# 开源监控系统的选择 - 简书

http://www.jianshu.com/p/5bdeb896af08

2017.02.20 18:44 字数 6128 阅读 13评论 0喜欢 1

开源的或者商业的监控系统很多，具体可以参考维基百科的条目Comparison\_of\_network\_monitoring\_systems。一个完整的监控系统，往简单了讲，主要包括三个主要部分：数据采集、根据策略告警、数据图表展示。我们围绕这三个方面，简单地对现有的几个有代表性的产品做一些介绍和分析。

## Cacti

Cacti，最悠久的监控系统之一，2001年9月，一个名叫Lan Berry的高中生，当时他还在为一家小的ISP厂商工作，为了更好地监控网络质量，开发了Cacti的第一个版本，基于RRDtool，提供更友好的使用体验。

Cacti的数据采集，采用的是Server端poll的方式，在默认安装情况下，会有一个PHP的Poller，来定期执行相关的数据采集脚本，或者直接采集SNMP的接口输出，采集到的数据会以RRDtool的格式存储到磁盘上，供后续绘图、展示。在要采集的数据较多的情况下，PHP版本的Poller效率较低。为了解决这个问题，Cacti官方提供了一个组件Spine（早期叫Cactid），这是一个用C语言编写的多线程程序，用来代替PHP版本的Poller，有效地提升了数据采集效率。用户可以在Cacti提供的设备管理页，对设备列表进行增、删、改操作，来控制Poller要拉取的任务。

Cacti的数据图表展示，底层是基于RRDtool的，Cacti提供了一个用PHP开发的页面，可以让用户很方便地查看每个数据采集项的历史趋势图。Cacti提供了一个树状的graph列表，方便用户高效地组织、管理多个graph。另外，Cacti的graph预览功能很强大，允许在一个页面中放置大量的graph预览图，便于用户了解其所关注的指标全貌。

Cacti之所以很强大，原因在于其强大的插件。比如aggregate插件可以对多个采集项进行聚合展示，discovery插件用来自动发现设备，thold插件用来配置告警策略，slowlog插件可以方便地用来分析MySQL的慢查询等。完整的插件列表可以在[http://docs.Cacti.net/plugins](http://docs.cacti.net/plugins) 查看，包括官方支持的和社区支持的。

总之，Cacti很强大，用的人很多，社区支持也好，可以作为快速上手的一个不错的选择。

## RRDtool

RRDtool，在时间序列数据（time-series data）的存储、展示方面，其独创的round-robin database数据存储格式，基本上是事实上的工业标准了。包括Cacti、MRTG、Collectd、Ganglia、Zenoss等系统，都是采用RRDtool的格式来存储数据，以及使用RRDtool的Graph工具来绘图的。

RRDtool使用的数据存储格式，大家也常常称之为环状数据库，其工作方式有三个显著的特点：第一，RRD文件在创建的时候，其文件大小就确定下来了，随着数据的不断写入，RRD文件的大小一直保持不变；第二，数据每次更新到RRD文件的时候，都会触发RRD文件中的归档策略，也就是数据采样策略；第三，查询历史数据的时候，会自动选择最优化的采样数据，而不是全量获取数据，查询效率很高。

RRDtool包含了一组工具集，用于创建RRD文件，更新RRD文件，获取RRD文件中的数据，根据RRD文件直接生成相应的图片等，具体可以参考<http://oss.oetiker.ch/rrdtool>。

我们在使用RRDtool的过程中遇到过一些问题，RRDtool的数据是以文件的形式存储在磁盘上，以单机的形式来提供服务的，这样就存在容量上限。该上限的决定因素较多，比如磁盘容量、磁盘IO、CPU等，但是最核心的制约因素就是磁盘IO，用户每push一次数据，都会转化为对相应的RRD文件的一些全量的读/写，磁盘IO会最先遇到瓶颈。在一台普通的Linux服务器上，在1分钟push数据的频率下，一般20万条的Counter上报就会跑满磁盘IO。这显然无法满足较大规模数据下的监控需求。

