

# 杭州电子科技大学

## 创新实践综合结课报告



题    目：\_\_\_\_\_ 基于根因分析的报警算法落地 \_\_\_\_\_

课程时间：\_\_\_\_\_ 周四下午第三、四节 \_\_\_\_\_

成    员：\_\_\_\_\_ 洪晨晖、石力源、韩世容、盛琼怡、毛忆宁 \_\_\_\_\_

指导老师：\_\_\_\_\_ 吴以凡 \_\_\_\_\_

## 目录

<b>前期调研 .....</b>	<b>4</b>
<b>    聚类算法 .....</b>	<b>4</b>
聚类算法的定义 .....	4
三种常用聚类算法 .....	4
<b>    根因分析 .....</b>	<b>5</b>
根因分析的定义 .....	5
根因分析的步骤 .....	5
根因分析的目的 .....	6
<b>环境准备 .....</b>	<b>6</b>
<b>    Open-Falcon .....</b>	<b>6</b>
Open-Falcon 介绍 .....	6
Open-Falcon 安装 .....	7
Open-Falcon 架构 .....	8
Open-Falcon 源码解读 .....	9
<b>功能实现 .....</b>	<b>12</b>
<b>    中间件设计 .....</b>	<b>12</b>
<b>    报错获取 .....</b>	<b>13</b>
<b>    报警聚类 .....</b>	<b>13</b>
聚类测试 .....	13

<b>问题与解决措施</b> .....	<b>15</b>
<b>数据库被攻击</b> .....	<b>15</b>
解决措施.....	15
<b>小组分工</b> .....	<b>16</b>

