**5、业务智能运维典型应用场景和关键设计**

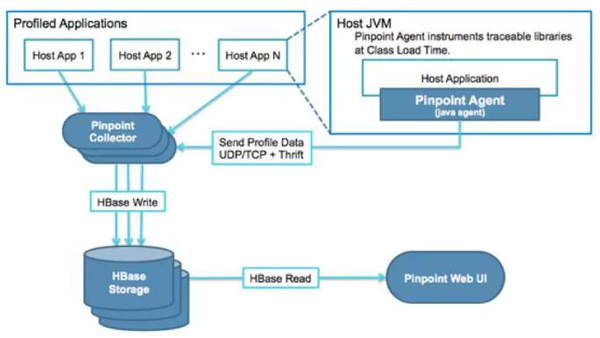
5.1 数据的采集

**（1） 业务层数据的采集**

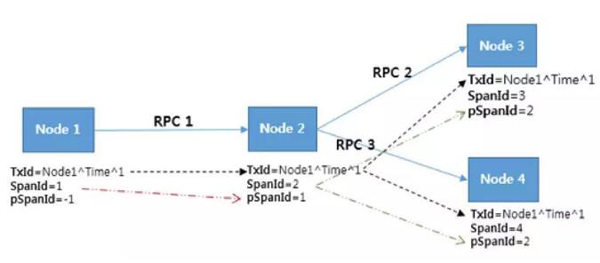
包括接口响应时间、调用次数、服务间调用关系、时延、慢SQL、JVM内存消耗、以及线程栈信息，上述数据的采集可以参考Google Dappe的思想实现，其中一款较好的开源软件就是pinpoint。

pinpoint运用JavaAgent字节码增强技术实现应用服务端数据的采集，且无侵入式的设计，使用方便，无需更改业务代码。可以内置支持JAVA程序内几十种协议交互的兼容，如http、okhttp、mysql、oracle、postgresql、dbcp、cubrid、kafka、rabbitmq、springboot、log4j、logback、redis等。

**Pinpoint的架构原理图如下：**



采用hbase 实现海量数据的存储，通过部署在业务远端的agent通过UDP+thrift的方式将应用采集的数据传输到collector，经过处理后实现hbase的落存。Web UI实现监控的可视化。



上图是通过pinpoint进行链路追踪的原理图，可以简单的理解为在一次交易过程中，贯穿的整个分布式系统的各个环节内都维持着一个唯一的transactionid，且允许记录上下文环节的spanid，从而实现链路信息的洞悉。

且pinpoint允许开发者自定义开发插件，实现更多协议的监控支持，如activemq、zookeeper、consul等。

不过pinpoint的功能如此强大的同时，还需要我们做适当的优化，如：

Agent发送海量的udp数据到collector，很有可能遇到网络和collector的阻塞，那么，这个时候可以在agent和collector之间加一层kafka实现消息的缓冲，提升系统稳定性。

Pinpoint没有用户权限体系，需要我们自己实现。

可以通过参数自定义的方式来指定实际需要采集的指标项，避免agent多余的性能损耗，降低系统负载。

**（2） 关联数据的采集**

关联数据包括基设施数据和中间件数据。

首先，基础设施数据如服务器的性能状态的数据，包括CPU、磁盘、内存、IO、负载等维度各个参数的获取，您可能首先想到的是zabbix，那么zabbix确实功能强大，但是“杀鸡焉用牛刀”，上述CPU/磁盘/内存等几个参数就是我们随手敲行代码就可以搞定的事，只不过做成定时任务即可，所以我们不用zabbix，转向轻量化的开源手段，其实TICK数据采集框架您可能都听过，那么我们模仿TICK，通过TIG（Telegraf+influxdb+grafana）框架也可以轻松搞定了。