为了在一定程度上缓解磁盘IO压力的问题，RRDtool官方提供了一个组件rrdcached，这是一个常驻内存的后台程序，用户可以把读/写请求通过网络发送给rrdcached，而不是直接操作磁盘。rrdcached内部做了一些优化措施来减轻对磁盘的读/写压力，包括：缓存RRD文件的header部分，每次数据push上来的时候可以减少一次读取操作；对RRD文件的写入，提供了用户态的缓存，即把用户的多次写入操作合并成一次flush到磁盘上，这样有效地提高了写入效率。通过该项优化，使得单机的容量提升不少。

不过上述优化，也只能解决一定程度上的问题，整体容量仍然局限于单机的容量上限。

## Collectd

Collectd相比Cacti、RRDtool来说，较为年轻一些，项目最早是在2005年由Florian Forster开发的，之后便蓬勃发展成为一个开源的项目，很多开发者对其做了大量的改进和扩展。Collectd的定位是收集和传输数据。在告警方面不是Collectd的设计初衷，不过它也支持一些简单的阈值判定，并发送告警信息。要支持更高级的一些告警需求，Collectd可以和Nagios配合使用，有一个名为collectd-nagios 的插件可以很方便地完成这个功能。

Collectd是一个用C语言开发的常驻内存的程序，由一堆功能强大的插件组成，其架构示意图如图1所示。

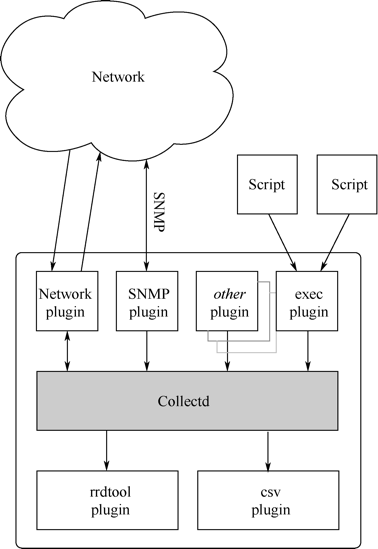


图1 Collectd架构示意图

插件化是Collectd最重要的一个设计思想，Collectd的所有功能都是通过插件来支持的，Collectd自身没有任何额外的依赖，这使得它几乎可以跑在大多数的操作系统上，包括一些嵌入式系统如OpenWrt。从图1来看，用户可以通过各种插件push数据到Collectd，然后通过RRDtool插件存储为RRD的格式，或者通过CSV插件存储为CSV的格式。Collectd支持的上百个插件，可以在 <https://collectd.org/wiki/index.php/Table_of_Plugins>中查阅。

## Nagios

前面讲到的几个系统，都专注于数据的采集、传输、聚合、存储和展示。说到告警，Nagios可谓是事实上的工业标准，可以用来监控主机和网络基础设施，以及各种应用服务。在监控对象出现问题时，及时发送邮件或者短信通知相关人员；当问题解决后，发送恢复信息。

Nagios 从结构上来说，可以分为核心和插件两个部分。Nagios 的核心部分只提供了很少的监控功能，因此要搭建一个完善的监控管理系统，用户还需要在Nagios服务器上安装相应的插件，插件可以从Nagios官方网站[http://www.nagios.org](http://www.nagios.org/) 下载，也可以根据实际要求自己编写所需的插件。

Nagios可以监控各种网络服务，比如SMTP、POP3、HTTP、NTP、ICMP、FTP、SSH等，也可以监控主机资源，比如CPU、Load、磁盘使用、Syslog等。基本工作模式如图2所示。