# 前期调研

## 聚类算法

### 聚类算法的定义

[聚类分析](#)又称群分析，它是研究（样品或指标）分类问题的一种统计分析方法，同时也是数据挖掘的一个重要算法。

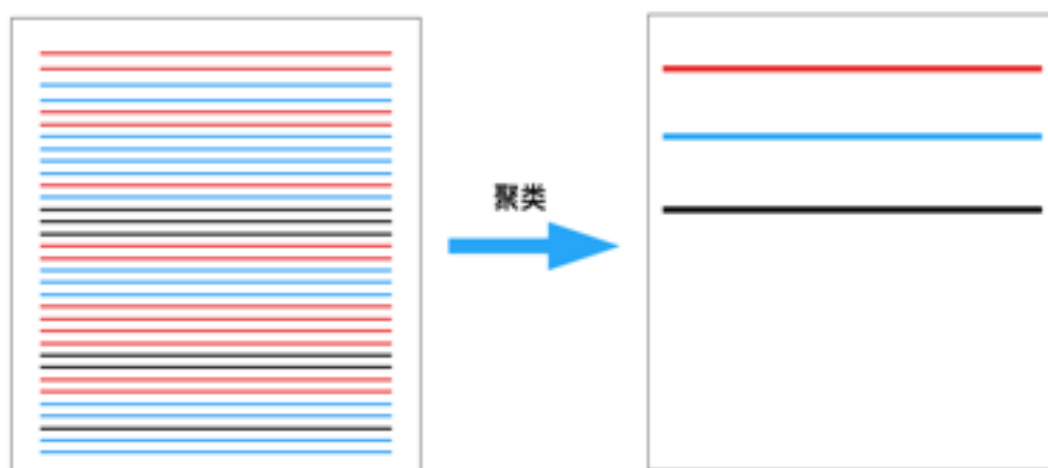


图 1-1 聚类算法

## 三种常用聚类算法

### K-means

原理简单，易于实现，可解释度较强，适用于大型数据集，高效。但 K 值比较难确定，容易出现局部最优的情况，对噪音和异常点也比较敏感。

### 层次聚类

距离定义简单，可以不预先设定类别数，能够发现类别间的层次关系，以及可生成非球形簇。但它计算量大；对异常值敏感；很可能聚类成链状。

## 密度聚类

对噪声不敏感，能发现任意形状的簇。聚类结果没有偏倚，相对的，K-Means 之  
类的聚类算法初始值对聚类结果有很大影响。但数据密度不均匀时，很难使用该算法。

## 根因分析

### 根因分析的定义

通过一系列流程找到问题的根本原因，定位问题所在的一种算法通过根因分析，  
将相同根因的报警合并。

### 根因分析的步骤

1. 输入报警数据集合
2. 对数据集进行关键特征的提取
3. 泛化&聚类&合并相同的报警信息
4. 将不需要继续聚类的结果输出-报警摘要

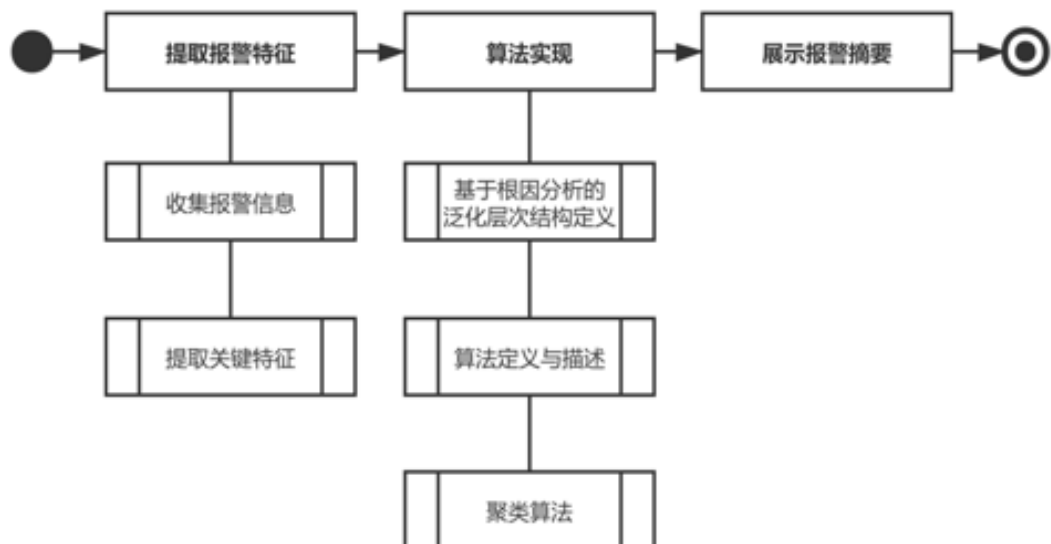


图 1-2 根因分析的步骤

# 根因分析的目的

- 减少报警信息
- 快速定位错误

# 环境准备

# Open-Falcon

# Open-Falcon 介绍

open-falcon 是一款用 golang 和 python 写的监控系统，由小米启动这个项目。可以从运营级别（基本配置即可），以及应用级别（二次开发，通过端口进行日志上报），对服务器、操作系统、中间件、应用进行全面的监控，及报警，对我们的系统正常运行的作用非常重要。