Telegraf是一种轻量级的采集框架，支持秒级别间歇的采集粒度，对服务器的资源占用很小（不到3%）；

Influxdb是一种高性能的时序数据存储引擎，可支持百亿级别的时序数据的存储，而且内置强大的连续计算、API功能，你可以轻松的实现数据的汇聚和外部调用；

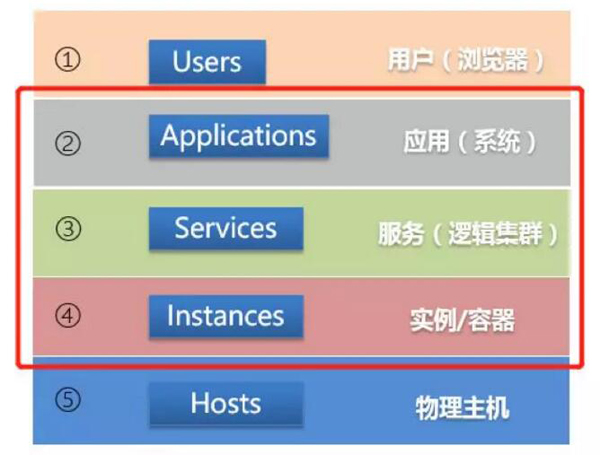
Grafana是一款基于JS的前端可视化引擎，支持丰富的dashboard组件，如图表、仪表盘、表格、清单等，你可以利用它轻松实现各种高大上的性能监控页面，另外grafana和influxdb的兼容性也异常的友好。

同样，常见的Paas和Daas层中间件，如nginx、apache、zookeeper、docker、mesos、ZFS、Elasticsearch、mysql、mongodb、postgresql、sql server、rethinkDB、influxdb、couchDB、redis、memcache、rabbitmq等的组件的监控也都可以通过TIG框架实现监控。

至此，我们已经可以实现应用层数据和其关联数据（Iaas层、Paas层）的集中采集和汇聚，那么有了数据，能做的事情简直太多了。

5.2 业务层面的精细划分

**欲建立面向业务服务维度的监控体系，首先需要针对业务服务做出分层次的划分，即对业务监控对象的管理需要建立体系，智能运维产品的业务服务管理体系结构如下：**



如上图中的②③④层，专注面向业务维度的监控的同时，更要对业务层面进行精细化分层，比较容易想到的办法就是建立系统、服务、实例三层的业务监控体系。

**针对系统、服务、实例做一个概念的普及：**

**系统：**完成某一类完整需求的系统体系，如OA系统，系统是一个比较抽象的概念，一般由一个或多个运维人员来管理

**服务：**系统的下一层模块，即完成系统内某一个完整的相对独立功能的模块，如个人信息管理服务、薪资管理服务、流程引擎服务等；一个服务一般部署为一个集群，包含多个应用实例（如tomcat）

**实例：**属于一个服务集群中的一个具体的应用实例，一般一个服务集群会部署多个实例到不同主机上，如薪资管理服务实例一、薪资管理服务实例二，实现负载均衡。

在这三个层次上进行性能的监控，实现了业务应用从上到下三层的数据关联，服务运维人员可以更深入的掌控系统业务的关联状况。

那么我们是否可以针对系统、服务、实例分别进行性能监控呢？如果发生故障，就可以寻根溯源，举例：如果一个服务层的指标（如服务整体平均响应时间发生偏高的异常），那么必定是由其下的一个或多个实例导致，现在我们去查看每个实例的性能信息，通过皮尔曼相关系数，发现性能曲线和服务性能曲线最近的实例，就是异常实例，进而可以针对该实例的Top N请求进行下钻分析就可以得到故障所对应的代码行，问题就可以解决了。

上面所建立的系统服务实例的关系，本身就是利用了业务应用运行时本身就存在的关系，那么为何不利用起来呢，到这里还没用到高大上的AI、机器学习呢。

5.3 故障可视化与故障重现

**故障可视化**

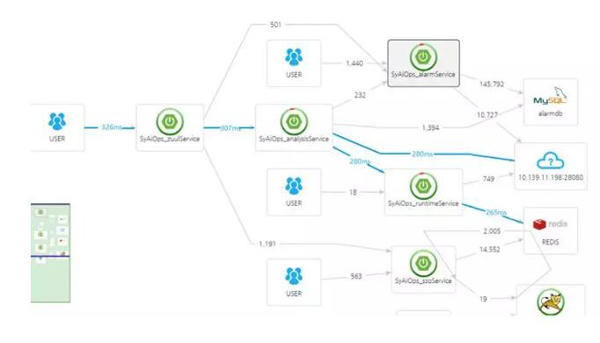
当发生故障时，可以在指标的运行图谱高亮显示该异常点，也是可视化工作中必须的，正如如下图：



上图内，系统识别到了“响应时间”的异常，当前时间点的异常指标为11ms，同时一个友好的智能运维系统会把该时刻系统其他方面的指标也展示出来，运维人员可以直观的看到不同曲线之间的关系，并且图中每一个坐标图的右上角都展示了该指标与异常指标之间的“相关系数”，并且按相关系数绝对值倒序排列，相关系数绝对值越接近于1，那么就越有可能是问题的直接或间接原因。