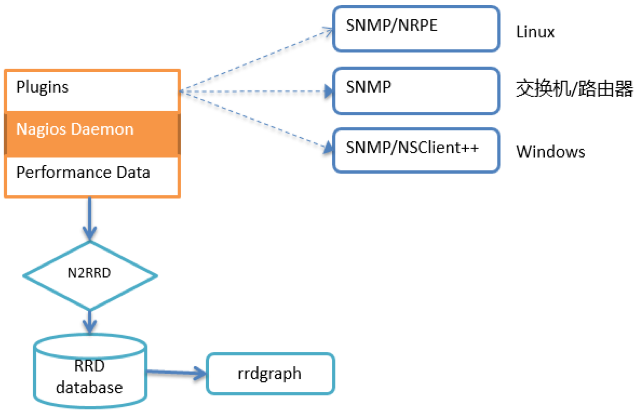


图2 Nagios的基本工作模式

这里介绍两个比较重要的概念：NRPE和SNMP。

NRPE的全称是Nagios Remote Plugin Executor，是Nagios的Agent，这可以让Nagios具备监控远程主机和设备的能力。Nagios服务端，通过check\_nrpe插件会定期地调用运行在远程主机上的NRPE，执行具体的脚本来获取数据，比如check\_load、check\_disk、check\_ftp等。

SNMP（Simple Network Management Protocol，简单的网络管理协议）是一种应用层协议，被路由器、交换机、服务器、工作站、打印机等网络设备广泛支持，主要用于管理和监控网络设备。SNMP的工作方式主要有三种：管理员需要向设备获取数据，SNMP提供了“读”操作；管理员需要向设备执行设置操作，SNMP提供了“写”操作；设备需要在重要状况改变的时候，向管理员通报事件的发生，SNMP提供了“Trap”操作。

SNMP的基本思想是：为不同种类的设备、不同厂家生产的设备、不同型号的设备，定义一个统一的接口和协议，使得管理员可以使用统一的方式对这些需要管理的网络设备进行管理。通过网络，管理员可以管理位于不同物理空间的设备，从而大大提高了网络管理的效率，简化了网络管理员的工作。Nagios很好地利用了SNMP的读和Trap功能，很容易地获取各种网络设备的运行数据，达到监控的目的。

## Zabbix的使用经验和优化

前面介绍了一些常见的、传统的监控系统，我们在初期选型的时候，也都有考虑过，不过最后还是选择了Zabbix。Zabbix作为一款企业级分布式监控系统，功能齐全，用户体验良好，文档完善，API强大，适合于中小规模的公司或者团队使用。

Zabbix的主要特点有：

* 文档齐全，安装、维护、学习成本低。
* 多语言支持。
* 完全免费开源。当然，如果需要技术支持的话，Zabbix官方是收费的。
* 自动发现服务器和网络设备，便于用户配置。
* 支持告警策略模板的概念，方便用户对一批服务器的监控策略进行管理。
* 用户体验良好的Web端，用户可以进行集中配置、维护和管理。
* Zabbix Agent功能强大，默认的采集项足够丰富，且支持用户自定义插件扩展。
* 不用Agent配合，也可以完成监控任务，支持SNMP、JMX等标准的协议，用户也可以自行推送数据到监控系统中。
* Web端也提供了良好的Dashboard功能、绘图查看功能。
* 可以配置历史数据的存储周期，历史数据支持采样存储。
* 支持分布式监控，比如多个IDC之间，或者有防火墙阻隔。
* 强大、完善的API支持。

在以上特点中，尤其是API功能，完善程度很高，基本上Zabbix的大部分操作都提供了相应的API接口，方便用户编程，和现有的一些系统进行整合。比如以下一些场景。

* 利用历史数据查询API，定期从Zabbix中获取线上服务器的各项资源使用情况，生成每日检查报表；同时，对某些资源使用率不达标的服务器和业务进行筛选，每周进行通报，有效地促进资源利用率的提高。
* 利用Zabbix graph的API，可以对关注的指标获取对应的趋势图，嵌入到各个运维系统中，方便运维人员快速地了解服务情况。
* 利用Zabbix的告警添加API，可以让监控系统和部署系统联动起来。比如某个模块增加了一个实例，那么可以自动添加所需要的监控策略；反之，下线一个实例，可以自动删除关联的监控策略。

