大数据平台价值调研

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本 | 修订者 | 修订描述 |
| 2018.11.08 | v1 | 林 清 | 初稿 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[1. 故障处理 2](#_Toc530417930)

[1.1 减少硬件故障：磁盘寿命预测 2](#_Toc530417931)

[1.2 减少代码BUG：FaceBook自动Debug工具 3](#_Toc530417932)

[1.3 时间序列检测 3](#_Toc530417933)

[1.4 日志管理工具 4](#_Toc530417934)

[1.5 链路监控工具 4](#_Toc530417935)

[2. 组件选型 5](#_Toc530417936)

[2.1 沟通业务场景 5](#_Toc530417937)

[2.2 数据量评估 5](#_Toc530417938)

[2.3 简要调研 6](#_Toc530417939)

[2.4 深入调研 6](#_Toc530417940)

[2.5 基准测试 6](#_Toc530417941)

[2.6 成本评估 6](#_Toc530417942)

[2.7 文档输出 7](#_Toc530417943)

[3. 结论 7](#_Toc530417944)

# 故障处理

故障处理涉及的内容比较宽泛，包括以下几点：故障诊断、快速恢复、以及监控等内容。

**造成80%的故障是由人为操作失误、硬件故障、环境故障造成的，仅仅只有20%的问题是软件Bug造成的**。

故障从发生到解决的一般流程：

0、业务失效 🡪

1、查看异常指标 🡪

2、 初步确定故障原因（根据业务流程确定，是否是数据库失败、采集失败等原因） 🡪

3、确定具体故障的组件&Role（查看各个组件的日志信息）🡪

4、 判断是否是网络、磁盘等原因 🡪

5、 判断是否是组件Bug（Google、MailList等手段）🡪

6、解决故障（调整配置、打补丁、脚本规避等手段）

上面流程的关键步骤是3、4，因为80%的原因都是因为各种环境问题。这个部分实际是进行根因分析，需要专家介入。一些莫名其妙的环境问题，如网络故障、磁盘IO飙升会影响软件的运行，想要彻底确定根因，要对组件本身有比较深入的认识、有丰富的Linux经验（懂得利用工具快速发现磁盘、网络等异常，运维技巧！！）。多数开发不具备这两个技能。

通过完善的监控工具（一些说法将监控工具分成三类：Metrics、Logging、Tracing，即指标、日志、链路），可以极大的缩短工作时间、或者降低一些工作难度。

这些工具包括：

* 硬件故障检查工具
* 指标监控工具
* 日志聚合工具
* 分布式链路监控工具
* 知识库
* 代码检查工具

以下几个例子，说明了当前这些工具的发展情况：

## 减少硬件故障：磁盘寿命预测

由于85%的硬件故障是磁盘故障，及时替换掉故障磁盘能够极大的避免设备故障的发生。磁盘故障预测在AI热潮开始之前，国内外各巨头就做过这方面的研究，比较成功的是百度。2015年，百度在IEEE上发表了相关论文[《Hard Drive Failure Prediction Using Big Data》](https://dl.acm.org/citation.cfm?id=2924693)，宣布其[HDoctor系统](https://segmentfault.com/a/1190000000347668)实现了97.82%的检出率以及0.3%的误告率。百度没有开源HDocker。

大部分磁盘预测的算法，其数据源都来自磁盘的[Smart指标](https://www.cnblogs.com/york-hust/p/3380928.html)，在这些指标上使用神经网络、SVM等算法评估磁盘的状态（Smart指标，通过smartctl、Disk Genius等工具非常容易获得）。

当前，网上不到AI+磁盘检测的开源软件，但是相关的研究论文非常多。

## 减少代码BUG：FaceBook自动Debug工具

FaceBook从今年9月起，在[Developer Tools](https://code.fb.com/developer-tools/)网站上陆续介绍了3个自动Debug的AI，这三个工具都是为了测试安卓APP代码设计的，目前都已经在FaceBook内部上线的。后续FaceBook会将这个三个工具开源。

[Sapienz](https://code.fb.com/developer-tools/sapienz-intelligent-automated-software-testing-at-scale/)：自动生成大部分测试用例。

[SapFix](https://code.fb.com/developer-tools/finding-and-fixing-software-bugs-automatically-with-sapfix-and-sapienz/)：基于知识库的debug系统，主要发现代码中可能使程序崩溃的Bug（后续可能聚焦导致性能问题的代码）， 并且自动提出多个修改意见，进行自主验证。当自主验证通过后，SapFix会通知工程师进行审批。SapFix使大部分Debug工作自动化，人工只要采纳SapFix的建议即可。

[Getafix](https://code.fb.com/developer-tools/getafix-how-facebook-tools-learn-to-fix-bugs-automatically/)：Getafix是Debug系统，他与SapFix的区别在于，SapFix是基于模板库进行Bug修复的，而Getafix同时具备了一些学习功能。Getafix除了能够修订Sapienz测试出的Bug以外，还能修复Infer（FaceBook已经开源的静态代码检查工具）扫描出的Bug。

