面向电网流式数据处理的性能测试平台①

覃 海, 姬 源, 周思明, 沈冠全

(贵州电网公司电力调度控制中心, 贵阳 550002)

摘 要: 提出一种面向电网流式数据处理的性能测试平台的系统架构及关键技术. 针对电网流式数据的业务场景特点, 测试平台支持基于浏览器的测试脚本设计和测试场景设计, 并支持测试任务的全生命周期管理和高可用保障. 为了满足大规模负载测试需求, 测试平台提供基于 Linux 容器的虚拟化测试资源池, 实现了测试资源的轻量化、弹性管理.

关键词: 流式数据处理; 性能测试; 测试资源管理

Performance Testing Platform of Stream Data Processing for Power Grid

QIN Hai, JI Yuan, ZHOU Si-Ming, SHEN Guan-Quan

(Guizhou Electric Power Grid Dispatching and Control Center, Guiyang 550002, China)

Abstract: In this paper, we present the architecture and technology of a new performance testing platform of the stream data processing for the power grid. This testing platform provides a browser-based test script and test scenario design service. It also supports the full life cycle management and high availability guarantee for the execution of test task. Meanwhile, this platform uses Linux container to build a virtualized testing resource pool, which can achieve the lightweight, flexible management of testing resources for large-scale load testing.

Key words: stream data processing; performance testing; testing resource management

随着我国电力行业的快速发展, 电网规模逐步扩大, 电网结构日趋复杂, 电网调度运行数据信息急剧增加. 如此庞大的系统规模, 如何对大量智能监测终端所产生的大规模流式数据信息进行实时可靠、安全高效的采集、传输、存储和管理就显得十分必要, 也是电网调度自动化系统应用分析的基础保障数据[1-3].

电网流式数据处理系统的高效、稳定运转需要利用系统性能评测工具对系统的软硬件架构、配置进行技术验证和测试保障, 检验系统性能指标是否符合需求, 分析系统的性能瓶颈, 评估系统的资源容量. 性能评测工具通过自动化的评测工具模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试, 并收集分析被测系统的性能表现和硬件资源状态.

本文给出一种面向电网流式数据处理的新型性能测试平台的系统架构及关键技术. 测试平台针对电网流式数据处理的业务场景特点, 提供了基于浏览器的测试脚本设计和测试场景设计支持, 并支持测试任务的全生命周期管理和高可用保障. 为了满足大规模负载测试需求, 测试平台还提供基于 Linux 容器的虚拟化测试资源池, 实现了测试资源的轻量化、弹性管理.

1 系统概述

面向电网流式数据处理的性能测试平台包括性能测试用例设计、测试负载生成及测试资源管理等功能, 平台系统的总体框架如图 1 所示. 其中:

1)性能测试用例设计. 测试平台为用户提供基于浏览器的测试用例设计方式, 其中包括测试脚本设计和测试场景设计, 可以根据不同业务场景的数据特点, 产生符合特定数据传输协议、特定数据发送频率以及满足特定测试目标的测试负载.

2)性能测试任务管理. 测试平台负责测试任务的参数初始化、运行和测试结果收集, 同时, 在测试任务

① 收稿时间:2015-07-16;收到修改稿时间:2015-09-21

执行过程中提供测试资源的高可用保障机制, 保证测试任务的有效完成.

3)性能测试资源管理. 性能测试资源管理主要包括测试资源池管理和测试资源分配. 测试平台提供基于 Linux 容器的虚拟化测试资源池, 在执行测试任务前, 首先评估测试任务的测试资源需求, 然后从测试资源池中申请所需的测试资源节点, 运行测试用例, 产生测试负载.

