OpenResty与语音交互

— OpenResty及其组件的全栈开发

张顺@AISpeech

2016年12月

内容

- 用OpenResty构建语音云服务
- OpenResty组件在语音交互系统里

第一部分

用OpenResty构建语音云服务

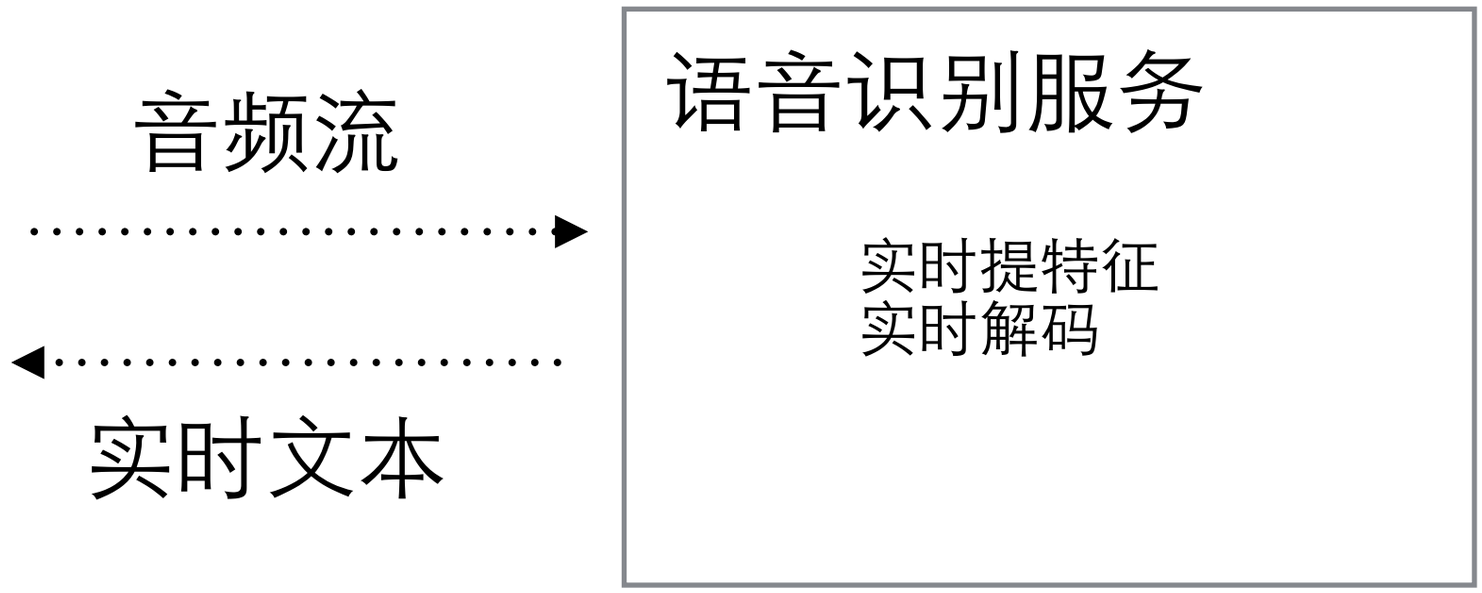
语音服务的特点

语音识别　audio -> text

语义理解　text -> intent, slots

语音合成　text -> audio

语音服务的特点

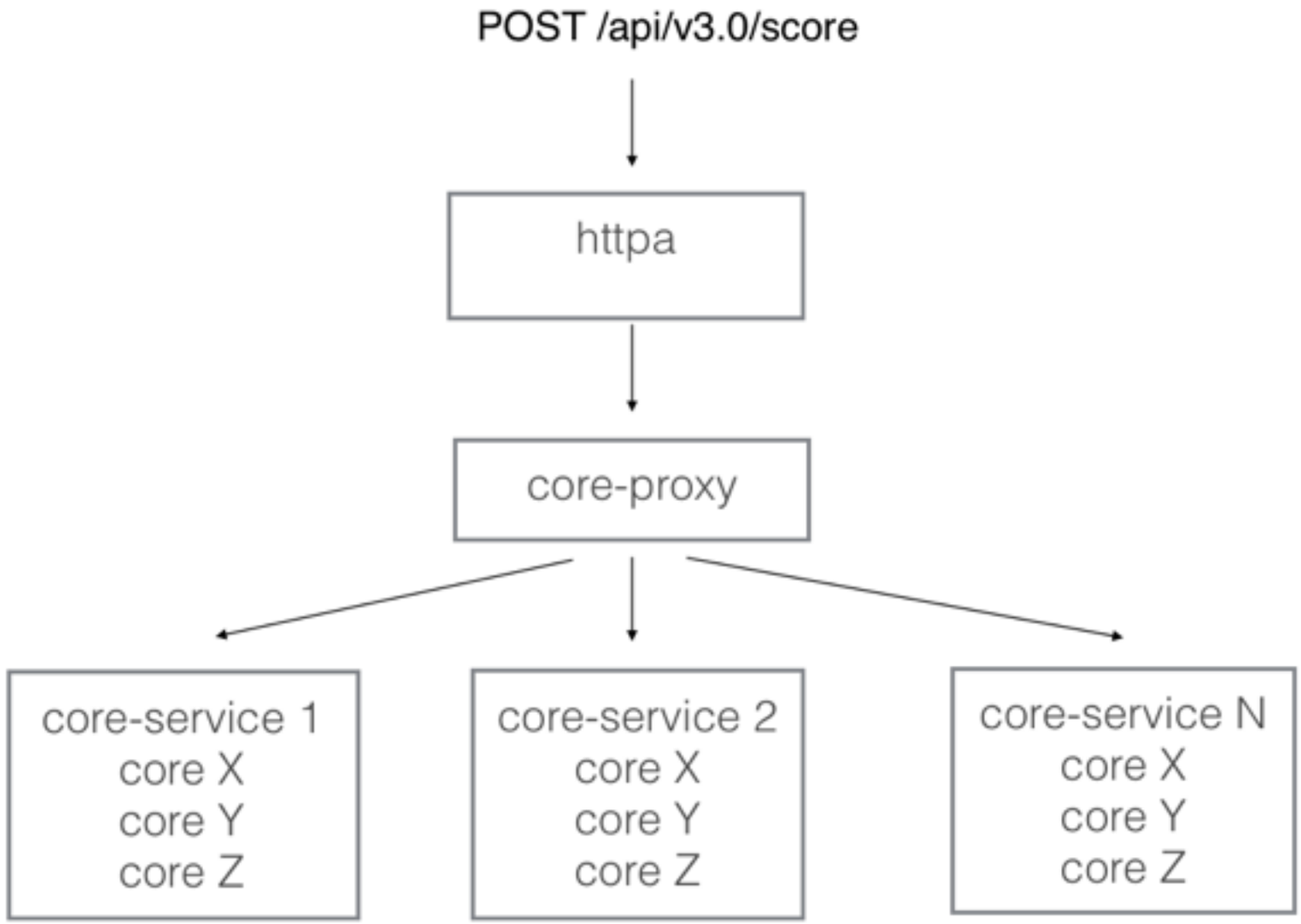


* 实时计算
* 单机并发少
* xxG模型
* 计算密集型
* 双向通信

旧版本服务

当年
几个初生牛犊的工程师参考nginx等用c语言自已实现了
http proxy, server, client

两个”理由”:
我们要自已完全可控, 不使用任何第三方的代码
我们能实现得更好



server端分三层:
httpa, core-proxy, core-service

慢慢地旧服务的诸多问题呈现了出来:

1. 很难扩展新功能
2. 性能差, httpa/core-proxy仅支持1k并发, 更别谈C10K
3. 整体架构层次太多, 部署维护很费神
4. 单服务多内核对机器要求太高
5. 代码结构设计不合理
6. 代码风格奇特, 可读性差
7. 文档不全, 新人无从快速上手

跟不上业务发展的需要

基于OpenResty的新服务

2014年初我们开始重新思考计算密集型服务的设计和实现

只有一个核心诉求: 简单

这也是UNIX哲学里最重要的KISS原则：

1. 简单才易于开发和维护
2. 简单才易于整体架构
3. 简单才易于扩展
4. 简单才易于做到稳定
5. 只有做简单才有资格谈性能

由于有了上次的教训, 所以这次绝不从头开发。

尝试基于nginx开发, 当开发完第一个c module时, 觉得还比较麻烦。

偶然发现了春哥开发的OpenResty。

由于略懂Nginx, 平时又受益于lua, 便尝试OpenResty作为计算服务的容器.

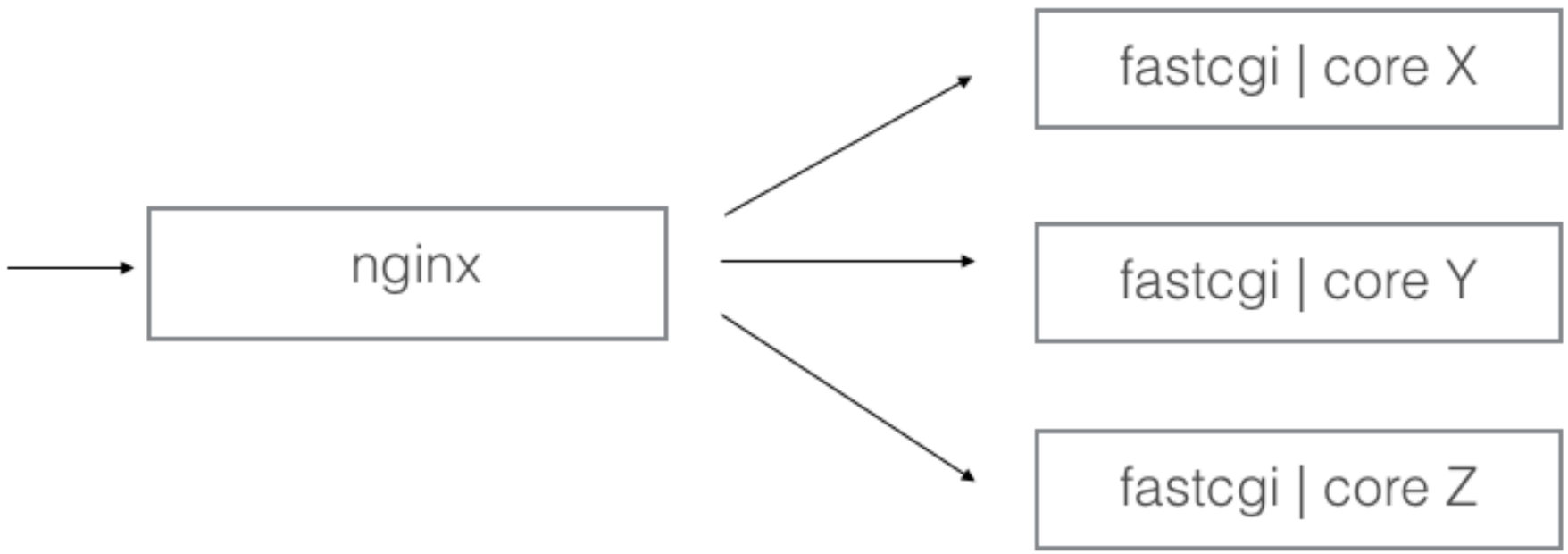
Nginx作为单线程EventLoop的代表, 以其每秒轻松过万的QPS的出色性能, 一般是作为网络IO密集型的web服务器或代理服务器.

OpenResty的出现让nginx具备应用服务器的能力.

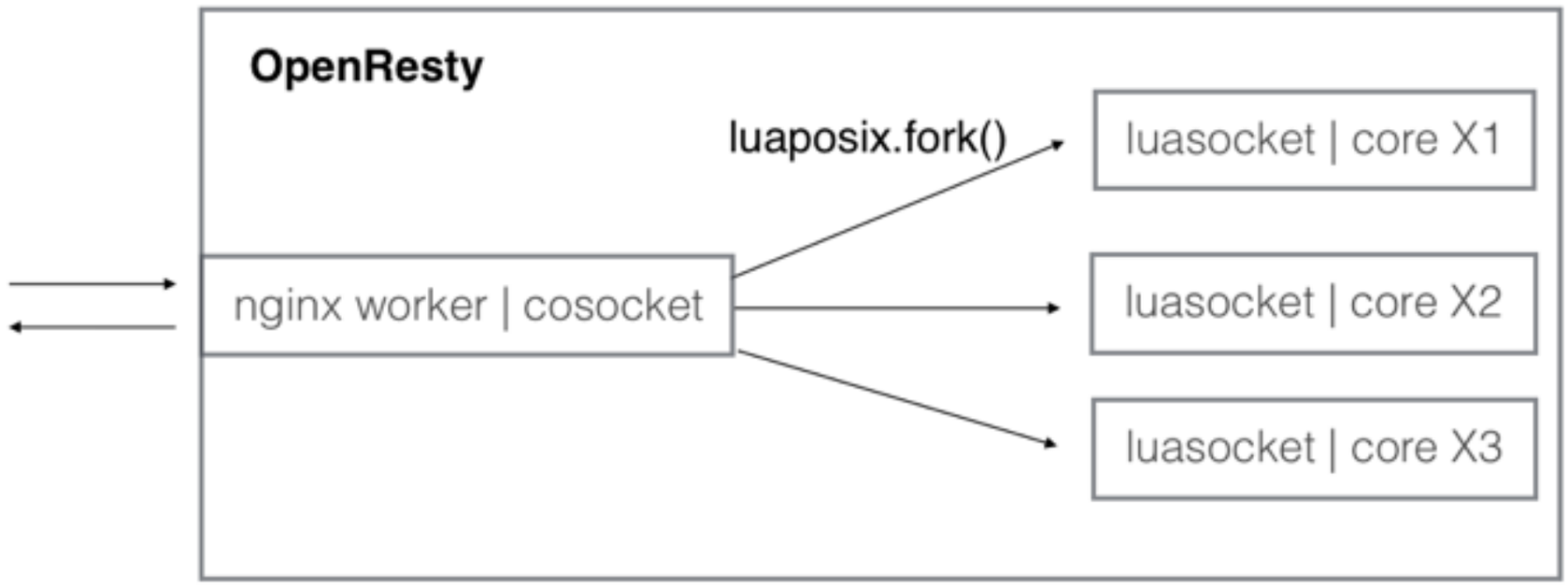
OpenResty作为计算服务容器还有两个主要问题需要解决:

