阈值，人们需要向上层软件获取更多数据以识别问题所在，可以表达为如下查询：

```

SELECT * FROM ts_table
WHERE f_id = ?1
AND d_id = ?2
AND s_id = ?3
AND time >time_start
AND time <?(time_start + 1 hour);

```

其中，f\_id 为风场号，d\_id 为设备号，s\_id 为传感器号，time 为时间戳。?1，?2，time\_start 为用户提供的参数，时间窗口宽度固定为 1 个小时。

## 6.2 第二类查询

有时可能还需要计算同一问题发生的频率。让当前标识问题的时间戳（例如，当前读数超过某个阈值）成为问题点，我们需要获取问题点附近的数据，以便上层软件进行一些深入的分析，例如聚类。当某些传感器读数超过阈值时，该查询将标识时间戳，可表达为如下查询：

```

SELECT s_id, d_id, f_id, time FROM ts_table
WHERE f_id = ?1
AND s_id = ?2
AND time >time_start
AND time <(time_start + 1 week)
AND value >?3;

```

其中，?3 为一些传感器读数的阈值，请注意，时间窗口的宽度固定为一周。在这种情况下，设备不限于一个设备号，需要一个风场所有设备的相同传感器读数。

## 6.3 第三类查询

有时我们可能需要在一周内每小时每小时地计算每个设备的功率传感器的平均值（测量产生的电量），可表达为如下查询：

```

SELECT f_id, d_id, avg(value) FROM ts_table
WHERE f_id = ?1
AND time >time_start
AND time <time_start + 1 week
AND s_id = "power"
GROUP BY f_id, d_id, hour (time);

```

## 6.4 第四类查询

有时我们需要返回与发电相关的主要传感器的读数，以便在上层软件中进行深度分析。执行第三类查询后，我们发现在几个小时内，设备 ID 会产生最大的电量。基于此，我们希望同时获取所有设备的主要传感器的读数，以进行深入分析，可表达为如下查询：

```
SELECT * FROM ts_table  
WHERE f_id = ?1  
AND s_id in (S1, S2, S3, S4, S5)  
AND time >time_start  
AND time <(time_start + 1 hour)  
ORDER BY d_id;
```

## 6.5 第五类查询

在许多情况下，我们不仅关注眼前的问题，而且关注不久的将来可能发生的潜在问题。通过上层软件进行面向预测的深度分析（例如线性回归）需要风场的最新数据，第五类查询就用于此目的。它在一个时间窗口内获取同一风场所有设备的传感器读数，可表达为如下查询：

```
SELECT * FROM ts_table  
WHERE f_id = ?1  
AND time >time_start  
AND time <(time_start + 15 min);
```