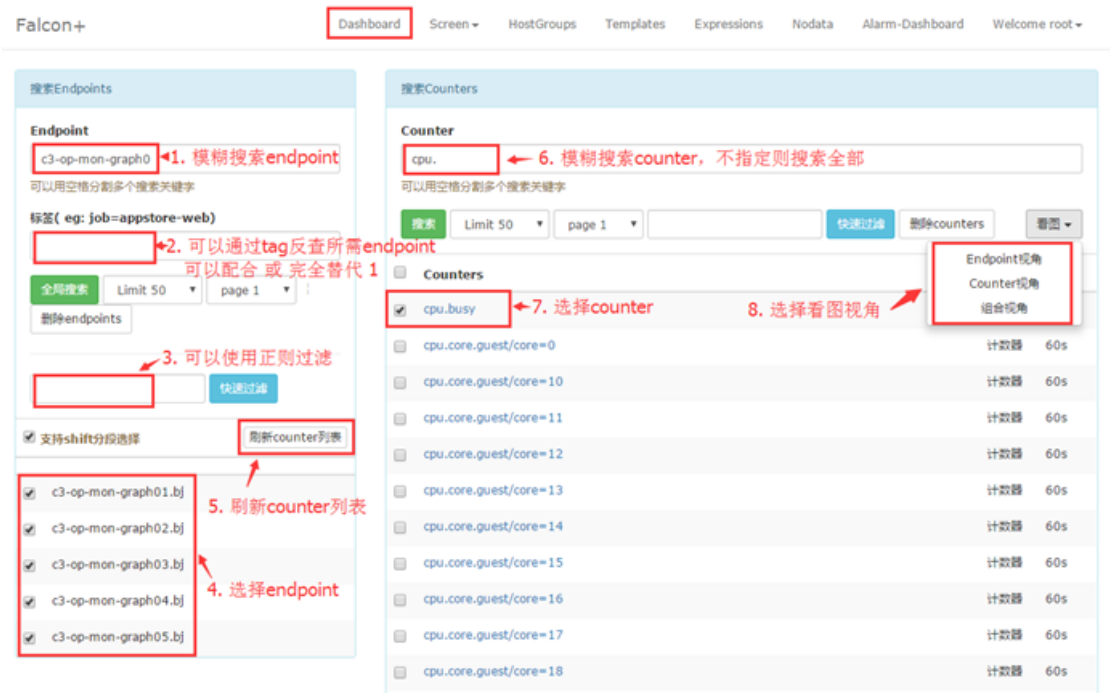


图 2-1 Open-Falcon 前端界面及解读<sup>(1)</sup>



图 2-2 Open-Falcon 前端界面及解读(2)

## Open-Falcon 安装

### 前端

1. 创建工作目录
2. 克隆前端组件代码
3. 安装依赖包
4. 初始化数据库
5. 修改配置以开发者模式启动
6. 在生产环境启动查看日志
7. dashboard 用户管理

### 后端

1. 安装 redis
2. 安装 mysql
3. 初始化 MySQL 结构
4. 从源码编译

## Open-Falcon 架构

### ● agent

1. 需要监控的服务器都要安装 falcon-agent
2. agent 提供了一个 http 接口/v1/push 用于接收用户手工 push 的一些数据，然后通过长连接迅速转发给 Transfer。
3. 部署好 agent 后，能自动获取到系统的基础监控指标，并上报给 transfer，agent 与 transfer 建立了 TCP 长连接，每隔 60 秒发送一次数据到 transfer。

### ● transfer

1. transfer 进程负责分发从 agent 上送的监控指标数据，并根据哈希分片。
2. 将数据分发给 judge 进程和 graph 进程，供告警判定和绘图。

### ● judge

1. Judge 从 Heartbeat server 获取所有的报警策略，并判断 transfer 推送的指标数据是否触发告警。
2. 若触发了告警，judge 将会产生告警事件，这些告警事件会写入 Redis（使用 Redis 消息队列）。
3. redis 中告警事件，供处理告警事件的 Alarm 进程转发告警消息，或是 Email，或是手机短信等。

### ● alarm

1. Alarm 进程监听 Redis 中的消息队列，并将 judge 产生的告警事件转发给微信、短信和邮件三种 REST 接口，REST 接口才是具体的发送动作。
2. 另外，关于告警，每条告警策略都会定义不同的优先级，Redis 中的消息队列也按优先级划分。
3. Alarm 不仅消费告警事件，优先级比较低的报警，其合并逻辑都是在 alarm 中做，所



以目前 Alarm 进程只能部署一个实例。

4. 已经发送出去的告警事件，Alarm 将会负责写入 MySQL。

## ● graph

1. graph 进程接收从 transfer 推送来的指标数据，操作 rrd 文件存储监控数据。
2. graph 也为 API 进程提供查询接口，处理 query 组件的查询请求、返回绘图数据。