1. 计算与主循环分离
2. 与核心c语言计算代码集成

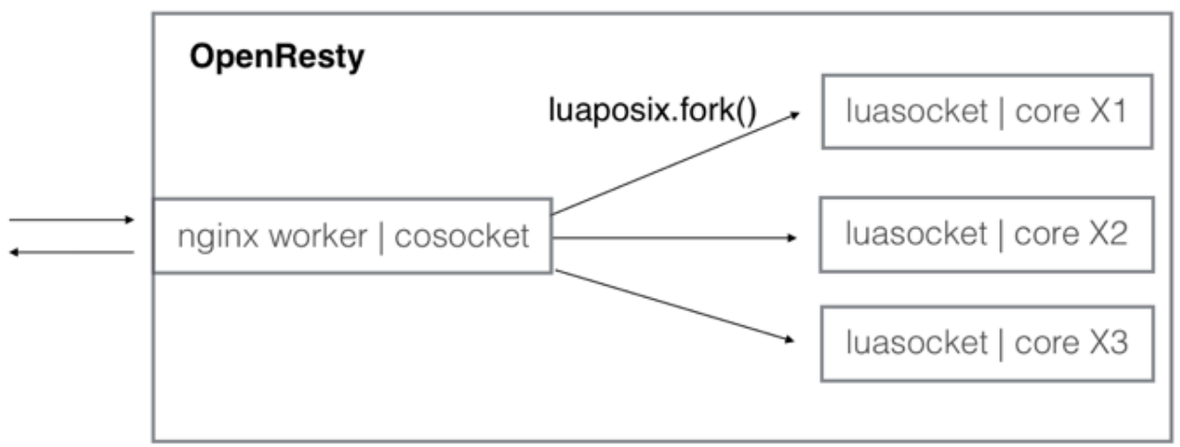
OpenResty的lua代码是执行在主线程事件循环里的, 在主循环里不能有任何复杂的计算或其它可能阻塞主循环的代码。



经典方案: nginx + fastcgi

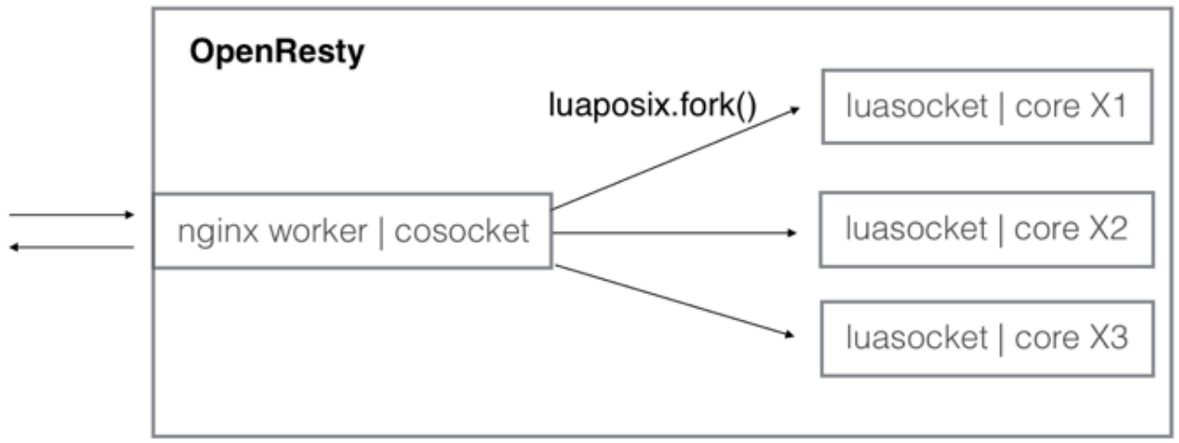


all in OpenResty



1. 只使用一个nginx worker

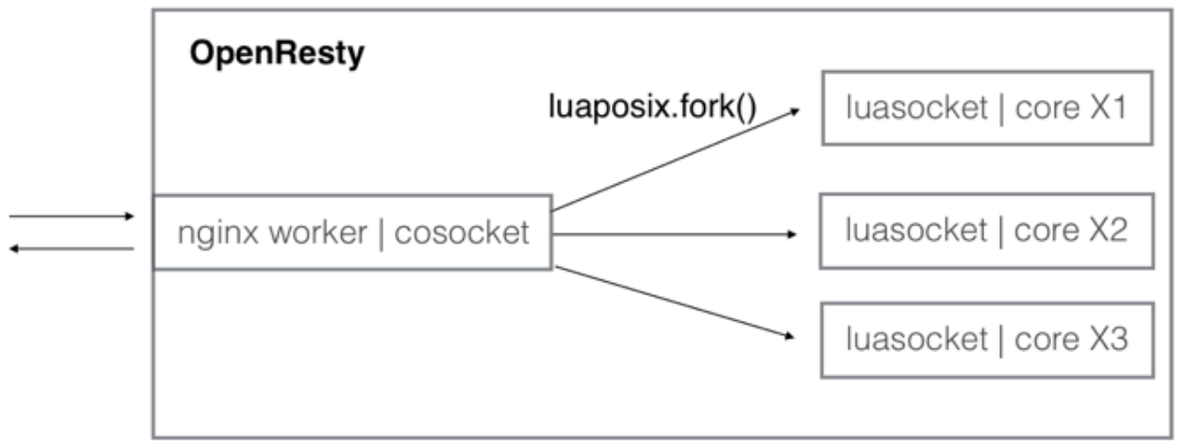
因为是计算密集型服务, 一台服务器只能处理不到一百个并发请求甚至更少, 一个nginx worker处理网络IO足够了



2. posix.fork()创建计算进程

ngx_lua没有提供fork方法, 这里用的是luaposix, luaposix提供了丰富的posix方法的lua binding.