Zabbix主要由Server、Agent、Proxy和Web-portal几个部分组成。典型的Zabbix的部署模式如图3所示。

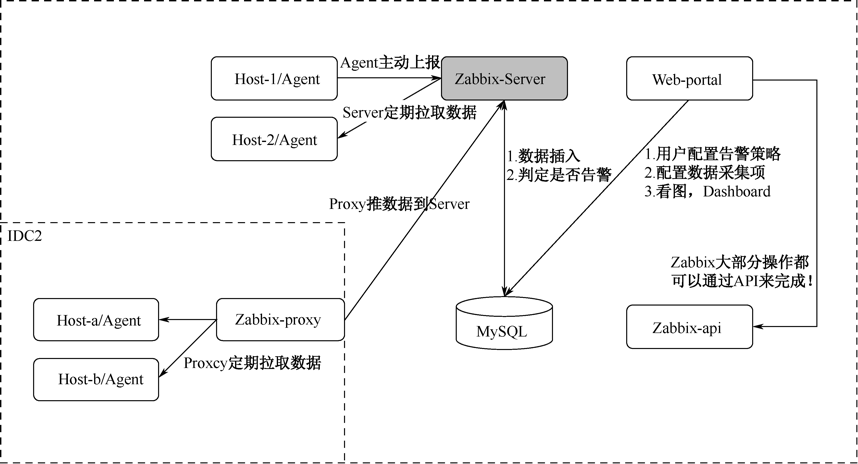


图3 典型的Zabbix的部署模式

Zabbix的数据采集，主要有两种模式：Server主动拉取数据和Agent主动上报数据。以前者为例，用户在Web-portal中，配置好机器，并给机器应用相应的模板后，Zabbix-server就会定期地去获取Agent的数据，存储到MySQL中，同时根据用户配置的策略，判定是否需要告警。用户可以在Web端，以图表的形式，查看各种指标的历史趋势。

在Zabbix中，将Server主动拉取数据的方式称之为active check。这种方式配置起来较为方便，但是会对Zabbix-server的性能存在影响，所以在生产环境中，一般会选择主动推送数据到Zabbix-server的方式，称之为trapper。即用户可以定时生成数据，再按照Zabbix定义的数据格式，批量发送给Zabbix-server，这样可以大大提高Server的处理能力。

Proxy是Zabbix具备分布式监控能力的一个必备条件，试想我们有一批服务器和网络设备位于防火墙之后，Zabbix-server无法直接访问这些Agent，这时候我们可以选择在防火墙的后面放置一个Zabbix-proxy，那么Proxy就会充当Server的角色，定期收集它所负责的这些Agent的数据，然后定期推送回Zabbix-server。另外，Proxy还可以分担Server的压力，代替Server定期拉取数据，再统一push给Server，这样可以有效地降低Server的开销。

在Zabbix的设计中，以下几个概念是最重要的。

* Host：主机，是Zabbix里面的监控主体，可以是服务器，也可以是网络设备，通过DNS或者IP地址来连接。
* Item：Host的某个数据采集项，比如hostA的cpu\_idle就是一个Item。
* Template：这是Zabbix的一个很先进的概念，是对具有相同属性和资源的Host的一种抽象。即与某个Template关联的Host，会自动具备该Template所具有的Item、Trigger、Graph等属性。同时Template具备继承的能力。
* Trigger：触发器，具有三种状态，即unknown、problem和ok。只有当状态从problem变为ok的时候，或者ok变为problem的时候，才会触发相关的Action。当Zabbix-server每次取到Item的值时，与这个Item相关的Trigger都会被检查一次，并生成相应的Event。
* Action：顾名思义，就是执行动作，Zabbix的Action支持多种动作的执行，用户可以配置满足什么样的条件，做什么样的动作，动作包括发短信、发邮件、执行脚本等。
* Event：当Trigger的状态每发生一次变化时，就会产生一个Event。