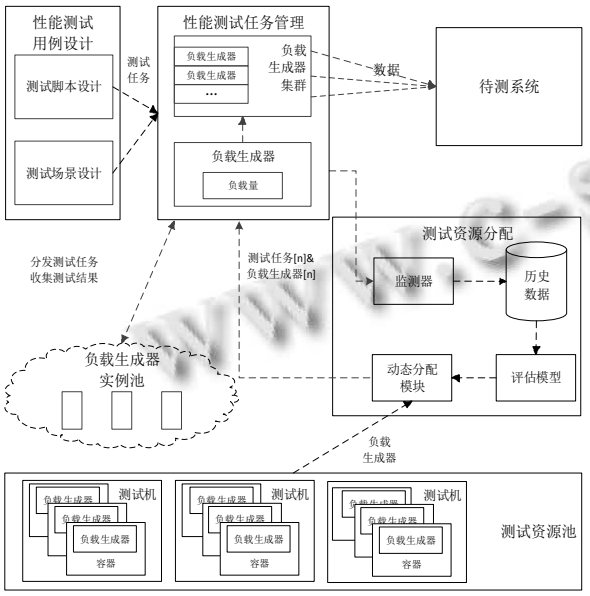

图 1　测试平台总体框图

2　各模块的关键技术与实现

2.1 性能测试用例设计

2.1.1 测试脚本设计

测试脚本用于指定负载生成器的种类、协议、频率、被测系统的地址以及数据产生方式等. 定义基于 XML 的测试脚本格式如下:

1)事务(transaction), 一个脚本定义了一组由若干事务组成的测试行为, 每个事务可以包含多个测试单元行为.

2)测试单元行为(behavior), 描述一次测试单元行为的具体需求, 包含负载生成器类型(type)、协议(protocol)、频率(frequency)、地址(address)等参数(params).

3)负载生成器类型(type), 其值域表示模拟仿真了哪种数据采集设备产生的流式数据负载.

4)负载协议(protocol), 其值域表示该负载所使用的数据传输协议.

5)数据发送频率(frequency), 其值域表示该负载生成器所模拟的数据发送频率.

6)地址(address), 其值域表示被测系统的访问地址.

在上述脚本模型基础上, 为了提供用户友好的脚本可视化编辑界面, 定义了面向可视化编辑的测试脚本 UI 模型, 如图 2 所示. 基于该 UI 模型, 实现了测试脚本编辑界面的自动生成与动态维护, 当脚本模型变化时, 只需更新相应的 UI 模型, 即可生成新的脚本编辑界面.

```
<!ELEMENT ui(transaction, help)*>
<!ATTLIST transaction
  name          CDATA          #REQUIRED>
<!ELEMENT help EMPTY>
<!ATTLIST help
  content  CDATA          #REQUIRED>
<!ELEMENT transaction (behavior, help)*>
<!ATTLIST transaction
  name          CDATA          #REQUIRED>
<!ELEMENT behavior (params, help)>
<!ATTLIST behavior
  name          CDATA          #REQUIRED>
<!ELEMENT params (param|group)*>
<!ELEMENT param
(radiobutton|field|checkbox|nfield|combo|dropdownbox|table|file|date
)>
<!ATTLIST param
  name       CDATA          #REQUIRED
  label      CDATA          #IMPLIED
  required CDATA          #REQUIRED>
<!ELEMENT group (param|group)*>
<!ATTLIST group
  name       CDATA          #REQUIRED>
<!ELEMENT radiobutton (choice*)>
<!ELEMENT choice EMPTY>
<!ATTLIST choice
  value      CDATA          #REQUIRED
  default   (true|false)   "false">
<!ELEMENT checkbox (choice*)>
<!ELEMENT field EMPTY>
<!ATTLIST field
  size       CDATA          #REQUIRED
  text       CDATA          "">
<!ELEMENT nfield EMPTY>
<!ELEMENT table EMPTY>
<!ATTLIST table
  cols       CDATA          #REQUIRED>
<!ELEMENT combo (choice*)>
<!ELEMENT file EMPTY>
<!ELEMENT date EMPTY>
<!ELEMENT dropdownbox EMPTY>
<!ATTLIST dropdownbox
  rows       CDATA          #REQUIRED>
```

图 2　面向可视化编辑的测试脚本 UI 模型

2.1.2 测试场景设计

性能测试场景通常用于指定测试任务的开始时间、持续时间以及负载规模等基本的测试场景要素. 在此基础上, 面向电网流式数据的负载特征, 设计增加了目标驱动的负载生成方法.