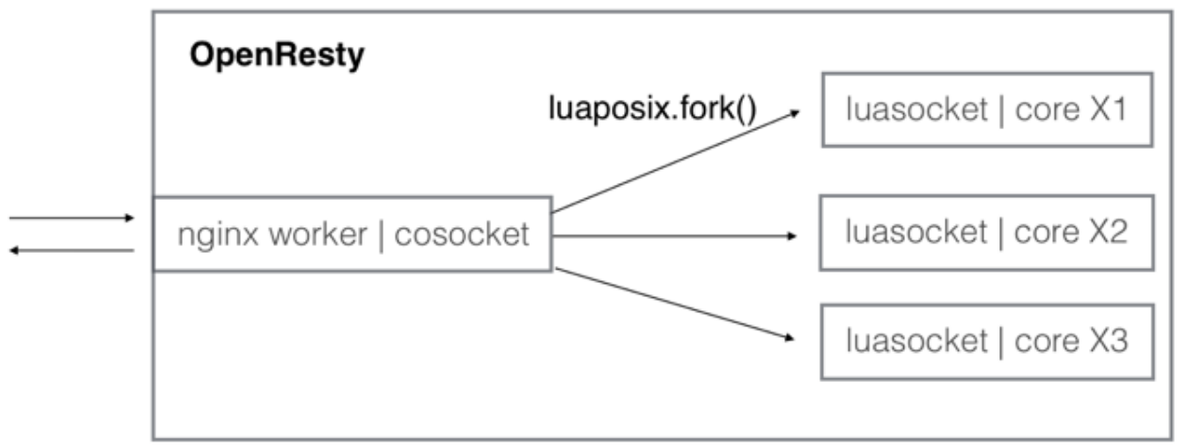
另外由于还需要对计算进程有更灵活的控制, 比如某个计算耗时异常时需要强行kill, 某个计算进程crash时的特殊处理, 所以选择了自已fork并管理计算进程.



3. nginx worker和计算进程间通过tcp通信

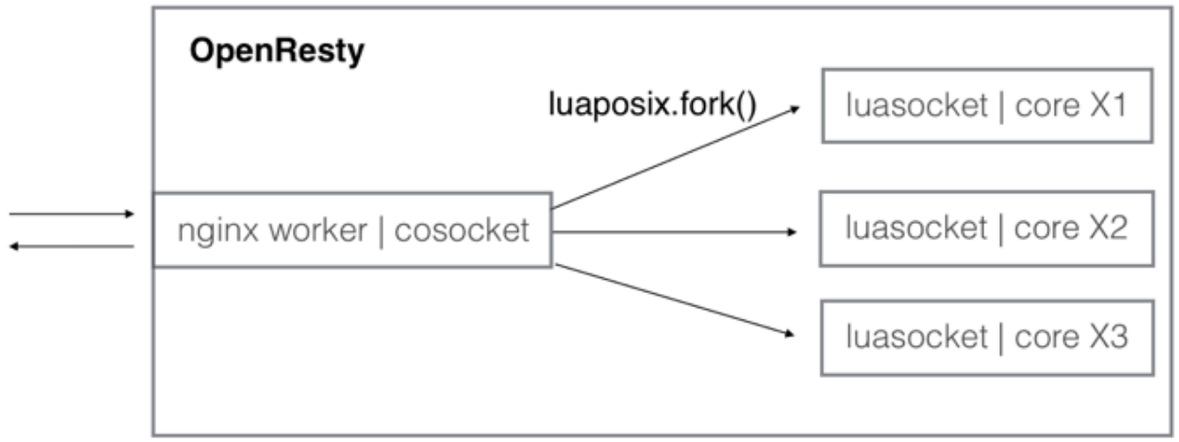
为了避免block主循环, nginx worker端的服务代码里使用cosocket与计算进程通信即可

计算进程用luasocket



4. 所有服务逻辑都用lua实现

a. 设计精巧, 核心代码量只有1w多行, 编译后完整的VM小于100KB
b. 内存开销小, 唯一字符串只有16B, 闭包只有24B 运行效率高, 加上LuaJIT更是接近c/c++
c. 原生支持coroutine, 依靠coroutine完成yield/resume的切换开销很小

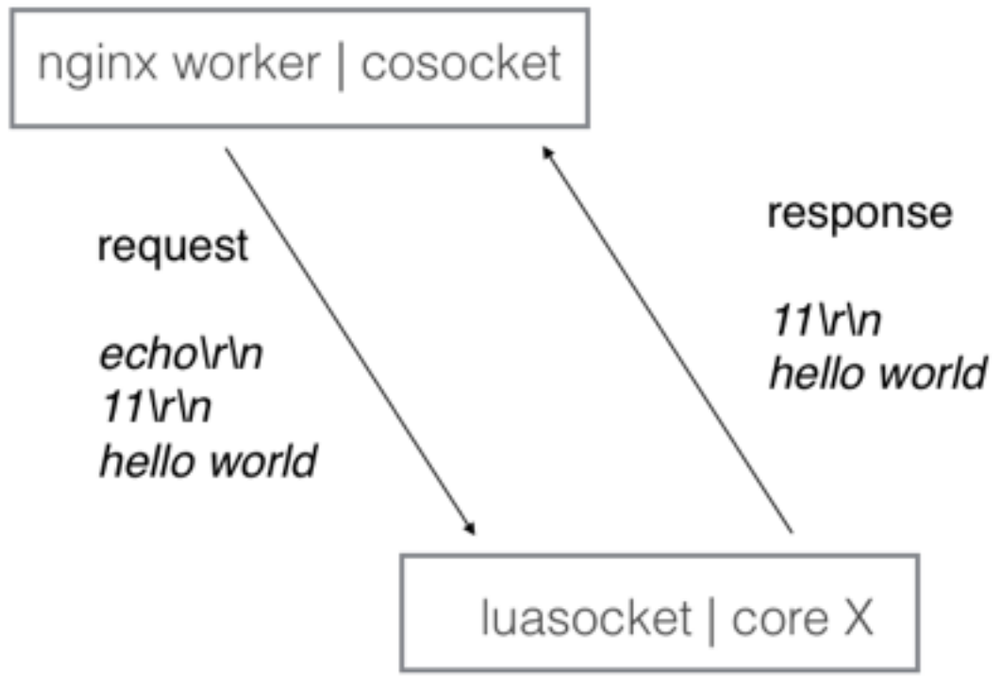


5. 对计算进程的一些特殊要求

fork后要释放无用的资源, 例如fd

不能使用ngx.socket, ngx.sleep, ngx.timer, ngx.log, ngx.thread等ngx_lua里的方法, 在计算进程用luasocket与nginx worker进行同步阻塞的tcp通信

在计算进程里只需埋头计算, 计算完成后等待下一个计算请求



6. 进程间简单的文本协议

简单直白可读易解析.

