




微服务架构 之服务管理

研究报告

Wen Zhenglin

2016-6-29



概要

1. 背景
2. 研究目标
3. 服务发现
4. 服务监控
5. 总结

背景

微服务是一种分布式系统架构，第一期的研究结果解决核心的数据通信和数据交互格式的定义问题)

微服务架构的实施仍面临多项挑战，其中以服务管理为首要问题

背景(续)

服务管理主要分为两类

- 服务发现(服务之间如何发现对方)
- 服务监控(服务的运行状态监控)

研究目标

通过对服务管理的研究,使微服务的管理更加完善,解决因服务管理的问题,避免造成服务管理上不可控的局面带来的损失

一、服务发现

研究内容

有什么主要难点

是否有现行的对应开源技术

它们的特点如何

对开发人员的影响

主要难点

服务之间如何发现对方的位置(IP地址和端口)(同时起到解耦的作用)

提供服务发现的服务自身如何保证高度错误可容忍性

如果是集群如何保证集群系统间的数据一致性

现行的对应开源技术

服务发现的三个主要的开源技术

Zookeeper

Etc

Consul

特点比较1

Area/Name	Zookeeper	Etcd	Consul
Service concept	no	no	yes
Multiple datacenter	no	no	yes
User interface	no	no	yes
Key-value store	yes	yes	yes
Event watching	yes	yes	yes
DNS support	no	no	yes
Health monitoring	no	no	yes
Polyglot (多语种) support	Java-only + 3rd-party	Go-only + 3rd-party	<u>yes</u>

特点比较2

Area/Name	Zookeeper	Etcd	Consul
Written in	Java	Go	Go
HTTP-API	no	yes	yes
Client-API	no	http-json	http-json, config-file, dns
CLI	zkCli	etcd-ctl	consul-cli
CAP type	AP	AP	CP
Consensus algorithm	paxos-variant	raft	raft
Vendor	Apache	CoreOS	Hashicorp
License	Apache License 2.0	Apache License 2.0	Mozilla Public License, version 2.0

比较结果分析1

以上三者都通过集群的方式解决自身的错误可容忍性问题，并通过一致性算法保证数据一致性，其中Zookeeper的Multi-Paxos算法因算法研究报告本身较侧重于理论，导致实现算法时带来的变化，并且由于该算法的复杂性导致集群系统的行为难于理解，不仅带来实现上困难，同时给维护也增加了难度，出现问题的概率较高。

Raft在Paxos的基础上进行了改进，使之更简化，Etcd和Consul都是使用Raft算法实现，但却是各自的实现，Consul的Raft的实现是建立在Etcd之上，以更切合Consul自身的需要，在一致性算法上我认为Consul胜出。

比较结果分析2

对于服务发现, Zookeeper和Etcd主要功能为key-value存储, 可勉强用较原始的方式实现服务发现, Etcd额外提供了HTTP接口, 对客户端的使用变得更简单, Consul提供了服务的概念并可通过HTTP和DNS等方式进行服务发现, 实现的较为完善, 通过与服务对应的健康检测和用户接口, 可在服务管理上满足我们切实的需求。

比较结果分析3

除去共同部分(key-value store和watch event), 可看出Consul比较突出, 不仅提供了服务的概念(更符合服务发现的方案)和服务健康状态的检测, 且在其它方面也提供了合理的实现(多数据中心支持, 用户接口, HTTP和DNS, 多语种等)还包含一些其它细节Consul也比较符合我们的的要求(如可维护性, 易用性等)

以此得出Consul作为服务发现的主要技术, 并作为基本的服务监控技术

比较结果分析4

关于Zookeeper，由于公司将要用到服务发现技术的大部分人员以java为主，Zookeeper显得有一定的吸引力，以下是一些我对此的看法

除去服务发现的众多要素考虑之外

关于Zookeeper的key-value store性能优势(有待确认)(对于服务发现不建议用key-value存储大量数据)对此key-value store的性能不能成为主要考虑依据(关于key-value store的性能方面，三大解决方案预计都能满足我们的需求)

