大米運維課堂

最前沿開源監控 Prometheus專題講座

第十三讲：Prometheus 企业级实际使用(一)

第十三讲内容：

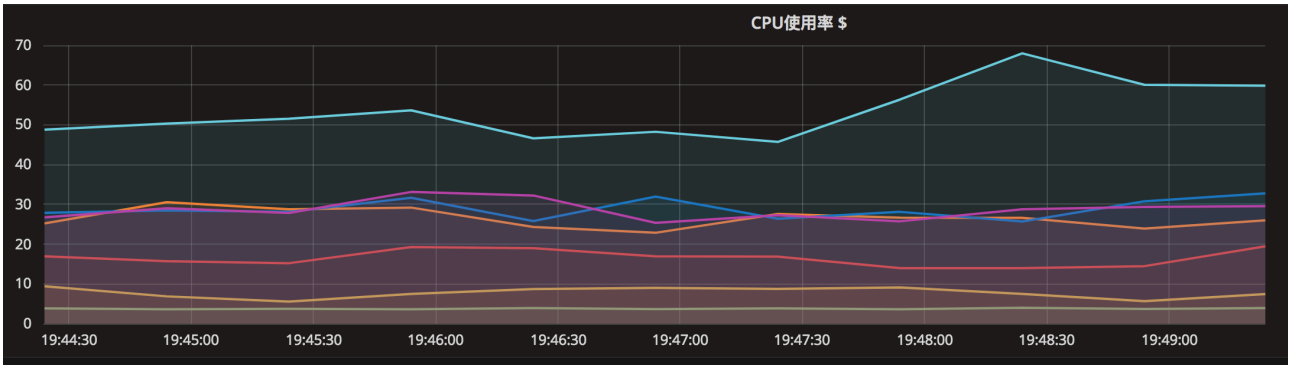
- prometheus+grafana　企业CPU监控 真实案例
- prometheus+grafana　企业内存监控 真实案例
- prometheus+grafana　企业硬盘/IO监控 真实案例
- prometheus+grafana　企业网络传输 真实案例

（一）prometheus+grafana　企业CPU监控 真实案例

我们在企业中 基础监控的第一项 一般就是 针对服务器集群的CPU 进行监控
为什么呢?
因为：1） CPU是处理所有任务的核心（这是废话）
　　2） 另外 Linux 由于CPU存在各种 状态类型CPU时间
所以 很多情况下 大部分的出现问题的情况
都可以 反应在CPU的表现上

下面举一个 在企业中对CPU使用率监控的实例

数据采集：　Node_exporter



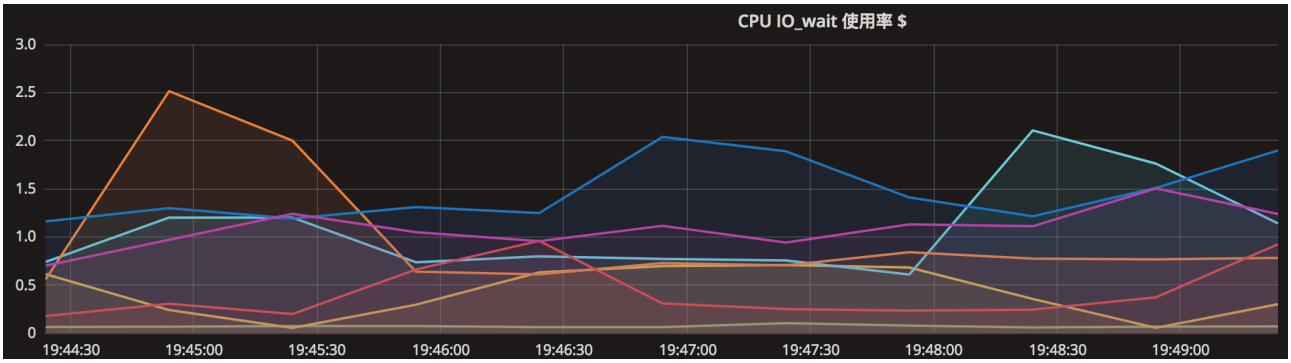
使用prometheus公式
(1-((sum(increase(node_cpu{mode="idle"}[1m])) by (instance)) / (sum(increase(node_cpu[1m])) by (instance)))) * 100

第一幅图 就是咱们之前 讲过的，计算CPU综合使用率 这里就不再重复它的计算方法了 （可以回顾 下篇的 6 7讲）
在生产环境中 一般70-80%以上的CPU高 是因为 用户态**user** CPU高所导致
我们使用**Top**命令随便查看一台服务器的时候 一般也会看到user%会最高

```
Cpu(s): 12.7%us,  1.5%sy,  0.0%ni, 84.8%id,  0.5%wa,  0.0%hi,  0.5%si,  0.0%st
```