7. websocket

与客户端双向通信

TODO

8. 灵活的负载均衡

得益于balancer_by_lua

无需nginx -s reload

一些优化的设计和实现

1. 共享内存

linux在fork时对内存的管理是写时拷贝(copy-on-write), 当有大量数据属于只读型时, 可以在init_worker_by_lua里一次性将资源加载到内存里做共享, 可以节省大量物理内存, 以及节省加载时间. 在我的实际项目中有xxGB的模型资源需要一次性load到内存.

2. 进程池

在我的intel i5机器上posix.fork()的耗时是十毫秒量级, 这当然会block住主循环的, 可以加一个进程池管理机制, 保留一些空闲进程, 同时增加计算进程的重复利用来做一定程度上的优化.

3. 过载保护

增加这个机制是防止计算负载过高时假死, 一般有两个控制参数max_utilization和pending_time.

max_utilization是cpu/gpu最高使用率的阈值一般设为0.8

pending_time是忙时等待时间一般设为2000ms. 使用率超过0.8就代表机器繁忙, 这时新的请求最多等待2000ms, 如果2000ms时间内cpu使用率降下来则处理计算请求, 否则响应server busy

4. 微服务

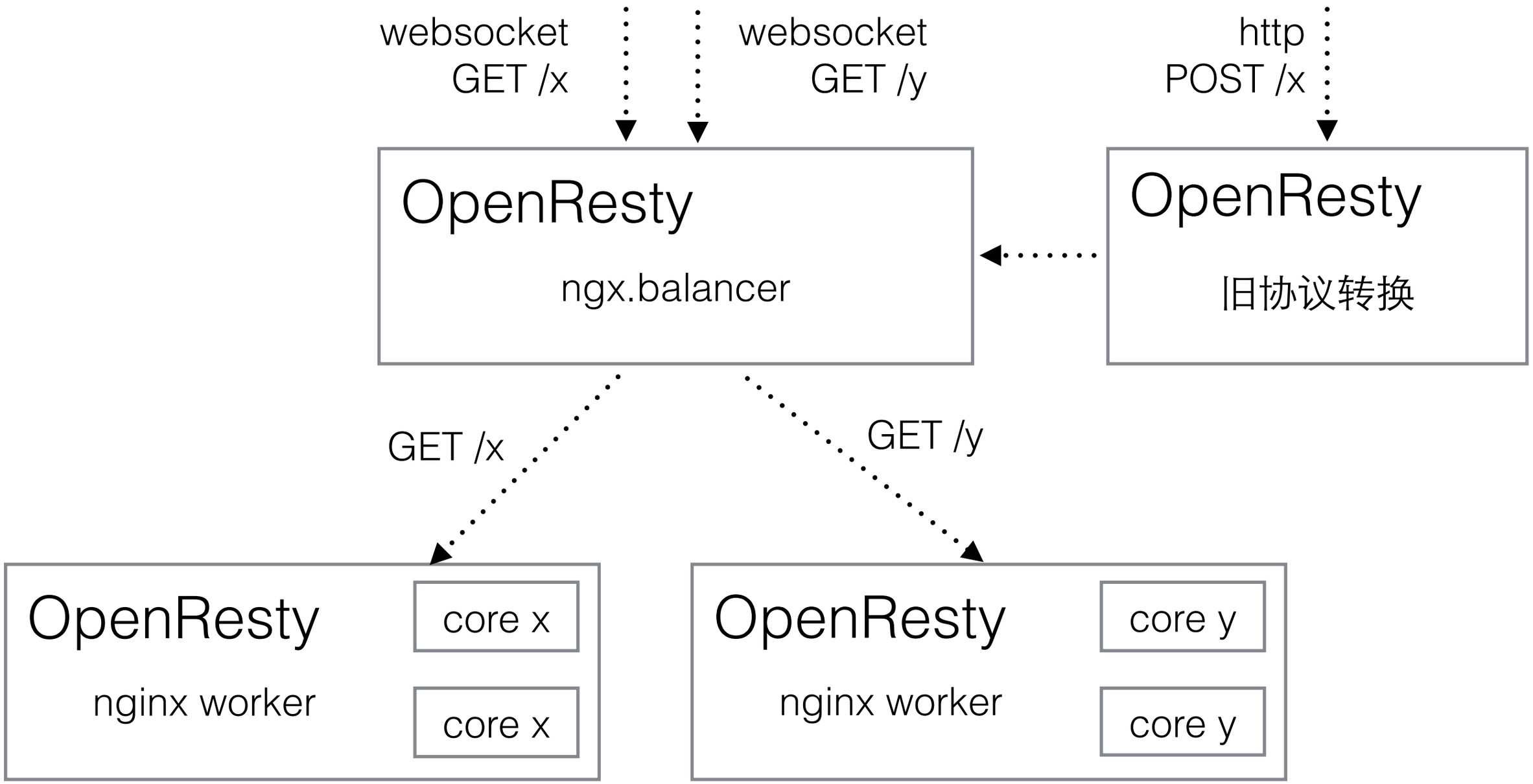
服务切分:

a. 一种类型的计算作为一个单独的服务

b. 一个nginx实例只运行一种类型的计算服务

不同服务用uri区分 这样可以方便用成熟的代理程序来实现负载分发, 例如haproxy或nginx

注意: 同一个连接只能处理一种语音服务请求


websocket
GET /x
websocket
GET /y
http
POST /x
OpenResty
ngx.balancer
OpenResty
旧协议转换
GET /x
GET /y
OpenResty
nginx worker
core x
core x
OpenResty
nginx worker
core y
core y

新计算服务逻辑架构

最终那5万多行c代码被简化成了几千行lua代码

以前要维护的三个服务程序也变成了一个

逻辑架构也变得简单

对OpenResty的贡献:

1. feature: duplex cosocket - OpenResty v1.7.2.1
2. bugfix: tcp_nodelay - OpenResty v1.7.4.1

演示/视频 - 实时语音字幕 <语音输入板>

https://v.qq.com/x/page/f0155wqxlmz.html

第二部分

OpenResty组件在语音交互系统里

一些背景知识铺垫

人机交互方式的发展

1868年克里斯托夫·拉森·肖尔斯发明QWERTY键盘

1968年道格拉斯·恩格尔巴特发明鼠标

1971年SamHurst发明触摸传感器

2007年iPhone诞生，人类进入智能机时代，触摸成为主流交互方式

语音、手势、眼睛、脑电波…


Siri,
why don't you
understand me?