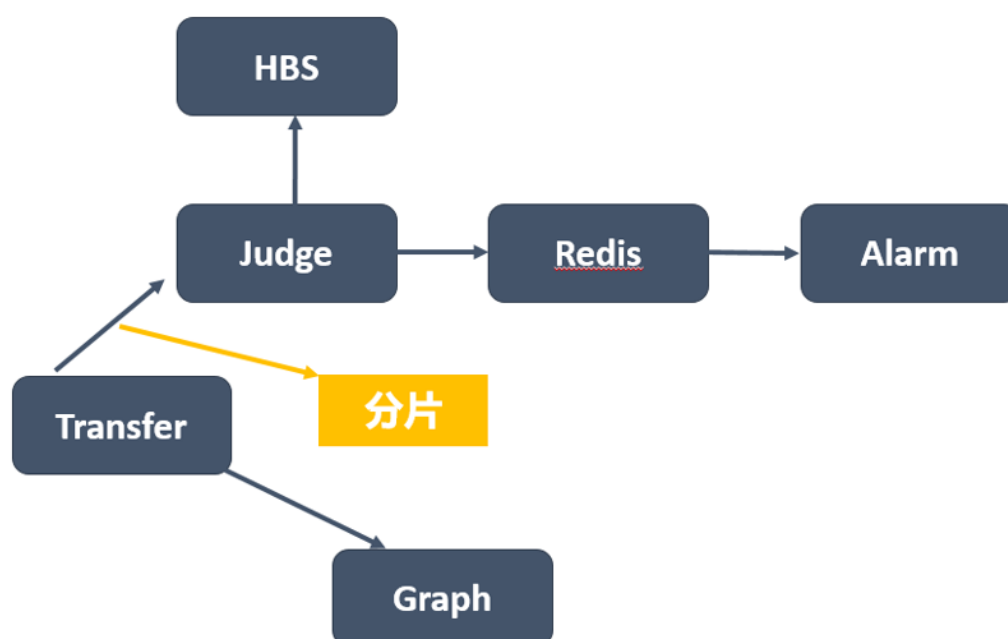


图 2-3 Open-Falcon 报警处理结构

## Open-Falcon 源码解读

### ● judge

1. HBS 获取策略列表
  - 周期性调用 RPC 接口同步策略列表
  - 策略列表包含 hostname 和具体策略
2. Transfer 获取报警数据
  - send()批量接收数据

- 与 remain 比较，超过历史点数删除数据
  - 拼接出 key，传入 map 中
3. 迅速找到关联策略
- rebuildStrategyMap()通过 endpoint 和 metric 重新处理策略列表，拼接成 key 列表
4. 阈值判断
- CheckStrategy()寻找对应的策略或表达式
  - Compute()计算点数是否触发报警
  - judgeItemWithStrategy()判断是否报警
5. 判断是否写入 redis
- sendEvent()
  - sendEventIfNeed()判断报警次数，确定是否需要写入 redis
6. 清理老旧数据和 debug 接口
- CleanState()比较时间戳，删除失效数据
  - /count 统计处理数据的条数
  - /history 获取历史数据

#### 模块测试：

停止 alarm，可以再 redis 中看到，event 是按照优先级存储的。

metric/tags [note]	condition	max	P	run	operation
cpu.idle/test1 [cpu过载]	all(#2)<100	10	6		🔔   🗑️   📄
cpu.iowait/test1	all(#1)>0.5	3	0		🔔   🗑️   📄
cpu.user/test1 [访问用户大于等于0]	all(#1)>=0	3	2		🔔   🗑️   📄
kernel.files.allocated/test2	all(#3)>1300	10	4		🔔   🗑️   📄
mem.memfree.percent/test3	all(#1)<100	10	0		🔔   🗑️   📄

图 3-1 各报警所在的优先级队列



图 3-2 redis 中的优先级存储

## ● alarm

1. 将报警分为高优先级和低优先级队列，在配置文件中设置优先级，根据优先级读取报警
2. 高优先级直接生成报警邮件、短信
3. 低优先级先进行报警合并，合并成提示信息和网页链接
4. 提供未恢复的报警列表，显示在前端界面中，作为报警处理的确认

### 模块测试:

为处理的报警中，同一个报警，Mysql 中仅保存最新一次的报警数据，没有重复数据。超过 10 次则不会继续获取该报警。



图 3-3 dashboard 上设置最大报警次数后获取的报警

id	endpoint	metric	func	cond	note	max_step	current_step	priority	status	timestamp	update_at	clc
s_10_47ee56	VM-0-17-ubuntu	cpu.idle	all(#1)	96 < 20	CPU	3	1	1	0 OK	2020-03-29	2020-03-29 06:26:00	(N
s_10_8fb5151	VM-0-4-ubuntu	cpu.idle	all(#1)	96.96969696	CPU	3	1	1	0 OK	2020-03-27	2020-03-27 02:47:00	(N
s_3_47ee563	VM-0-17-ubuntu	cpu.idle	all(#3)	95.95959595	cpu过载	10	10	6	6 PROBLEM	2020-04-14	2020-04-14 18:25:00	(N
s_3_5b12e94	open-falcon-server	cpu.idle	all(#1)	32.98969072	cpu过载	3	3	0	0 PROBLEM	2020-03-25	2020-03-25 11:05:00	(N
s_3_8fb5151	VM-0-4-ubuntu	cpu.idle	all(#3)	98.98989898	cpu过载	10	6	6	6 PROBLEM	2020-04-14	2020-04-14 19:00:00	(N
s_4_624deccf	VM-0-4-ubuntu	mem.memfree	all(#3)	65.42900003		10	10	0	0 PROBLEM	2020-04-14	2020-04-14 18:27:00	(N
s_4_980eb17	VM-0-17-ubuntu	mem.memfree	all(#3)	38.56858805		10	10	0	0 PROBLEM	2020-04-14	2020-04-14 18:27:00	(N