Zabbix在业务处于较小规模的时候，效果还是相当不错的。但是当监控的对象超过上千台设备，并且还包括一些服务自身的业务指标也推送到Zabbix的时候，我们遇到了两个严重的问题——Zabbix的性能问题和用户的“使用效率”低下问题。

Zabbix的性能问题主要存在两个方面，一是Zabbix-server处理能力有限，尤其当active check模式的采集项较多的时候，会显著消耗Server的Puller线程，使得数据采集延迟，产生堆积，造成报警延迟。我们可以调大Puller的线程数，缓解这个问题，但Zabbix-server自身无法水平扩展，所以不能解决根本问题；二是Zabbix的数据存储引擎存在性能瓶颈，我们线上采用的是MySQL，当数据采集项过多的时候，比如在每分钟大概有20万采集项的规模下，MySQL的写入会达到瓶颈。

综上所述，在业务规模较小的前提下，Zabbix是一个很可靠的开源解决方案。在业务规模不断增长的情况下，我们基于Zabbix做了一些优化尝试，在这里分享给各位读者供参考。

### 测试Zabbix Node

Node是Zabbix官方针对分布式监控扩展需求提出的解决方案。可以通过建立多个Zabbix Node来组成一个层级网络（如图4所示），每个Node都可以看作是一个全功能的Server，它负责监控自己所管理服务器的指标，并把历史数据和告警事件同步到自己的主节点。

用户可以选择在子节点上来配置策略，也可以集中在主节点上配置好策略，那么策略会被同步到所有的子节点。通过建立节点，可以带来以下好处。

* 监控能力可以大大提高。
* 当子节点和它的主节点网络连通性出现问题时，数据会被暂存在子节点本地，待连接恢复后，子节点会把暂存的数据再次发送给主节点。  
  在图4中，所有的节点，其数据和告警事件最终都会被汇聚到节点4上去。

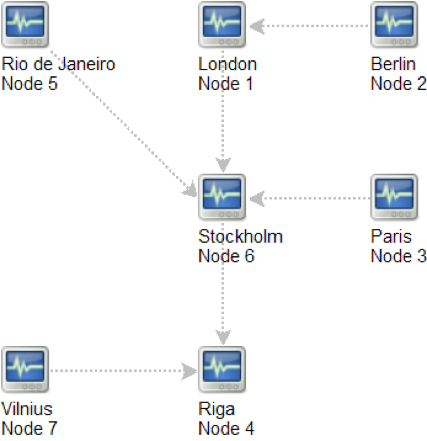


图4 Zabbix Node组成的层级网络

但是遗憾的是，这种Node模式，一方面不够成熟，稳定性较差；另一方面受限于Zabbix当前已有的设计局限，无法很好地在生产环境中使用，且从Zabbix-2.4版本起，这种Node模式已经被官方弃用了。所以，我们对Node模式的尝试，失败了。

### Zabbix代码优化和使用模式优化

通过分析，Zabbix的性能瓶颈主要体现在两块，一是数据库更新压力大；二是Zabbix的Puller线程忙。针对这两个问题，我们采用了以下解决方案来提高性能。

* 尽量不使用active check模式，而改用trapper模式，主动推送数据给Zabbix-server，来解决Puller线程忙造成的数据采集延迟问题。
* 每次数据更新时，都会更新数据库items表，而items表存在外键约束，导致更新速度太慢，拖累了Zabbix的整体性能。因此，我们通过修改代码，不再依赖于MySQL自身的外键约束来提高整体性能。
* MySQL所在的服务器采用SSD硬盘来提高数据库的性能。  
  通过这些优化手段，可以充分地提高Zabbix的单机处理能力，但是仍然无法满足业务的快速增长需求，问题仍然在继续。