amazon echo


●●●○○ 中国移动 4G
11:19
我要去
查地图
路线规划
车生活
插件
我的

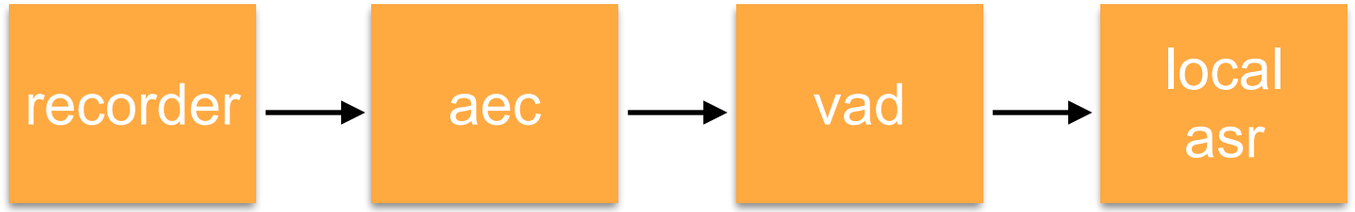

●●●○○ 中国移动 4G
11:18
宝贝
月饼礼盒
取消
热门搜索
货架
cfcdk
历史搜索
你可能要搜索的照片：
天天 天气 不说 我在 高腰 大码
你好你好你好
蜗牛 倒膜
按住 说出你要的宝贝
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M
123
空格
搜索