图 3-4 MySQL 中存储的报警数据

## 功能实现

### 中间件设计

Judge 和 Alarm 是通过 redis 配置的一个消息队列通信的，可以修改消息队列，所以我们可以把我们的算法逻辑加在 judge 和 alarm 的中间层来处理，大致步骤如下：

1. 通过模拟的报警数据使用 python 跑聚类模型，保存效果最好的模型，放到中间件中
2. 从 redis 获取经过 Judge 处理的报警数据，通过模型进行分类操作，并给出新的优先级
3. 按照新的优先级作为 key，将数据重新存回 redis 中
4. 修改 alarm 的配置文档、让 alarm 获取的经过处理的数据，再利用 alarm 的报警机制发出告警信息

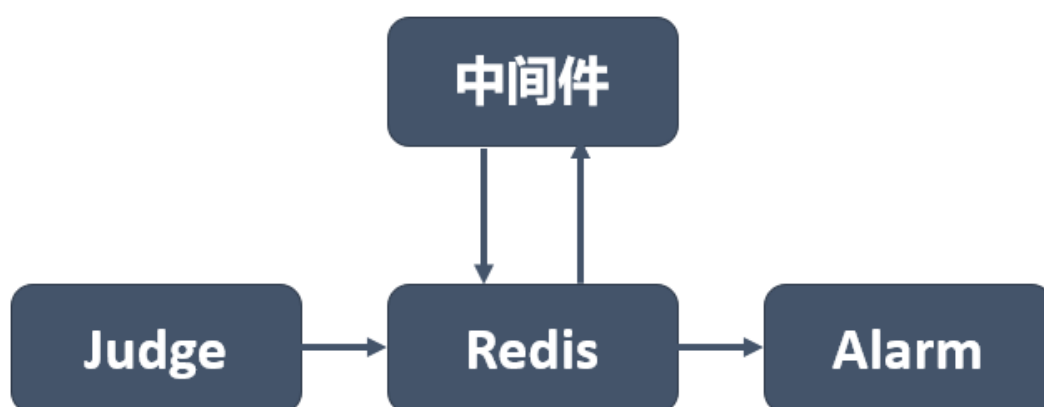


图 4-1 中间件设计

## 报错获取

### 1. 使用 chaosblade 来进行故障场景的模拟与实现

- 数据的模拟与获得一直是困扰我们的大问题之一、在报错获取方面，因为受到了设备与资源的限制，我们在有限的几台设备上尽可能真实且完善的模拟各种报错
- 我们利用 chaosblade 模拟了 cpu、磁盘、io、网络端口等等的报错，获得了在 openfalcon 上监控的大量数据，并对这些数进行我们的聚类操作