1)基于自定义负载变化函数的测试场景设计

用户通过定义多段一次线性函数进行基本测试场景设计. 对于多段负载变化函数的解析, 以一定的时间周期 T, 计算当前时间点与测试开始时间的相对距离Δt, 根据Δt, 可以利用公式(1)计算当前时刻的负载量.

$$f(t)=\begin{cases}0 & t=0\\ at+b & 0<t\le T_1\\ bt+c & T_1<t\le T_2\\ \cdots & \cdots\end{cases} \quad (1)$$

2)目标驱动的测试场景设计

目标驱动的测试场景设计是根据用户指定被测系统资源度量(如 CPU、内存、网络等的使用情况)或性能度量(如吞吐率等)的目标值, 以及安全阈值(通常是执行时间与单位时间负载增长率), 自动产生相应规模的负载以及终止条件.

本文借鉴 TCP 拥塞控制机制, 实现如图 3 所示的负载生成效果. 方法具体思路: 在测试开始阶段采用指数增长的方式快速接近用户设定的目标值, 在达到或超过目标值时, 记录当前的负载为 L_1, 然后进入负载生成的第二阶段; 在第二阶段, 将初始负载规模设置为 $L_1/2$, 并根据用户设定的负载单位时间增长量, 采用线性增加的方式再次逼近目标点; 重复上述过程, 直到获得满足用户目标要求的负载规模.

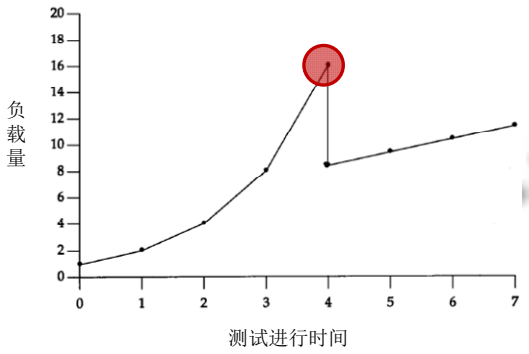

图 3 目标驱动的负载生成机制

2.2 性能测试任务管理

如图 1 所示, 一个测试任务可以分解为多个子测试任务, 由一组负载生成器并发执行. 测试任务控制模块接收包含测试脚本和测试场景的测试任务, 通过测试资源分配模块分解测试任务, 并分配每个子测试任务对应的负载生成器实例. 测试任务控制模块管理执行测试任务的负载生成器实例集群, 向每个负载生成器分发测试脚本及测试场景, 控制负载生成器集群向被测系统发送负载请求, 收集、汇总负载生成器的测试结果、资源消耗信息.

性能测试任务的生命周期如图 4 所示, 包括参数准备、初始化、分发参数、运行、采样、结束等阶段. 其中, 参数准备阶段是为了处理参数唯一性和参数化所需的步骤; 初始化阶段负责与负载发生器节点进行时钟同步验证, 保证互相之间时间同步; 分发参数阶段是将参数准备阶段处理后的参数文件分发到各个负载发生器节点; 在运行阶段, 首先同步脚本执行的内部相对时间, 然后下发执行命令; 在开始运行后, 在各个负载发生器进行测试数据的采样和统计; 测试任务结束后, 回收测试资源.