**故障重现**

另外当业务系统的一次请求发生了错误，如果我们可以提供手段将该次请求的过程进行一次重现，对于运维人员的排错支持也“将是极好的”。



如上图所示，可以对一次应用的请求进行回放，每一个环境执行了多长时间都可以一目了然。

5.4 异常检测

**说到异常检测，应该是业务智能运维领域中的一个最常见的场景了，异常检测的方法很多，本篇中会重点的介绍一下我的见解：**

**（1） 传统的异常检测方法**

传统模式下完全基于人的主观经验，也即基于固定阈值的异常判断，如 CPU usage高于80%就告警，这种方式适配性是很差的，需要针对不同的场景设定不同的阈值，甚至同一个业务不同时间段的阈值都是不一样的，大量个性化的配置要求，对于运维人员来说是十分崩溃的。

后来就出现了一定的改进，如3-sigma算法，是根据正态分布的概率，自动的调整告警阈值，是的，告警阈值的配置不用人工进行，一定程度上提高了运维效率。但是，该类的算法机器容易忽略指标的周期性和趋势性，造成误判的问题也很常见了。

**（2） 基于统计学和机器学习的异常检测方法**

总结前面的异常检测方法，可以概括为两点：人工运维工作量大、算法适配性低下。其实归结为一句话，就是动态阈值怎么评定的问题。

这个时候就比较适合引入机器学习了，比如基于指数的三次平滑算法、基于分解的傅里叶/小波分解算法等，可以有效的识别出指标的周期性、趋势性，可以快速识别出一些尖峰（spike）异常。

另外自回归移动平均模型（ARIMA算法），对于稳定的时序数据的异常检测是非常有效的，该算法也非常适合用作时序数据的预测场景。

还有基于深度学习的循环神经网络 RNN算法和长短期记忆网络LSTM算法，比较适合处理和预测时间序列中间隔和延迟相对较长的重要事件。

基于机器学习的众多算法，都可以大大的提高运维的效率，发现人工难以发现的问题，提高预警的及时性。

**（3） 异常检测模型优化**

上一小节提到的各类机器学习算法，虽然都功能强大，但是往往都有一定的局限性，那么我们在对具体的一个场景指标（如响应时间）做异常检测的时候，我们到底选哪个算法呢？

方法一：这个问题可以通过“自动模型选取”方式来解决，即采用多个算法同时运行，然后通过投票的方式抉择产生最终的结果。

举个例子，针对“响应时间”指标进行异常检测，采用同比、环比、ARIMA、LSTM、KNN、高斯共5个算法同时进行异常检测，当其中的一半（即>=3）的算法判定为异常时，方认为该时刻的指标是异常的。

方法二：在方法一的基础上为每个算法加入权重值，5种算法初始值均为20（总合为100），当一次异常的判断后，比如算法1/2/3都判定是异常，算法4/5都判定为非异常，那么最终结果为判定为异常，系统向运维人员发出告警，当运维人员在平台上通过指标横向对比、请求下钻、事件挖掘之后发现该时刻的指标确实为异常，那么运维人员会将这个告警处理掉，那么此时后台就会默认向投票正确的算法的权重倾斜，为其权重加1，同时为投票错误的算法权重扣分（但总分仍保持100分）；而如果运维人员发现该告警是误报，则会在平台上反馈“误报”，则后台会向投票为非异常的算法权重倾斜，为每个算法权重加1，同时为投票为异常的算法权重扣分（但总分仍保持100分）。如此经过长时间的不断调整，算法组合就越来越接近于准确。

**（4）答疑解惑**

不过有朋友可能会遇到如下问题：

**Q：如果我要检测的指标刚刚上线，我根本就没有离线的训练模型怎么办？**

A：那就初始阶段不利用离线模型的算法，先使用ARIMA、同比、环比、KNN这类的算法跑起来，等待历史数据足够了生成离线模型之后，再以同等权重（取得和现有算法权重的平均值，再进行100分支均衡）的方式加入到算法集合中。

**Q：我使用这么多的算法来进行异常检测，对于前端告警规则配置的时候来说，我该怎么去选择我去使用哪种智能的算法呢？**

A：异常检测的目的就是要识别异常并发出告警，那么在告警规则出进行配置，选择智能化的方法来检测异常的思路是正确的，但是没有必要让普通的运维人员来看到我们所提供的众多算法，还有算法逻辑，对于他们来说我们只需要让他们选择诸如“智能告警”这样的选项就好了，后面的算法选择交给专业的“运维算法工程师”来搞定就好。