用户态的CPU使用率 是跟应用程序（或者说软件）的运行密切相关的
当软件启动大量进程 并行处理任务时， 当进程之间频繁上下业切换时 对用户态的CPU 消耗最大

不过我们 在做监控的时候 一般倒是不用 单独列出一个 user%态的CPU使用率图 因为 除去IO等待造成的CPU高之外，大部分情况 就是 user%造成
另外， system% 内核态的CPU 使用率 偶尔也会出现高的情况，我们这个课程中 就不再讲解了



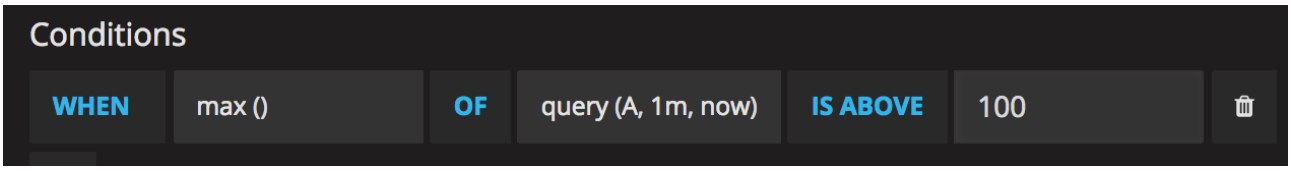
使用prometheus公式

(sum(increase(node_cpu{mode="**iowait**"}[1m])) by (instance) / sum(increase(node_cpu[1m])) by (instance)) * 100

第二个图 是针对 IOWAIT类型的 CPU等待时间 user%
其中不同的地方 是mode=iowait

很多情况下， 当服务器 硬盘IO占用过大时，CPU会等待IO的返回 进入 interuptable 类型的CPU等待时间
所以 对于 IOWAIT CPU的监控 是很有必要的

grafana
另外 对于CPU高的报警阈值
是这样的设置的



设置成 99 或者 100 都可以
如果设置成 80 90 就报警，根据实际测试 并不合适，因为 80% 90%状态下的服务器 还是可以处理请求的
只不过速度会慢了

但是 一旦综合CPU上了 98 99 100 那么整个服务器 就几乎失去可用性了 连SSH登录 有时候都很困难
所以 针对Linux系统的优化 非常重要 要通过各种内核参数 软件参数 来控制服务器 尽量不让CPU堆到 99 100 （更多Linux内核优化知识 请关注大米运维 第三 四 阶段）

（二） prometheus+grafana 企业内存监控 真实案例

接下来 就到了 内存监控了
首先 大米需要给大家 说一下 内存的计算方式

我们先从Linux命令来看起

free -m

Centos 6.x 5.x

```
[root@log ~]# free -m
             total       used       free     shared    buffers     cached
Mem:         15950      12743       3207          0       5705       1303
-/+ buffers/cache:       5735      10215
Swap:            0          0          0
```

Centos 5/6 的版本中 如上图显示内存（4.x就不说了 基本绝迹了）

内存管理 是Linux内核的 非常重要的一个强势功能
可以说 Linux对于内存的使用率 非常的高校 比起windows来说 真的智能了很多

主要依赖于 Linux内存管理的 缓存功能 （简单来说就是 刚用过的内存中的内容 会被暂时缓存一段时间 以备下次再使用 快速调用）

然而 5.x 6.x 的 内存命令 却有一点 不太善解人意
对于 大多数的零基础和初级学员来说，命令行显示的这个

```
 used        free
12743        3207
```

很容易让人误解（甚至我遇到 很多工作5年以上的人 居然还有搞不清这里的）

关于Linux的内存种类 和内存管理细节 我们这里就不过多阐述了

直接给出大家 5.x 6.x 的真实内存使用率公式即可

从应用程序的角度来说，Linux 实际**可用内存=系统free memory+buffers+cached。**

使用率 = 实际可用内存 / 总内存

Centos 7.x

```
[root@prometheus ~]# free -m
              total        used        free      shared  buff/cache   available
Mem:          64263       10543        4750           0       48970       53092
Swap:             0           0           0
```