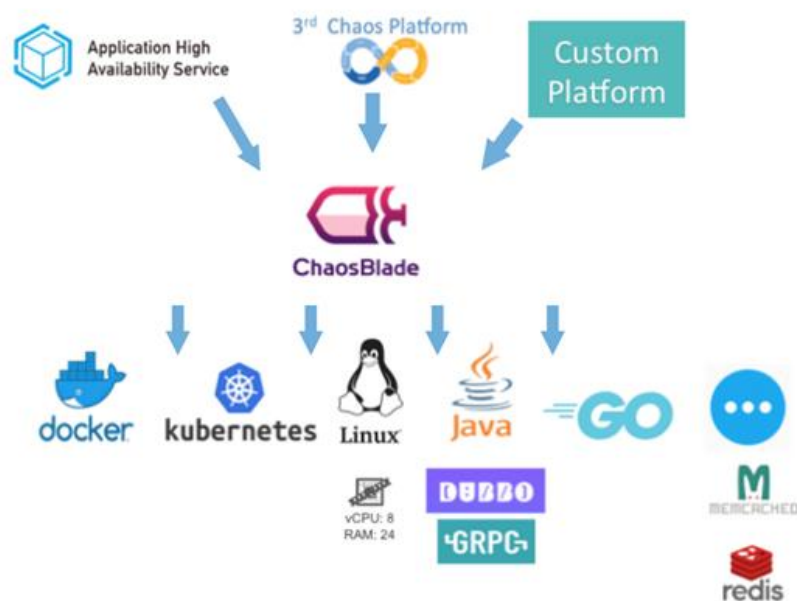


图 4-2 chaosblade 面向的操作系统

## 报警聚类

- 在聚类问题上，最主要的是聚类类别的确定与分析，以及对原始报警进行一些清洗与处理
- 将聚类的重点放在合并相似的告警数据，以便于用户进行更好的分析这方面
- 根据告警本身的 id、告警模板的 id、监视的对象、监视对象的左值右值、告警时间来

## 聚类测试

图 4-3 位聚类前的报警数据，含有 216 条数据，经过聚类后形成了图 4-4 中的 6

条数据。图 4-5 中为聚类在前端页面上的具体效果。

```
id":"s_12_37d76cbc42ba28dea1aac082702ac538","strategy":{"id":12,"metric":"cpu.iowait","tags":{"func":"all(#1)","operator":
id":"s_12_37d76cbc42ba28dea1aac082702ac538","strategy":{"id":12,"metric":"cpu.iowait","tags":{"func":"all(#1)","operator":
id":"s_12_37d76cbc42ba28dea1aac082702ac538","strategy":{"id":12,"metric":"cpu.iowait","tags":{"func":"all(#1)","operator":
id":"s_12_998b740e86680e073acd9f182a6c9e90","strategy":{"id":12,"metric":"cpu.iowait","tags":{"func":"all(#1)","operator":
id":"s_12_998b740e86680e073acd9f182a6c9e90","strategy":{"id":12,"metric":"cpu.iowait","tags":{"func":"all(#1)","operator":
id":"s_12_998b740e86680e073acd9f182a6c9e90","strategy":{"id":12,"metric":"cpu.iowait","tags":{"func":"all(#1)","operator":
id":"s_11_9a7adc453ff8f4f4fc7d2e24cc0c6e94","strategy":{"id":11,"metric":"kernel.files.allocated","tags":{"func":"all(#1)","op
id":"s_11_9a7adc453ff8f4f4fc7d2e24cc0c6e94","strategy":{"id":11,"metric":"kernel.files.allocated","tags":{"func":"all(#1)","op
id":"s_11_9a7adc453ff8f4f4fc7d2e24cc0c6e94","strategy":{"id":11,"metric":"kernel.files.allocated","tags":{"func":"all(#1)","op
id":"s_11_9a7adc453ff8f4f4fc7d2e24cc0c6e94","strategy":{"id":11,"metric":"kernel.files.allocated","tags":{"func":"all(#1)","op
```

图 4-3 聚类前获取的报警数据

```
{
  "id": "s_12_998b740e86680e073acd9f182a6c9e90",
  "strategy": {
    "id": 12,
    "metric": "cpu.iowait",
    "tags": {
      "func": "all(#1)",
      "operator": "\u003e",
      "rightValue": 0.5,
      "maxStep": 3,
      "priority": 1,
      "note": ""
    }
  },
  "id": "s_12_37d76cbc42ba28dea1aac082702ac538",
  "strategy": {
    "id": 12,
    "metric": "cpu.iowait",
    "tags": {
      "func": "all(#1)",
      "operator": "\u003e",
      "rightValue": 0.5,
      "maxStep": 3,
      "priority": 1,
      "note": ""
    }
  },
  "id": "s_10_8fb515131b2f538017751b71fbecdb",
  "strategy": {
    "id": 10,
    "metric": "cpu.idle",
    "tags": {
      "func": "all(#1)",
      "operator": "\u003c",
      "rightValue": 20,
      "maxStep": 3,
      "priority": 1,
      "note": "CPL"
    }
  },
  "id": "s_10_47ee563e6c59a014a11b469e88e0ed6",
  "strategy": {
    "id": 10,
    "metric": "cpu.idle",
    "tags": {
      "func": "all(#1)",
      "operator": "\u003c",
      "rightValue": 20,
      "maxStep": 3,
      "priority": 1,
      "note": "CP"
    }
  },
  "id": "s_11_9a7adc453ff8f4f4fc7d2e24cc0c6e94",
  "strategy": {
    "id": 11,
    "metric": "kernel.files.allocated",
    "tags": {
      "func": "all(#1)",
      "operator": "\u003e",
      "rightValue": 1300,
      "maxStep": 3,
      "priority": 1
    }
  },
  "id": "s_3_8fb515131b2f538017751b71fbecdb",
  "strategy": {
    "id": 3,
    "metric": "cpu.idle",
    "tags": {
      "func": "all(#2)",
      "operator": "\u003c",
      "rightValue": 100,
      "maxStep": 3,
      "priority": 6,
      "note": "cpu"
    }
  }
}
```