由于Zookeeper使用的一致性算法较为难懂，和使用Zookeeper本身的难度，加上维护的问题，不建议使用Zookeeper(观点来自于官方资料和其它人的使用经验)

对开发人员的影响

Consul提供的接口较为简单，组织结构上容易理解，提供的DNS接口除了可以更丰富的定制服务发现外，也使传统的应用也更容易支持服务发现

通过对时间服务添加服务注册和服务发现的试验，不仅作为示范的作用，同时也可表明其中的简单性

通过对文件服务在外层包装服务注册，表明一种更简单性的无代码侵入的方案，但服务发现的实现方式通过外层包装有一定的弊端(不利于支持错误可容忍性，进而使该服务比较脆弱)

二、服务监控

研究内容

有什么主要难点

是否有现行的对应开源技术

它们的特点如何

对开发人员的影响

主要难点

大数据量(监控数据在节点规模, 时间细化, 数据类型的多样性)给整个监控系统带来的挑战

监控数据的可视化

监控数据的采集

现行的对应开源技术

ElasticSearch + Logstash + Kibana

Prometheus + Grafana

特点比较

	ElasticSearch + Logstash + Kibana	Prometheus + Grafana
实现语言	Java + Ruby + Javascript + Go	Go
结果获取方式	Push to elasticsearch	Prometheus do Pull (or PushGateway)
倾向于	日志分析监控	度量值监控
日志分析技术	Logstash (grok plugins) (regexp)	Mtail (regexp)
日志处理模型	Input ->Filter->Output	Condition ->Action
定制导出	Beats(Filebeat,Packetbeat,Topbeat)	Exporter (NodeExporter,HAExporter)
可视化接口	Kibana	Grafana
图形种类	多种(line,map,bars)	线型 (line, stack line)

比较结果分析1

两者对日志的分析都依赖正则表达式，Prometheus本身未提供日志分析功能，而是通过第三方Mtail实现

Logstash的日志分析主要通过Grok特定模式语言匹配实现，Mtail则是单纯的正则表达式实现模式查找

日志分析模型Mtail较为简单，可做的修改也较为有限，Logstash支持更丰富的修改及插件定制

比较结果分析2

通过试验Prometheus在使用上较为简单，对查询的响应速度较理想，数据模型更倾向于度量值，适合服务的运行状态监控，但数据的来源是一个短板，依赖于服务上程序内部加入相关代码，也可采用日志分析对服务进行监控，需引入日志分析技术方可达到无代码侵入性监控

Elasticsearch在日志分析较为强项，通过试验Logstash的运行速度明显感觉较慢，个别例子无结果输出，且无处理情况的提示

比较结果分析3

图形方面Grafana的种类也相对较少，主要是以时间序列化数据的度量值可视化为主，Kibana则相对较多，但对应数据的生成却更复杂

Grafana支持的数据源较多，独立于Prometheus，Kibana则支持的数据源较少，须结合ElasticSearch使用

对开发人员的影响

ElasticSearch相应的监控方案带来的学习成本较高，另外使用上也存在不确定性，使风险增加

Prometheus相比较而言，较为简单，但依赖程序内部作相应的指示性代码的嵌入，使业务代码增加了额外的复杂性

另外一种方式是通过日志分析技术实现数据的采集，须对日志作一定的规范说明，不仅是格式上的规范，对日志里包含什么数据也须作更多的考虑

后续建议

目前服务监控为初步结果，通过试验Prometheus对传输队列监控进行度量值转化并生成对应的运行规律分析，得出可用于一般性分析，但实际用处还有待考量

ElasticSearch整个系统较为庞大，除开Logstash性能较大的问题，需要更深入的了解和理解才能真正发挥作用

总结

本研究通过对服务发现和服务监控的主要问题和相关对比得出：

服务发现的主要技术可通过Consul实现

服务监控的主要技术可通过Consul实现基本的状态监控，可通过Prometheus对服务的运行状态监控，要实现无代码侵入性的监控，则对日志分析技术有更多的要求。