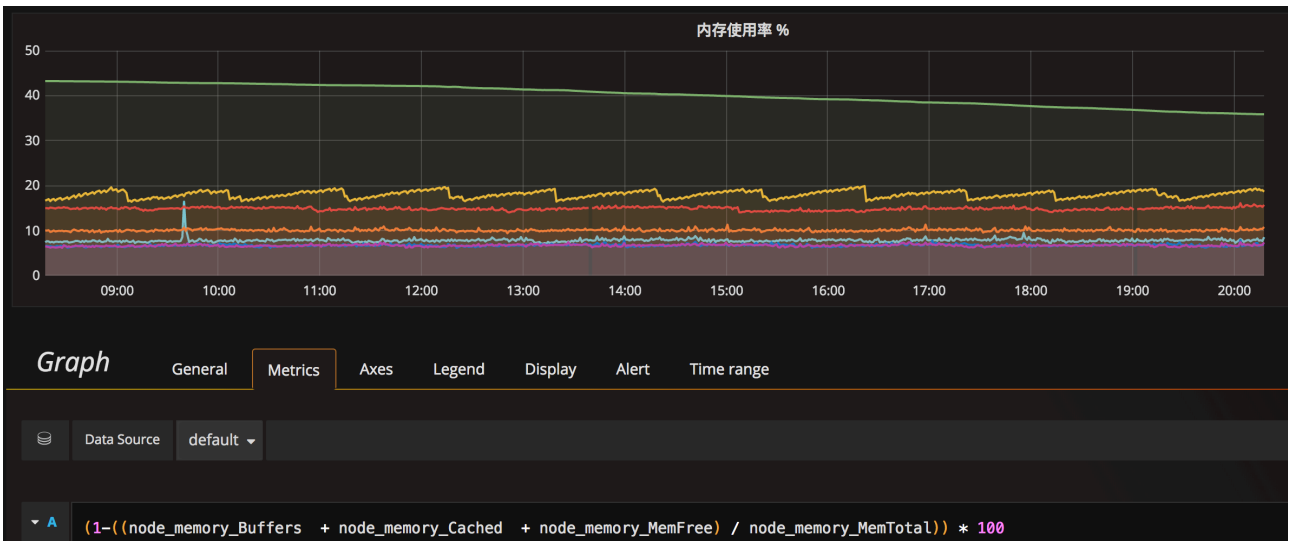
对于 最新的 7.x中
free 命令行的输出 解决了这个问题

变得简单易懂

实际可用内存 直接放在最后一列 直接使用

```
available
    53092
```

接下来我们来看 企业实际内存监控案例



给出大家prometheus 公式

(1-((node_memory_Buffers + node_memory_Cached + node_memory_MemFree) / node_memory_MemTotal)) * 100

(6.x 7.x free , 监控 6 7)