图 4-4 聚类后的报警数据

告警case列表	
<div> <div>PROBLEM P6 [第2次/最大100次] <span>just now</span> [template strategy delete 告警事件列表]</div> <div>VM_0-4-centos   load.1min   all(#3)   0.06 &gt; 0   note: original</div> </div>	
<div> <div>OK P6 [第1次/最大100次] <span>2 minutes ago</span> [template strategy delete 告警事件列表]</div> <div>VM-0-17-ubuntu   load.1min   all(#3)   0 &gt; 0   note: original</div> </div>	
<div> <div>PROBLEM P1 [第28次/最大100次] <span>2 minutes ago</span> [template strategy delete 告警事件列表]</div> <div>VM-0-17-ubuntu   cpu.idle   all(#3)   96.93877551020408 &lt; 100   note: original</div> </div>	
<div> <div>PROBLEM P4 [第28次/最大100次] <span>2 minutes ago</span> [template strategy delete 告警事件列表]</div> <div>VM-0-17-ubuntu   kernel.maxproc   all(#3)   32768 &gt; 0   note: original</div> </div>	
<div> <div>PROBLEM P1 [第2次/最大100次] <span>2 minutes ago</span> [template strategy delete 告警事件列表]</div> <div>VM_0-4-centos   cpu.idle   all(#3)   96 &lt; 100   note: original</div> </div>	
<div> <div>PROBLEM P0 [第28次/最大100次] <span>2 minutes ago</span> [template strategy delete 告警事件列表]</div> <div>VM_0-4-centos   load.5min   all(#3)   0.09 &gt; 0   note: original</div> </div>	
<div> <div>OK P5 [第1次/最大100次] <span>3 minutes ago</span> [template strategy delete 告警事件列表]</div> <div>VM-0-17-ubuntu   load.15min   all(#3)   0 &gt; 0   note: original</div> </div>	
<div> <div>PROBLEM P5 [第28次/最大100次] <span>2 minutes ago</span> [template strategy delete 告警事件列表]</div> <div>VM_0-4-centos   load.15min   all(#3)   0.06 &gt; 0   note: original</div> </div>	
<div> <div>PROBLEM P4 [第28次/最大100次] <span>2 minutes ago</span> [template strategy delete 告警事件列表]</div> <div>VM_0-4-centos   kernel.maxproc   all(#3)   32768 &gt; 0   note: original</div> </div>	
<div> <div>PROBLEM P0 [第1次/最大100次] <span>4 minutes ago</span> [template strategy delete 告警事件列表]</div> <div>VM-0-17-ubuntu   load.5min   all(#3)   0.04 &gt; 0   note: original</div> </div>	
<div> <div>全选/ 反选 批量删除</div> </div>	