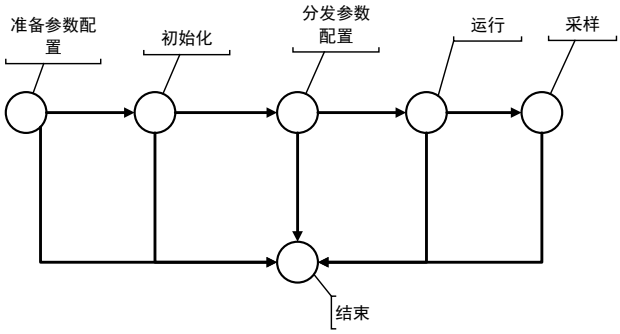

图 4 性能测试用例的生命周期

性能测试本身是资源密集型的工作, 如果测试工具未能达到测试场景中定义的负载规模, 将会误导测试人员对系统性能的评估. 例如, 在一次持续时间较长(7*24 小时)的疲劳测试中, 如果在执行过程中有部分负载发生器失效, 则从失效时间点开始一直到测试结束的测试结果均是无效的. 针对负载发生器的可用性问题, 本文采用如图 5 所示的心跳机制来探查与维护负载发生器的生存状态, 一旦在测试进行中发生负载发生器节点失效, 则可以在一个心跳周期(通常为秒级)内发现, 并启动备用负载发生器节点继续执行测试任务. 为了实现上述机制, 负载发生器被设计为无状态, 随时可以通过加载测试用例继续执行测试任务.

2.3 性能测试资源管理

2.3.1 基于 Linux 容器的测试资源池

测试平台使用 Linux 容器管理工具 Docker[4]实现虚拟化的测试资源池管理. 容器(Container)是一种新型的虚拟化资源管理技术, 具有轻量化、启动速度快等特点. 测试资源池根据用户测试任务的测试资源需

求(包括 CPU、内存、IO 等), 动态创建和回收容器. 如图 6 所示, 测试资源池架构由容器客户端(Client)和服务器(Server)两部分组成.

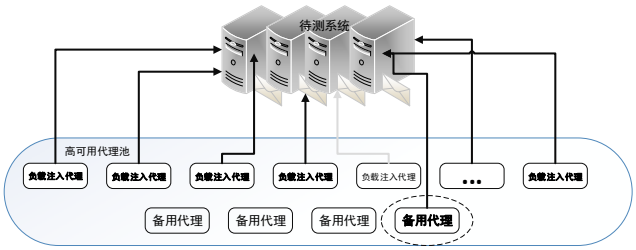

图 5　测试任务高可用保障机制

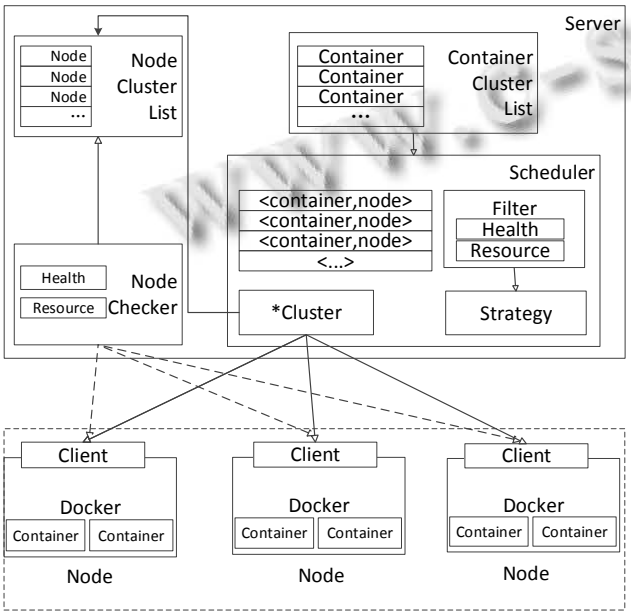

图 6　测试资源池架构

1) Client

Client 负责管理测试机资源, 通过读取系统文件, 初始化测试机资源配置信息, 并在创建、删除容器时, 维护测试机可用资源信息; 同时, 负责调用 Docker 的 Remote API, 完成容器的管理、环境部署功能.