这里就很清楚了 跟上面讲的是对应的算法

从图上 可以看出 大部分机器的内存使用率 比较低 基本都在20%以下

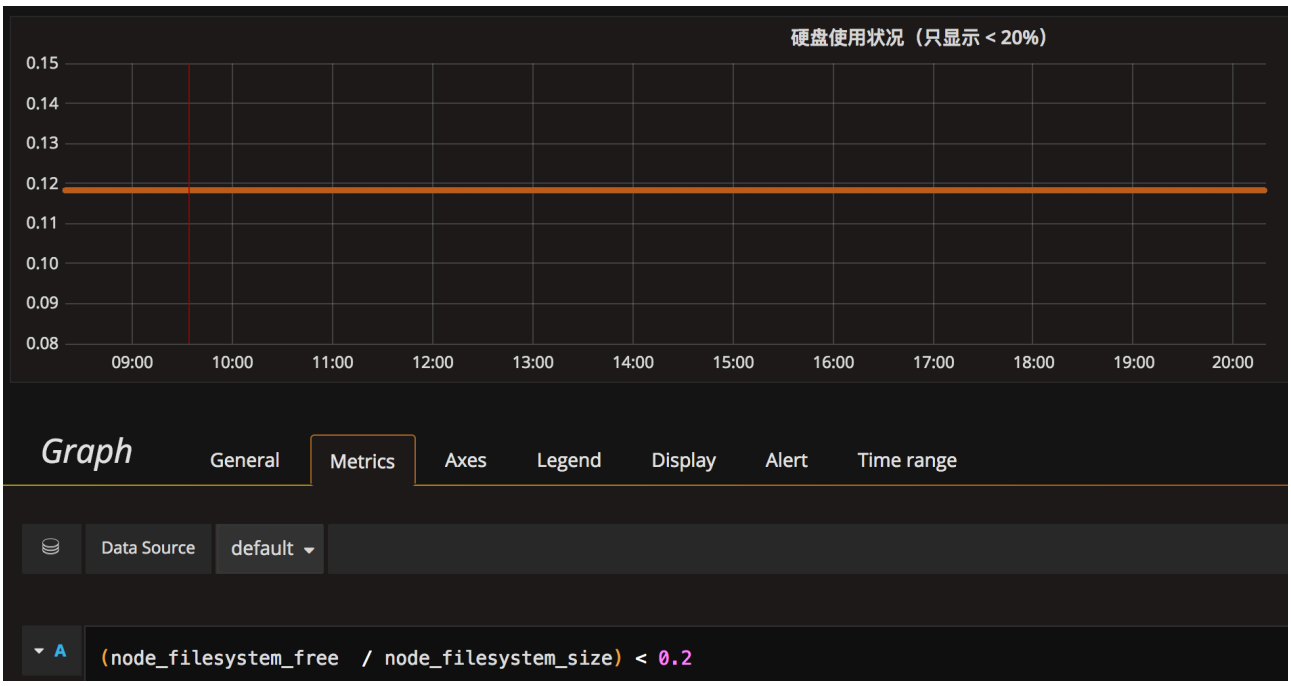
是因为 之前的生产环境使用中 并不是内存密集型的 （是CPU密集型）

所以说 我们从内存的计算公式来说， promehtues也让我们很精细 很放心， 很多老式的监控 直接返回一个 内存使用率 很多时候 无法确认准确性

- prometheus+grafana 企业硬盘/IO监控 真实案例

数据来源 : Node_exporter

硬盘剩余容量的监控 相比上面的2个 就简单很多



(node_filesystem_free / node_filesystem_size) < 0.2(nagios)

公式上看 也很好理解 当空闲硬盘 小于 0.2的时候 就显示在图上

另外 提到了 硬盘使用率
我在这里 给大家推荐另一个 难度较高的 prometheus 函数

`predict_linear()`

`predict_linear(v range-vector, t scalar)` predicts the value of time series `t` seconds from now, based on the range vector `v`, using simple linear regression.

`predict_linear` should only be used with gauges.

关于这个函数 我们后面会讲解
在这里 我们先提出一个 对于硬盘监控的比较新的理念

对于硬盘使用率来说
通常不管使用 什么样子的监控工具 基本上 都是简单算法 空闲/总量 或 以使用/总量 当大于或小于 一个阈值时 报警

这么定义的方法 比较简单也普遍

那么我给大家提一个问题

假如有这么一种情况

有一天晚上8:00的时候 一台很重要的服务器上 硬盘使用率超过了 90%
超过了报警阈值 于是乎 报警电话就响了

结果运维人员上线一看 哦 还有百分10的空间呢 根据以往的经验 10%的空间基本上 还能撑个 3,4天呢
况且 我们的服务器 每天24:00还会做 日志压缩呢 会释放很多空间
所以 不着急处理 就回去睡觉了😪

又有一天晚上 10:00的时候 还是这台服务器 又报警了 >90%
这次运维人员 还是老样子 觉得没啥大事 就去睡觉了 报警也给屏蔽了（电话是很吵人的）

结果 到了10:10分的时候 整个生产线上报警了 老板也给吵起来了

运维慌慌张张打开电脑一看.... 我靠！！

硬盘100% 导致nginx服务无法响应了.....

怎搞的啊 平时不会这样子啊
后来经过分析 是由于 晚上 9:00的时候 服务器系统上的APP软件 出现了一个死循环的错误
超大量的 info/warning日志 把整个 根目录都给盛满了 而且用的时间 很短

运维哭了😢

运维这个时候想，既然会有这种特殊情况 那么干脆把 报警阈值 设置的更高一些吧
70%就报警好了 ... 结果 可想而知 往后的日子里 运维都是黑圆圈了。。。

给大家讲这个真实发生过的 运维故事（嘿嘿 😁 其实那个运维 就是5年前的我）
就是想告诉大家 prometheus 针对这种特殊的问题 提供了一个 非常牛X的函数

```
predict_linear()
```

这个函数 如果想讲清楚它的底层实现原理 没个 2 3天还真说不完

我们在这里就给大家简单介绍一下它能做什么吧

对于刚才那种 硬盘百分比报警的案例(剩余空间的百分比)

predict_linear() 函数 可以起到 对曲线变化速率的计算 以及在一段时间 加速度的未来预测

说的更简单一些

它可以 实时监测 硬盘使用率曲线的 变化情况，假如在一个很小的时间段中 发现硬盘使用率 急速的下降（跟之前平缓时期相比较）

那么对这种下降的速度 进行一个未来一段时间的预测， 如果发现 未来 比如5分钟内 按照这个速度 硬盘肯定就100%了
那么 在当前硬盘还剩余 20%的时候 就会报警！