**Q：有了“智能告警”之后，是不是固定阈值告警就不需要了呢？**

A：并不是，智能告警解决的是无法直观、简单判定故障的场景，但是对于错误率、CPU利用率、磁盘剩余量这些基本场景时，还是可以使用阈值告警的，甚至做分级阈值告警（如一般告警、重要告警、严重告警等），这些基本的阈值告警发生后一般都是比较严重的情况，都是需要处理的；而且，这些告警信息汇聚起来，也可以作为业务层面异常故障定位的参考依据，因为很有可能这些固定阈值触发的告警就是业务层面故障发生的根因。

**（5）算法训练和模型管理平台**

好了，长篇大论了半天，我们似乎还忽视了一个关键的问题，那就是离线训练的模型是怎么来的，怎么用起来，怎么选算法，怎么调优，算法一定好用吗？

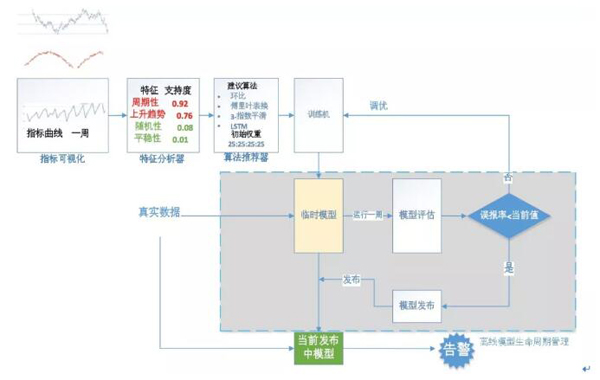
带着这一系列的问题，我们可以想象的到，一个离线算法训练和模型管理平台是十分必要的，这就是“运维算法工程师”所需要使用的平台了，这个平台至少要实现如下功能：

算法如何选择

算法的好坏可以评估

算法最好经过测试后才可上线

**离线算法训练管理平台的设计可以参考如下模型：**



离线算法训练管理平台架构简图

该平台可以获取需要检测的指标，展示过去一段（如一周或一天）时间的曲线；

特征分析器会根据预设的特征组合（事先定义好针对曲线可能的各种特征的识别判定方法库），提示出该指标的曲线对于各类特征（如上升趋势、周期性、随机性等）的支持度，支持度越高代表着该指标越具有什么特征；

然后算法推荐器会根据预设的特征-算法组合（事先定义好各种特征所适用何种算法的映射库），推荐出建议的算法集合（可1可多），当然也允许“运维算法工程师”在查看了第一步的曲线后，自定义选择算法库。

下一步就将通过前面算法推荐器推荐的算法或运维算法工程师自定义的算法组合进行模型的训练，将生成的临时模型保存起来；

然后，采用真实的线上数据来跑这个临时模型，会得到对应的告警；

当运行一段时间（如一周或一天）后，将临时模型发出的告警和线上模型产生的告警进行对比，去掉重复的部分，剩余部分通过运维工程师的标注和反馈，得到两个模型的误报率（当然也可以采用漏报率），若临时模型的误报率低于线上模型，则认为模型是有效的，可以进行发布环节，该临时模型替换线上模型，进入生产。

注：临时模型和线上模型的对比如果无法通过运维工程师的评估快速得到的情况下，也可以采用比较通用的算法评估方法来计算得出，不过最好的手段就是“利用运维工程师的判断”。

5.5 关联分析

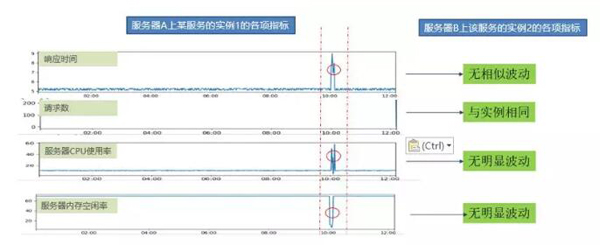
**关联分析一般会作用在故障定位和告警归集两个差劲**

**（1） 故障定位**

**基于关联关系的基础可视化辅助**

在针对系统的异常进行有效的检测后，极大的缩小了故障的范围，如将故障缩小到了某几分钟内，然后将相关的其他指标曲线和故障曲线同时可视化展示，则可以辅助我们深入数据进行问题的定位：