语音交互的工程技术现阶段还是比较复杂的

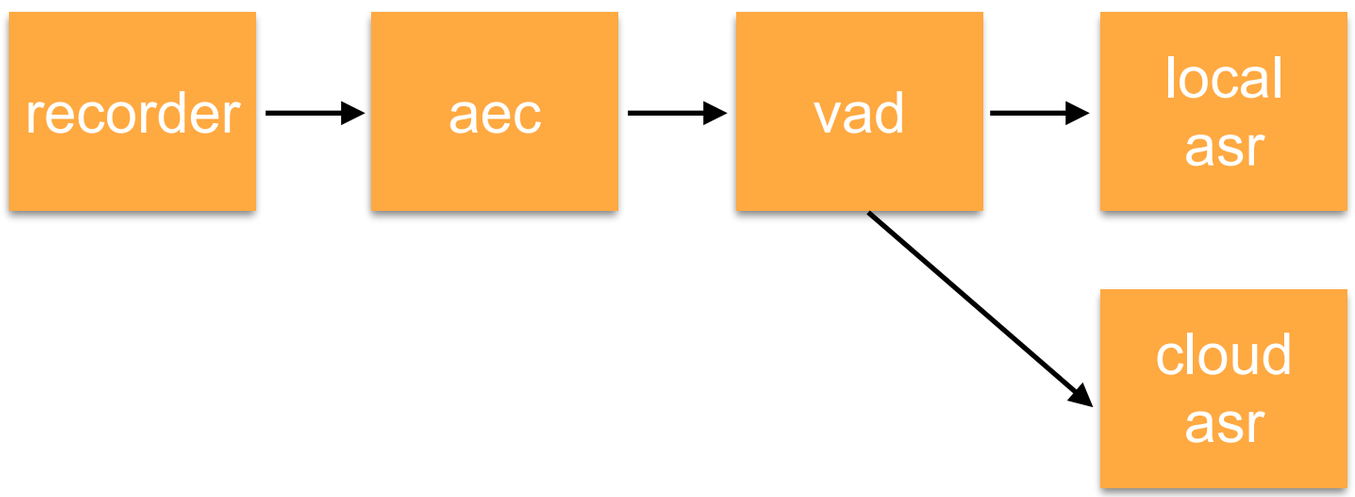
数据流复杂

休眠状态

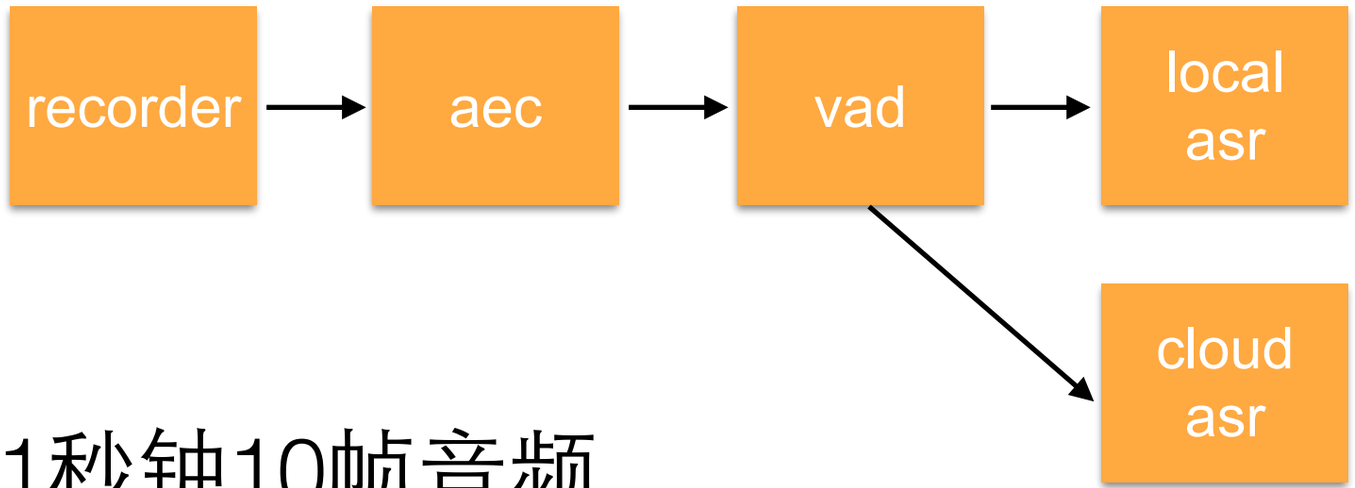
唤醒后本地识别



唤醒后混合识别



多模块实时处理



1秒钟10帧音频
本地各组件实时率 0.1~0.5
识别延时300ms
交互延时1s

不同技术平台支持

Android
iOS
OpenWrt
Linux
Windows
…

各种SDK
精通各种开发语言
熟悉各种技术栈
开发难度大
开发效率低
可移植性差
交互逻辑多份实现

比如在Android上

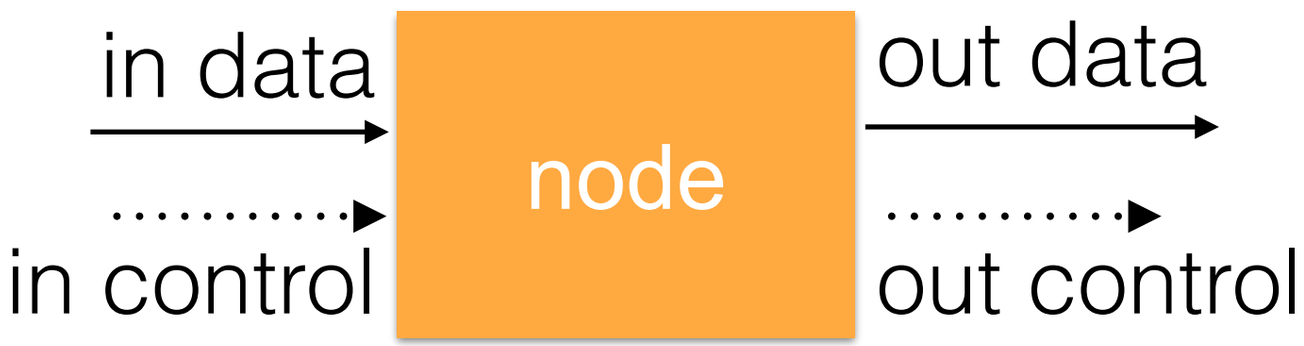
C写所有算法和交互逻辑
JNI绑定后提供android sdk
在android java层再做一些集成

C里面改个逻辑得重新编译so -> 重新编译sdk -> 重新编译apk -> adb install -> 运行调试

写的代码不能运行在其它平台上

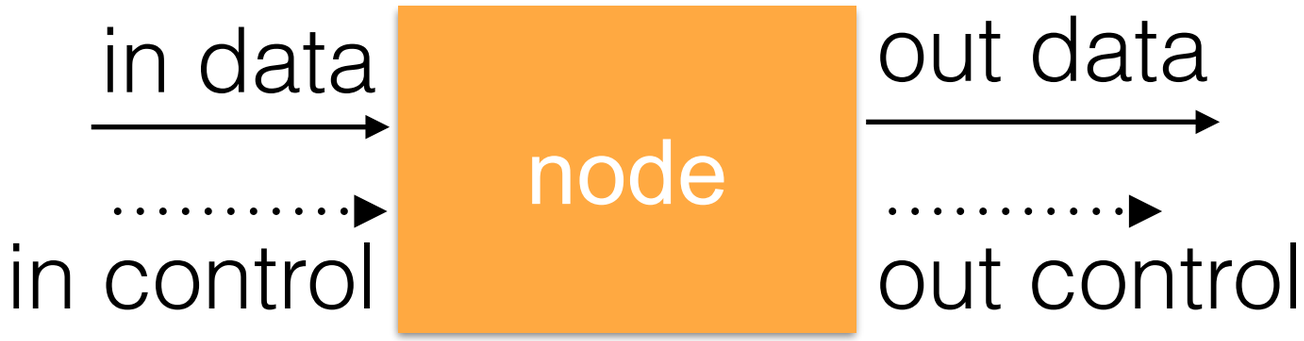
怎样简单 — 抽象

来自DirectShow, ROS的启发



节点

怎样简单 — 抽象



两种通信方式:
* rpc - control command
* message - pub-sub queue

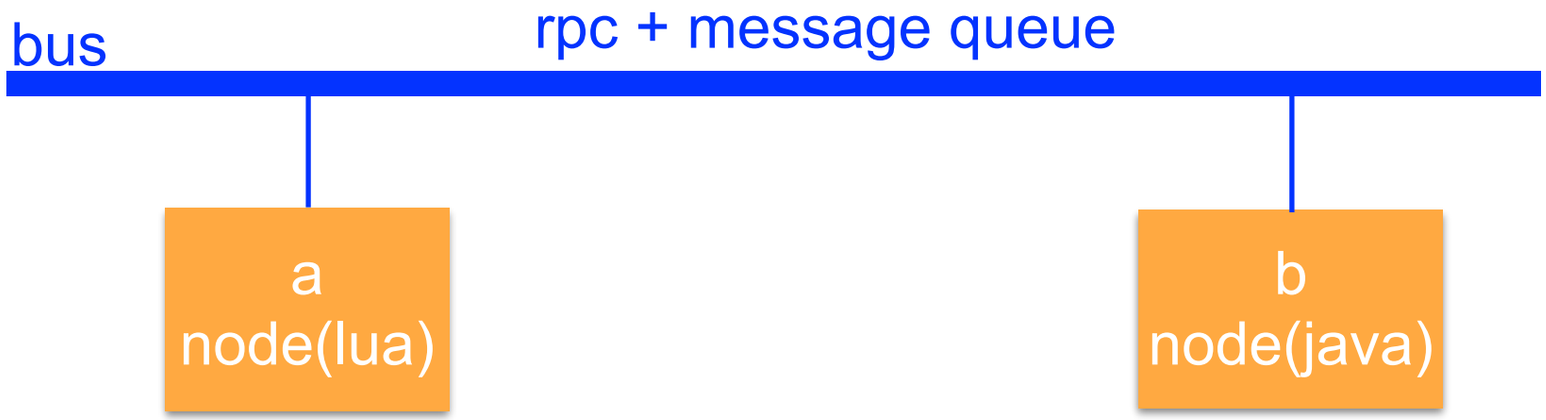
怎样简单 - lua

整体框架用lua编写
跨平台
开发效率更高
用luajit执行效率有保障

怎样简单 - socket

跨平台
可移植性最好的进程间通信方案
各节点提供网络接口
不需要编程语言绑定
分布式
像服务端编程那样

怎样简单 - bus




```
function a:onmessage(topic, …)
    print(topic, …)
end
busclient:subscribe(‘b.output’)
busclient:call(‘/b/start’)
```

```
public RPCResult onCall(url, byte[]... args)
    if url == ‘/b/start’ then
        busclient.publish(‘b.output’, ‘hello’)
    end
end
```

实现

* bus server
* bus client
* busclient.lua
* busclient.java
* busclient.c

* bus server监听端口
* bus client连接到server
* 每个node对应一个bus client
* bus client运行在独立的线程
  * 有独立的事件循环

实现

* 实现了兼容lua-nginx-module接口子集的运行时
  * cosocket
  * timer
  * ...
* 为什么不直接使用OpenResty?
  * 虽然stream-lua-nginx-module支持tcp server
  * 多线程多node与nginx单进程模型设计相背, 比如全局变量

代码复用

* OpenResty
  * lua-resty-websocket
  * lua-resty-http
  * lua-resty-dns
  * ...
* 公司内
  * 基于OR的模块
  * lua封装的c库

全栈?