告警case列表	
<div> <div>OK P6 [第1次/最大100次] <span>6 minutes ago</span> [template strategy delete 告警事件列表]</div> <div>VM-0-17-ubuntu   load.1min   all(#3)   0 &gt; 0   note: AAAAAA</div> </div>	
<div> <div>PROBLEM P6 [第1次/最大100次] <span>9 minutes ago</span> [template strategy delete 告警事件列表]</div> <div>VM_0-4-centos   load.1min   all(#3)   0.1 &gt; 0   note: AAAAAA</div> </div>	
<div> <div>PROBLEM P1 [第34次/最大100次] <span>6 minutes ago</span> [template strategy delete 告警事件列表]</div> <div>VM-0-17-ubuntu   cpu.idle   all(#3)   95.95959595959597 &lt; 100   note: idelidelidle</div> </div>	
<div> <div>PROBLEM P0 [第34次/最大100次] <span>6 minutes ago</span> [template strategy delete 告警事件列表]</div> <div>VM-0-17-ubuntu   load.5min   all(#3)   0.05 &gt; 0   note: 1515151515</div> </div>	
<div> <div>PROBLEM P5 [第15次/最大100次] <span>6 minutes ago</span> [template strategy delete 告警事件列表]</div> <div>VM-0-17-ubuntu   load.15min   all(#3)   0.05 &gt; 0   note:</div> </div>	
<div> <div>PROBLEM P4 [第6次/最大100次] <span>6 minutes ago</span> [template strategy delete 告警事件列表]</div> <div>VM-0-17-ubuntu   kernel.maxproc   all(#3)   32768 &gt; 0   note: MAXPROC</div> </div>	
<div> <div>全选/ 反选 批量删除</div> </div>	

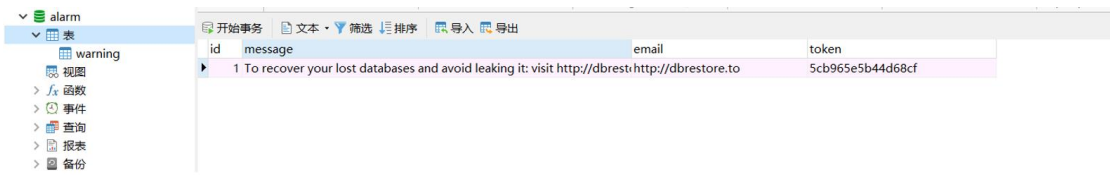
图 4-5 聚类后的报警数据在前端页面上的效果(左图为聚类前，右图为聚类后)



# 问题与解决措施

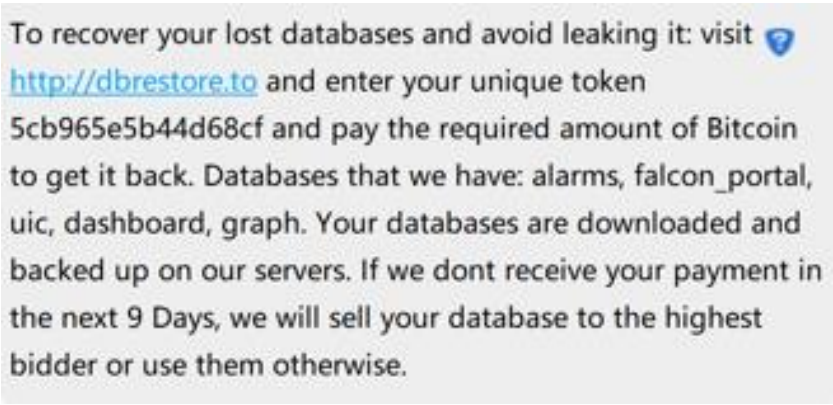
## 数据库被攻击

在开发过程中，因为服务器一直处于开启状态，并且数据库在初始化时使用的是默认密码，导致了我们的数据库被攻击。通过本次事件，我们开始将数据库的安全纳入考量。



alarm	开始事务	文本	筛选	排序	导入	导出
warning	id	message	email	token		
	1	To recover your lost databases and avoid leaking it: visit <a href="http://dbrest:http://dbrestore.to">http://dbrest:http://dbrestore.to</a>		5cb965e5b44d68cf		

图 5-1 所有数据被改为勒索信息



To recover your lost databases and avoid leaking it: visit <http://dbrestore.to> and enter your unique token 5cb965e5b44d68cf and pay the required amount of Bitcoin to get it back. Databases that we have: alarms, falcon\_portal, uic, dashboard, graph. Your databases are downloaded and backed up on our servers. If we dont receive your payment in the next 9 Days, we will sell your database to the highest bidder or use them otherwise.

图 5-2 勒索信息

## 解决措施

- 重新装载数据并连接设备

## 小组分工

洪晨晖：open-falcon 源码解读（agent 模块）、报告编辑

毛忆宁：open-falcon 源码解读（judge、alarm 模块）、前期调研（聚类算法）、ppt 编辑

韩世容：open-falcon 安装（前端、后端）

盛琼怡：前期调研（根因分析）、中间件设计、效果测试

石力源：制造报错、算法设计、效果测试