说起来都觉得绕口 不过使用起来 并不是很难
官网有 predict_linear（）的 使用方法介绍
https://prometheus.io/docs/prometheus/latest/querying/functions/#predict_linear()

另外 也有针对这个 函数的 基础理论 Linear 数学计算模型 的介绍
https://en.wikipedia.org/wiki/Simple_linear_regression

In other words, $\hat{\alpha}$ and $\hat{\beta}$ solve the following minimization problem:

$$\text{Find } \min_{a,\,b} Q(a,b), \quad \text{for } Q(a,b) = \sum_{i=1}^{n} \hat{\varepsilon}_i^{\,2} = \sum_{i=1}^{n} (y_i - a - bx_i)^2 \ .$$

By expanding to get a quadratic expression in a and b, we can derive valu

$$\begin{aligned}
\hat{\alpha} &= \bar{y} - \hat{\beta}\,\bar{x}, \\
\hat{\beta} &= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \\
&= \frac{\operatorname{Cov}(x,y)}{\operatorname{Var}(x)} \\
&= r_{xy}\frac{s_y}{s_x}.
\end{aligned}$$

Here we have introduced

看这个数学计算公式 …..😳

总之 promehtues就是这样一个 有各种生活大爆炸型的数学家（也可能是物理学家哦） 通过他们的辛勤努力 给我们提供了非常繁多的 这种复杂计算的实用函数
为的都是 把监控做的越来越好

然后 我们来看下 硬盘IO使用的 监控

使用的公式

((rate(node_disk_bytes_read[1m])+
rate(node_disk_bytes_written[1m])) / 1024 /1024) > 0

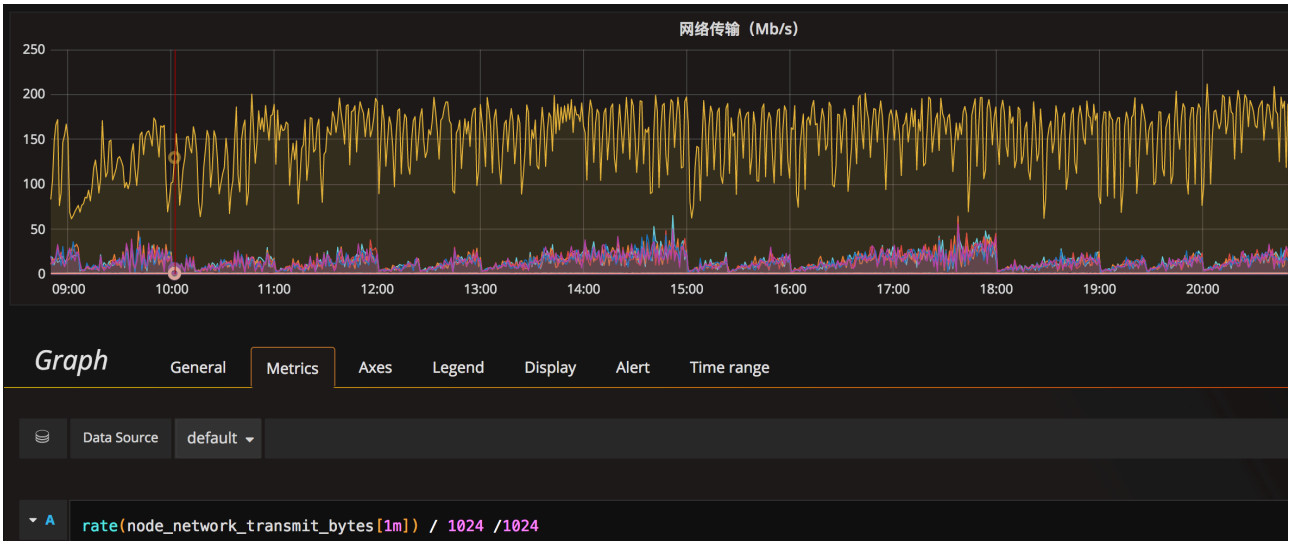
硬盘使用率 是 read + written 读和写 都会占用IO

/1024 两次后 就由 bytes => Mbs

如果这个指标标高了， 那么必然 CPU_IOWAIT 也会飙高

（所以说 从这里 我们也可以看得出 报警中 很多项目 虽然重要 但是无法避免重复 都有一定的连带关系 这也是大米为什么在上篇中 给大家提出了一个 真实链路报警的 未来展望）

- prometheus+grafana 企业网络传输 真实案例



使用公式 rate(node_network_transimit_btyes[1m]) /1024 /1024

最后这个 咱们 之前在 7 8讲的时候 用过它举过例子

这里就不再重复了 公式大家可以直接使用

休息一下吧 等待进入后面的课程