出现了跨语音云服务开发和语音交互系统的优秀工程师

其它一些便利

开发效率提升

* 核心算法和交互逻辑跨平台
  * linux: ./bootloader.sh /path/aios.bin
  * android: Bootloader.start(“/path/aios.bin”)
  * iOS: [Bootlader start:blabla)
* 直接在pc上开发交互逻辑, 远程调试
  * bus server及核心节点运行在pc上
  * 系统相关节点运行在设备上, 连接到bus server
  * 修改代码直接restart, 设备端app无需重新build

其它一些便利

开发效率提升

* bus上有完整的交互日志
  * 通过replay bus日志可完整复现用户使用过程
  * 通过profile bus日志, 可以发现交互系统中的性能瓶颈

```
$atk prof -i 2786 logcat-verbose-example1.log
= vad,player =
2786       80.709394543         98.58554             17.876145  vad,player      xfunction  user_wait_when_interact
 |- 2676       80.723680927         97.349232313         16.625551  cloud           function   cloud_wait    58007e4733279305a900000a
 |   \- asr_wait 0.269 delaytime 0.156 sestime 8.153 wavetime 2.926 systime 1.609 posttime 0.032 rectime 0.949 vadtime 2.926
 |   \- nlu_wait 0.199 systime 0.19
 |   \- net_wait 16.1576
 |- 1296       80.727401235         80.805894081         0.078493   asr             function   local_asr_wait 58007e4733279305a900000a
 |- 2706       97.439795851         97.449775544         0.00998    processor       call       /recorder/stop
 |- 2709       97.474039467         97.476877698         0.002838   processor       call       /keys/wakeup/words/majorpy   get
 |- 2723       97.49287639          97.504653236         0.011777   processor       function   localfeed    sds local feed
 |--- miss delay 0.139916
 |- 2729       97.644569159         98.018966313         0.374397   processor       function   data2py_hz2wpynew
 |- 2729       97.644569159         98.018966313         0.374397   processor       function   data2py_hz2wpynew
 |- 2751       98.194510082         98.219922159         0.025412   processor       function   segment_and_hz2py
 |- 2750       98.194533082         98.219258698         0.024726   processor       function   data2py_hz2wpyfeed
 |- 2784       98.312472005         98.583139544         0.270668   tts             function   local_tts_wait_first_frame 找到15个徐家汇相关地点
```

app layer

music

weather

stock

chat

…

voice layer

processor:
control logic

bus

recorder node

player node

aec node

cloud node

wakeup node

vad node

local asr node

tts node

… node

System layer

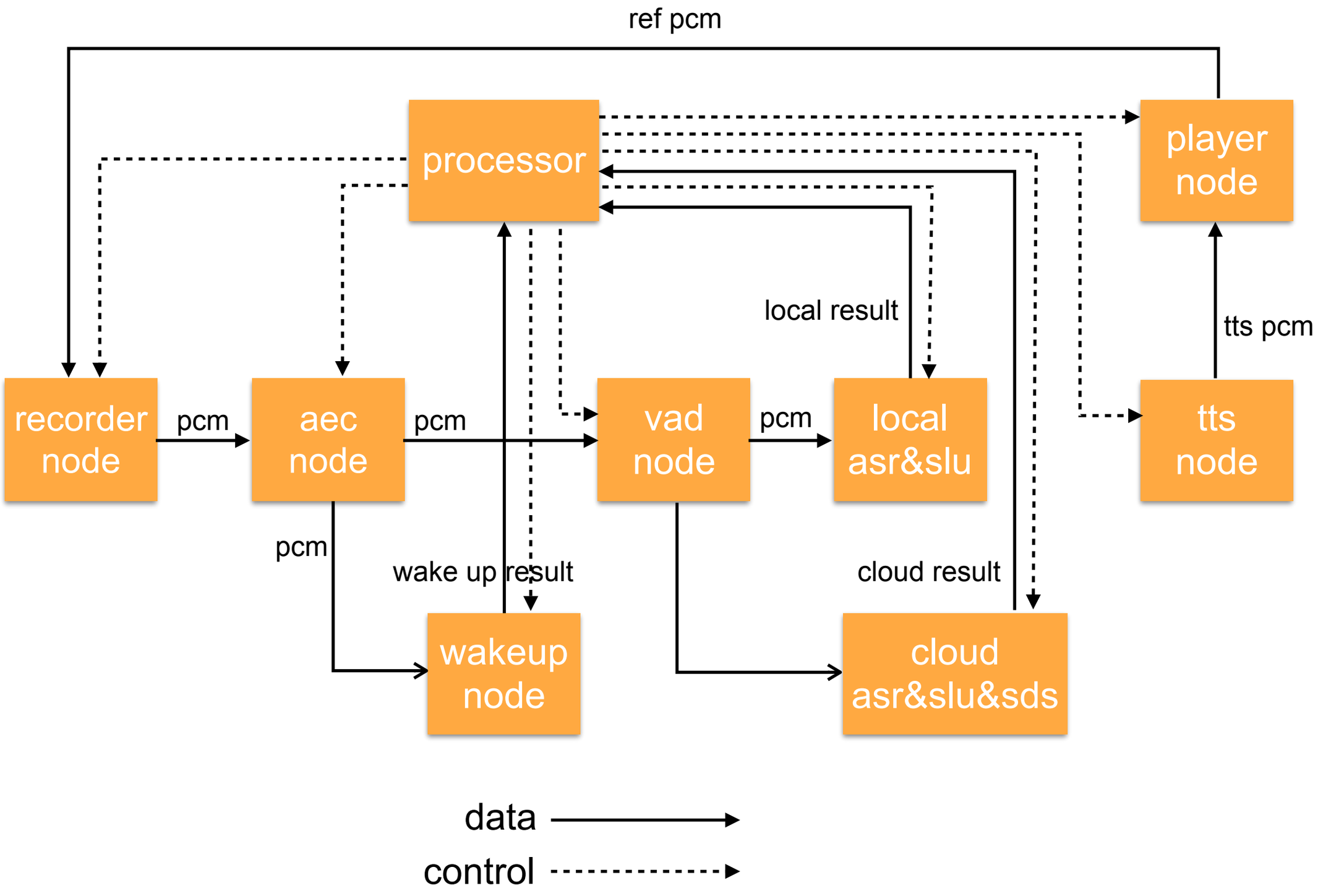
Linux

Android

QNX

OpenWrt

…


ref pcm
processor
player node
local result
tts pcm
recorder node
pcm
aec node
pcm
vad node
pcm
local asr&slu
tts node
pcm
wake up result
cloud result
wakeup node
cloud asr&slu&sds
data
control

演示/视频 - 车载后视镜

https://v.qq.com/x/page/q033669epi4.html

演示/视频 - 语音交互机器人

http://v.youku.com/v_show/id_XMTY4ODc5NDI0OA==.html?from=s1.8-1-1.2&spm=a2h0k.8191407.0.0

未来会有libresty吗?

Q&A