2) Server

Server 负责维护测试资源池的测试机集群信息和容器集群信息. 其中, 测试机集群信息包括测试机节点列表及每个节点的健康状态、资源状态等信息, 由测试机节点列表(Node Cluster List)和 Node Checker(节点检查)两个模块负责维护. Node Cluster 通过读取数据库或配置文件初始化, 管理员可动态添加、删除集群中的节点. Node Checker 与所有的 Client 通信, 更新节点状态. 容器集群信息由容器列表(Container Cluster List)和调度器(Scheduler)负责维护. 当需要创建容器时, Container Cluster List 向 Scheduler 发送创建请求, Scheduler 通过调度策略(Strategy), 选择一个可用节点, 向 Client 发送创建容器信息, 更新 Node Cluster List 中节点可用资源信息, 并把容器添加到映射表(<container, node>)和 Container Cluster List 中. 当需要删除容器时, 首先通过映射表 (<container, node>)找到容器所在的节点, 然后向 Client 发送删除容器信息.

2.3.2 测试资源评估与分配

在资源需求评估方面, 大量已有相关工作为本文测试平台的测试资源需求评估提供了技术基础. 文献[5]预测应用的资源需求, 通过时间序列分析方法建立预测资源需求的模型, 预测应用系统对 CPU、内存、IO 的资源需求. 文献[6]使用排队论建立预测模型, 通过评估单位负载量的资源需求, 预测测试任务的资源需求. 文献[7]针对被测系统中的每个 API, 通过排队论评估负载生成器模拟用户访问该 API 的资源消耗. 通过该方法评估负载生成器执行一个测试脚本时的单位负载资源消耗.

本文在文献[7]方法的基础上进行改进, 针对每个测试脚本, 在测试任务开始执行之前, 首先预运行测试脚本, 通过采集资源消耗历史数据、分析样本数据, 建立资源评估模型, 计算单位负载量的资源消耗. 在一次测试中, 由于被测系统的版本和性能都相对稳定, 因此该方法在每次测试之前, 对测试资源需求建模, 可以降低被测系统对测试资源需求评估的影响.

1) 采集评估数据

在测试任务正式执行之前, 针对一个指定的测试脚本, 首先建立资源评估测试场景, 并分配用于评估的测试资源. 在测试任务预运行过程中, 通过监测器监测测试资源消耗、负载量数值对集合{(R_1, L_1), (R_2, L_2), …(R_n, L_n)}. 其中{L_1, L_2, L_3, … L_n}是不重复的数值集合, n 是样本数据集合的大小, n 越大采用统计方法分析得到的资源需求模型准确率越高. 在采集评估数据时, 资源评估测试场景的负载量由公式(2)获得:

$$L(t)=\frac{L_{max}}{t_e}\times t \tag{2}$$

其中, t_e 表示评估阶段的执行时间, L_{max} 表示评估阶段的最大测试负载量. 为避免测试机和被测系统出现瓶

颈而导致样本数据异常需要限定该阶段的最大测试负载量 L_{max}. 若测试任务的最大负载量为 L_1, 根据经验值评估的测试资源最多可以运行的负载量为 L_2, 则 L_{max}=min(L_1, L_2).

2) 建立评估模型

本文假设测试资源需求与测试负载量成线性关系, 采用线性回归方法预估资源需求, 建立如公式(3)所示的回归方程:

$$R = a + bL \quad (L > 0) \tag{3}$$

其中 R 是资源需求, L 是测试负载量, a 是负载生成器等运行时所需的资源, b 代表单位负载量的资源需求. a、b 是需要通过线性回归学习的参数, 本文使用最小二乘法进行计算, 如公式(4)和公式(5)所示:

$$a = \frac{\sum r}{n} - b\frac{\sum l}{n} \tag{4}$$

$$b = \frac{n\sum rl - \sum r\sum l}{n\sum l^2 - \left(\sum l\right)^2} \tag{5}$$

其中, n 表示样本数量. 由于需要评估所需测试资源的上限值, 采用置信区间预测方法, 计算置信区间的上限值如公式(6)所示:

$$r = r_0 + t_{\frac{\alpha}{2}} u_r \left(l > 0\right) \tag{6}$$

其中, r_0 是在自变量为 l_0 时因变量的值, α为置信区间, u_r为抽样平均误差.