从Facebook公布的信息来看，上述工具都处在概念阶段，应用场景还比较单一（只能用于安卓代码的Debug）。

## 时间序列检测

时间序列检查，是指从一个或者多个时序指标（如磁盘IO、网络流量等）中提取规律，诊断当前指标是否存在异常，或者对指标的短期趋势做出判断。

时间序列检查是AIOps中非常聚焦的一个研究方向，AIOps会借助时间序列检查指定的KPI指标，对系统状态做出判断和预测，进而采取相应的措施。

当前Git上可以找到大量的时间序列检测算法库，这些库应用在不同的场景，基于不同的算法，效果也不尽相同。以下是[GIT](https://github.com/rob-med/awesome-TS-anomaly-detection)上收集的时间序列检查的相关算法库（Star数目超过1000）。

|  |  |  |  |
| --- | --- | --- | --- |
| Name | 语言 | Star数目 | 说明 |
| [Nupic](https://github.com/numenta/nupic) | C++ | 5726 | 基于HTM的时间序列检查算法库 |
| [Skyline](https://github.com/etsy/skyline) | Python | 1999 | etsy公司开源的指标检测工具，能够同时检测数百个不同指标。 |
| [AnomalyDetection](https://github.com/twitter/AnomalyDetection) | R | 2713 | 推特开源的基于Seasonal Hybrid ESD算法的检测工具。 |
| [Prophet](https://github.com/facebook/prophet) | R/Python | 6707 | Facebook开源的时间序列预测系统。 |
| [PyFlux](https://github.com/RJT1990/pyflux) | Python | 1280 | PyFlux 是一个包含了多个时间序列模型的算法库。 |

## 日志管理工具

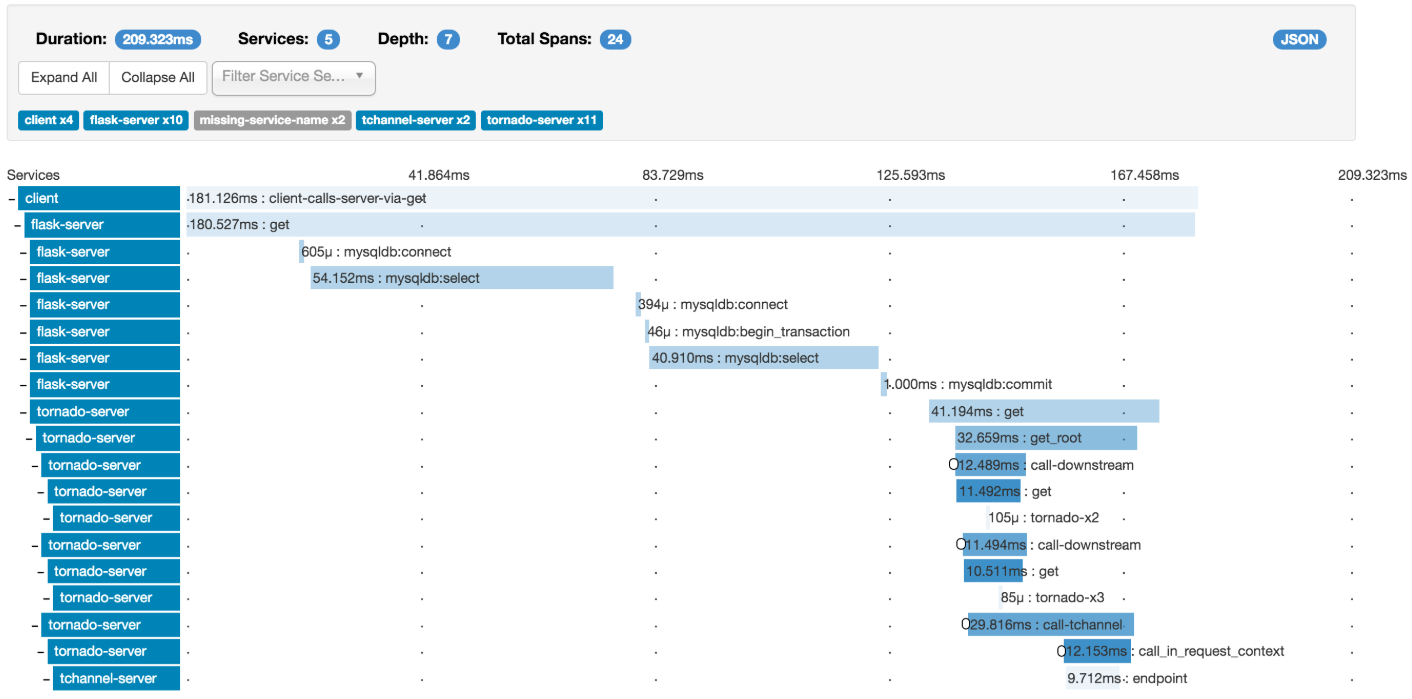
运维规模较大的集群时，日志管理工具能够极大的提升定位故障的效率（Openstack运维时有深切体会！！）。当前可以找到相当多的Log Management多的方案，这些方案提供日志聚合、报表展示、事件告警、日志审计等功能，部分方案还融合了一些数据挖掘的算法。绝大部分日志管理工具是付费的。