### 独立部署多套Zabbix，通过API进行封装整合

如果业务需求是成倍地在增长，那么单机版的监控系统无论如何优化，总是存在容量上限。我们迫切需要寻找一个可以水平扩展的方案来支撑业务的快速发展。  
前面已经讲过了，Zabbix的API功能非常强大，基本上日常用到的所有操作都有相关的API可以调用。因此，我们采用了一个较为稳妥的方案，即部署多套Zabbix，每套Zabbix只单独负责一个业务部门的监控需求。然后通过API，把对多套Zabbix的操作整合到一个Web前端系统中，尽可能地满足运维人员、研发人员的监控配置需求、图表查看需求。

这是我们在发展初期最终采用的方案，基本上可以满足业务需求。

## OpenTSDB

OpenTSDB是目前最优秀的时间序列数据（time-series data）存储和展示的分布式解决方案之一，遵守LGPL开源协议。OpenTSDB具有以下特点。

* 数据存储：基于HBase，并且存储的都是用户上报的原始数据，不会对数据进行采样和钝化处理，历史数据可以一直保存。支持毫秒级别的数据上报频率。
* 可扩展性：由于后端的数据存储引擎是HBase，因此可以轻易地支撑每秒上百万次的数据更新操作，并且处理能力随着HBase节点数量的增加而增加。
* 数据查询：OpenTSDB提供了友好的用户访问界面，方便用户查看相关的数据趋势。另外，也提供了HTTP方式的数据获取接口，方便用户通过编程，自动化地获取HBase中的历史数据，用作其他用途。

不仅可以查询单个指标的历史数据，OpenTSDB还提供了强大的聚合功能，比如可以查看多个指标求和后的数据。

OpenTSDB的部署结构和工作流程如图5所示。

HBase为数据存储引擎，TSD是OpenTSDB最核心的组件，和HBase的所有数据交互都通过TSD来完成。TSD是一个常驻内存的进程，是无状态的，可以水平扩展。  
我们可以通过tcollector主页是<https://github.com/OpenTSDB/tcollector>来收集每台服务器的各个指标，然后推送给TSD；也可以通过SNMP来获取网络设备的各项指标，推送给TSD。TSD收到数据后，会更新到后端的HBase中。

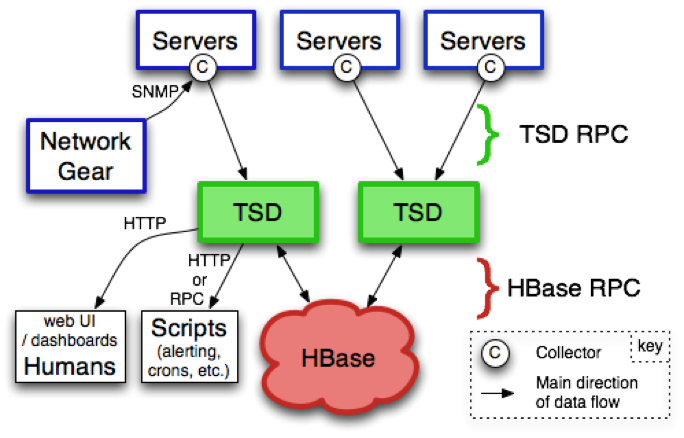


图5 OpenTSDB的部署结构和工作流程图

OpenTSDB提供了Web界面，通过HTTP的接口向TSD查询数据；我们也可以编写一些插件，比如告警插件，从TSD中获取某个指标的数据来判定是否满足阈值，以及是否需要告警。

OpenTSDB的出现，让时间序列数据的存储和展示多了一个很好的选择。对于大量写入的场景非常有用。不过也存在一些不足的地方，比如历史数据的查询速度较慢：由于OpenTSDB存储的是原始数据，没有做任何采样，因此在需要查询某几个指标在过去一个月甚至一年的历史数据的时候，TSD真的就会去HBase中扫描相应时段的所有数据。首先，数据量很大；其次，读操作并不是HBase最擅长的；最后，费了好大力气获取到大量的数据，也无法很好地展现给用户，仍然需要应用程序对数据做采样，造成无谓的一些消耗和浪费。

当然近几年也有一些非常优秀的开源监控解决方案诞生，influxdb 和 Prometheus都是其中的佼佼者，可以参考。