理论依据：当某一个维度的指标发生异常时，那么相关的其他指标也极有可能一定程度上体现出正向或反向的波动，如果可以将多个疑似相关指标的曲线在一个图上展示，并提供格线比对功能，那么相比于传统的翻阅日志看log的情况，将会更快的定位到问题的原因。



落地场景：如上图所示，某服务器上某服务实例在10:00左右发生了响应时间严重变慢的情况，经过对相同服务器的各项指标分析，可知当时系统CPU占用在同一时刻上升，且内存的空闲率也大幅下降，但是实际的业务访问量并没有飙升，说明并非业务繁忙导致，疑似服务器硬件问题所致；同时在针对部署在服务器B上的相同实例的指标进行对比，发现各项指标并无明显波动，且和服务器B上正常指标类似，所以可以确定是因为服务器A的硬件问题导致，完成故障初步定界，继而再去排查服务器的相关指标，便可迅速定位问题。

**基于多维度数据的异常诊断分析**

理论依据：通过贡献度和一致度评判问题根源（如ERROR数量维度）

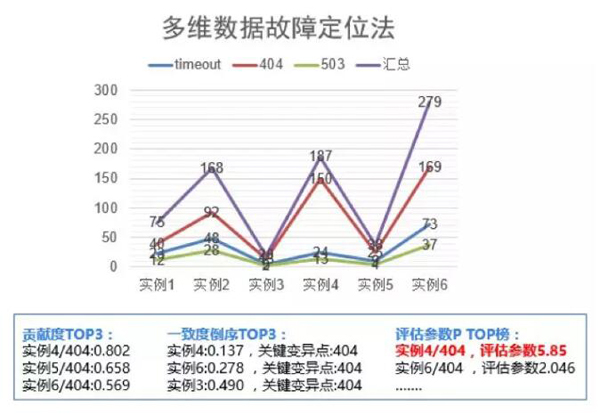
贡献度：即各维度异常数与总异常数的比例

一致度：即构成该维度的子维度的异常程度的相似度信息。

那么贡献度越高、子维度的异常相似度越高，则该维度为根因维度的可能性越大。

因此，可以将数据的各维度展开，分别计算各维度的贡献度、一致度两个特征，构建评估参数P=贡献度/一致度，该值越高，则该子维度为根因维度的可能性越大。

落地场景：当发现某服务（如充值服务）的错误率告警突然大幅增加时，传统运维人员往往无法快速定位，甚至问题的定界都需要大量的时间，如果运用智能运维产品，可以将该服务的所有6个实例上进行3个错误共6\*3=18中维度上进行分析，利用上述理论中的评估参数列出排名前N的组合，迅速将问题范围大幅缩小，提高排查的效率。



**可以定位到实例4的404错误是错误数的主要原因，可针对性进行排查**

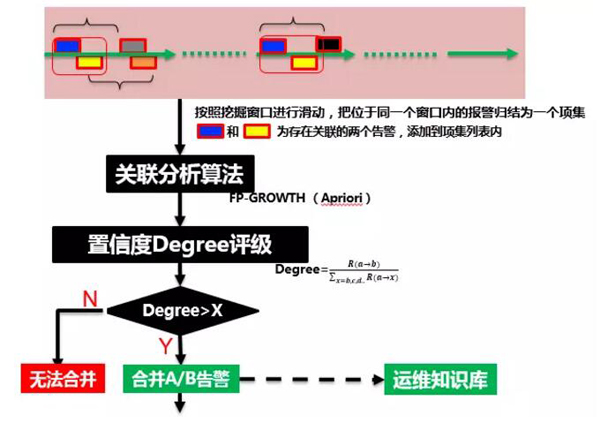
**（2） 告警归集**

**基于关联挖掘的告警分析**

采用机器学习算法实现告警的关联挖掘，进而实现告警前的合并优化，与告警后的数据分析，反哺合并策略。

理论依据：历史上每次某一个告警总是伴随着另外一个告警的出现，那么可以怀疑两类告警之间存在必然联系，甚至因果关系，所以可以考虑合并该两类告警，并积累在运维知识库内，随着历史数据的丰富，告警合并的准确性将不断提高。

落地场景：在历史数据上，A实例的策略R1和B实例的策略R2经常同时报警，那么A实例的策略R1和B实例的策略R2就极有可能存在关联，经过一定的置信评级，就可以合并在一起发送。



注：置信度是针对一条关联规则A告警->B告警而言定义的，代表了A告警导致B告警发生的可能的概率

**智能告警体系下充分利用从业务到主机的纵向数据关联，实现告警的聚合与收敛**