3 平台应用效果

测试平台在某电力调度系统的流式数据处理系统方案选型评测中得到实际应用. 被测系统分别搭建了基于 1 款关系型数据库(RDBMS)以及 2 款非关系型数据库(HBase、MongoDB)的数据管理系统测试环境, 测试目的是针对上述 3 种方案进行对比评测. 图 7-8 分别给出各方案在不同负载规模测试场景下的写入性能对比和写入失败情况对比. 测试结果表明, 与传统关系型数据库相比, 在高并发的数据写入负载条件下, HBase、MongoDB 通过水平扩展可以显著提升系统的数据处理能力, 降低写入失败率.

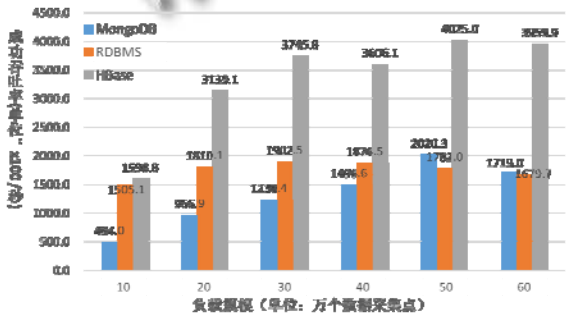

图 7 不同方案的写入性能对比

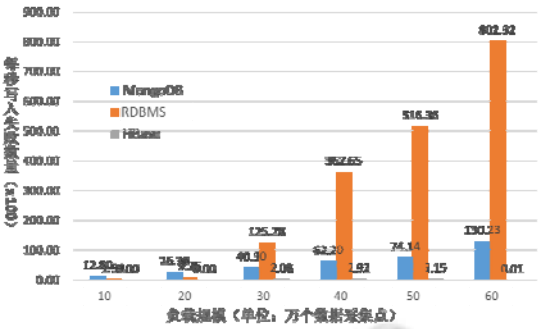

图 8 不同方案的写入失败情况对比

4 结语

本文对面向电网流式数据处理的性能测试平台进行充分研究, 给出了架构设计及关键技术, 为电网流式数据处理系统的软硬件架构、配置的方案验证提供了平台和技术基础.

参考文献

1 Yu YS, Yang J, Chen B. The smart grids in China-A review. Energies, 2012, 5: 1321–1338.

2 丁杰,奚后玮,韩海韵,周爱华.面向智能电网的数据密集型云存储策略.电力系统自动化,2012,36(12):66–70.

3 白红伟,马志伟,宋亚奇.基于云计算的智能电网状态监测数据的处理.东北电力,2011,39(9):1487–1488.

4 Docker. https://www.docker.com. [2015].

5 Jiang J, Lu J, Zhang GQ, Long GD. Optimal cloud resource auto-scaling for web applications. Proc. of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2013). Delft, The Netherlands. 2013. 58–65.

6 Urgaonkar B, Shenoy P, Chandra A, et al. Agile, dynamic provisioning of multi-tier internet applications. ACM Trans. On Autonomous and Adaptive Systems, 2004, 3(1): 217–228.

7 Zhou J, Li S, Zhang Z, et al. Position paper: Cloud-based performance testing: issues and challenges. Proc. of the 2013 International Workshop on Hot Topics in Cloud Services. ACM. 2013. 55–62.