开源方案最常见的是ELK，从易用性来说比商业软件差很多。

参考：50个[Log Management](http://mcp.ruijie.net/auth/wifidogAuth/newLogin/?gw_id=00d0f82233ea&gw_sn=1234942570022&gw_address=192.168.59.62&gw_port=2060&ip=192.168.21.37&mac=58:69:6c:ca:6d:a1&slot_num=0&nasip=192.168.59.62&ssid=@ruijie&ustate=0&mac_req=0&url=http%3A%2F%2Fwww%2Ebaidu%2Ecom%2F)介绍

## 链路监控工具

链路监控工具（Tracing），可以监控一个请求从发起到结束的整个生命周期。下图是OpenZipkin工具的系统监控页面。页面中，我们可以看出请求流过各个组件所用的时间，以及是否发生了连接失败、重试等。



Tracing的概念最早由Google提出（2010谷歌的两篇论文，[论文1](https://research.google.com/pubs/pub36356.html?spm=a2c4e.11153940.blogcont514488.19.11b730c2Zxoi7W)，[论文2](https://research.google.com/pubs/pub40378.html?spm=a2c4e.11153940.blogcont514488.20.11b730c2Zxoi7W)）。分布式链路监控系统最有用的场景是**性能测试（获取请求在各个节点的耗时）**、以及**故障定界（获取故障在哪个组件发生了请求失败）**。由于我们的业务涉及到的组网并不复杂，因此个人认为作用上：故障定界小于性能测试。

当前流行的开源分布Tracing系统包括：

* [OpenZipkin](https://zipkin.io/)（twitter开源）
* [Appdash](https://sourcegraph.com/start)（Go语言环境下的Tracing系统）
* 云服务商提供的Tracing系统，如AWS X-Ray、TracingAnalysis（Ali）等

分布式追踪系统发展很快，种类繁多，但是目前绝大部分Tracing系统，需要在修改业务代码（埋点），才能将系统接入Tracing系统中。

# 组件选型

组件选型涉及到的工作流程包括：

* 沟通业务场景
* 数据量评估
* 简要调研
* 深入调研
* 基准测试
* 成本评估
* 文档输出

上述是选型工作的流程拆解。实际上选型时，我们可能跳过某些步骤或者某些步骤工作不到位，导致选型结果不到位。

下面简要说明，每个步骤存在的问题以及是否有优化改进的建议。

## 沟通业务场景

沟通时需要明确哪些内容：

硬件规模、开发语言、数据流向、产品对标对象、用户类型、使用场景等等……

## 数据量评估

数据量评估是选型过程最重要的环节，后续进行基准测试时需要此部分数据作为参考依据。

需要评估的数据量包括：**原始数据、中间数据、结果数据**

* **原始数据**：即采集器获取到的数据。由于我们公司大部分项目的数据源是一致的，包括：AP、AC、流量等等。**如果评估手段合理，部分数据量可以公式化，在不同项目中进行复用。**
* **中间数据**：即流入各个组件的数据压力（如Spark）。由于此时业务没有开发完成，中间数据的数据量很难估计，只能对原始数据进行简单分析，给出一个参考值。
* **结果数据**：同中间数据，只能给出参考值。

原始数据、中间数据，主要的指标是流量大小、数据并发度。

结果数据，主要指标是数据规模、老化时间。

## 简要调研

资料收集过程，通过博客、Apache 官网等，尽量收集同类型组件的资料，并且按照给定的几个维度进行打分。需要收集的可能包括以下：

* 官网地址
* 应用场景、组件特点
* 社区活跃度：参考Git上信息
* 关键功能：是否支持HA、可扩展、API接口类型等等
* 使用案例

该部分工作难度不高，在确定信息收集内容以后，只需要花少量时间收缩即可。

## 深入调研

对单个组件进行较为深入的使用，工作包括：

* 了解部署方式
* 了解基本架构
* 了解组件Features
* 了解API接口
* 了解编写Demo程序

该工作实际上是深入学习组件的一个过程，当前调研可能超过60%的工作量发生在该环节。通过官方文档能够完成绝大部分工作。

## 基准测试

进行基准测试需要对组件的功能有一定了解。不同组件的指标是有差异的，需要组件的运行原理有比较深入的理解，才能对测试结果是否合理进行评价。**测试很容易，但是对结果进行分析需要一定“功力”。**

绝大多数组件在网上可以找到基准测试工具。

官方基准测试方案：

Hadoop：[参考1](https://hadoop.apache.org/docs/r2.8.0/hadoop-project-dist/hadoop-common/Benchmarking.html)、[参考2](https://dirtysalt.github.io/html/hadoop-benchmark.html)、[参考3](https://github.com/intel-hadoop/HiBench)

Spark：[参考1](https://codait.github.io/spark-bench/)

Kafka：[参考1](https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines)、[参考2](https://github.com/fede1024/kafka-benchmark)

Mongodb：[参考1](https://github.com/mongodb/mongo-perf)

## 成本评估

包括：硬件成本、开发成本。

## 文档输出

应当包括《架构&成本说明文档》、《安装部署文档》、《API文档》、《基准测试文档》、Demo程序。

# 结论