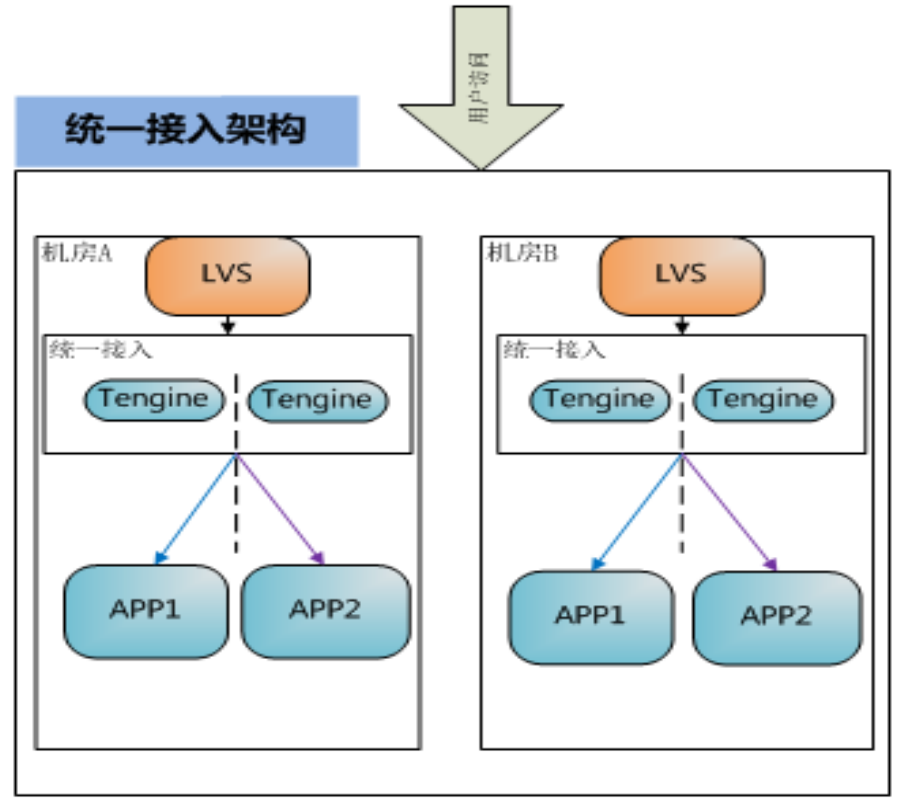
阿里七层流量入口
Tengine硬件加速探索之路

王发康（毅松）　https://github.com/wangfakang

前言

Tengine作为阿里集团七层流量入口核心系统，支撑着阿里巴巴双11等大促活动平稳度过，并提供智能的流量转发策略、HTTPS加速、安全防攻击、链路追踪等众多高级特性，秉着软硬件结合的性能优化思想，2017年接入层在硬件加速领域迈出了第一步。



大纲

contents

01 性能分析与调研

02 加速方案实施

03 加速效果

04 展望及总结

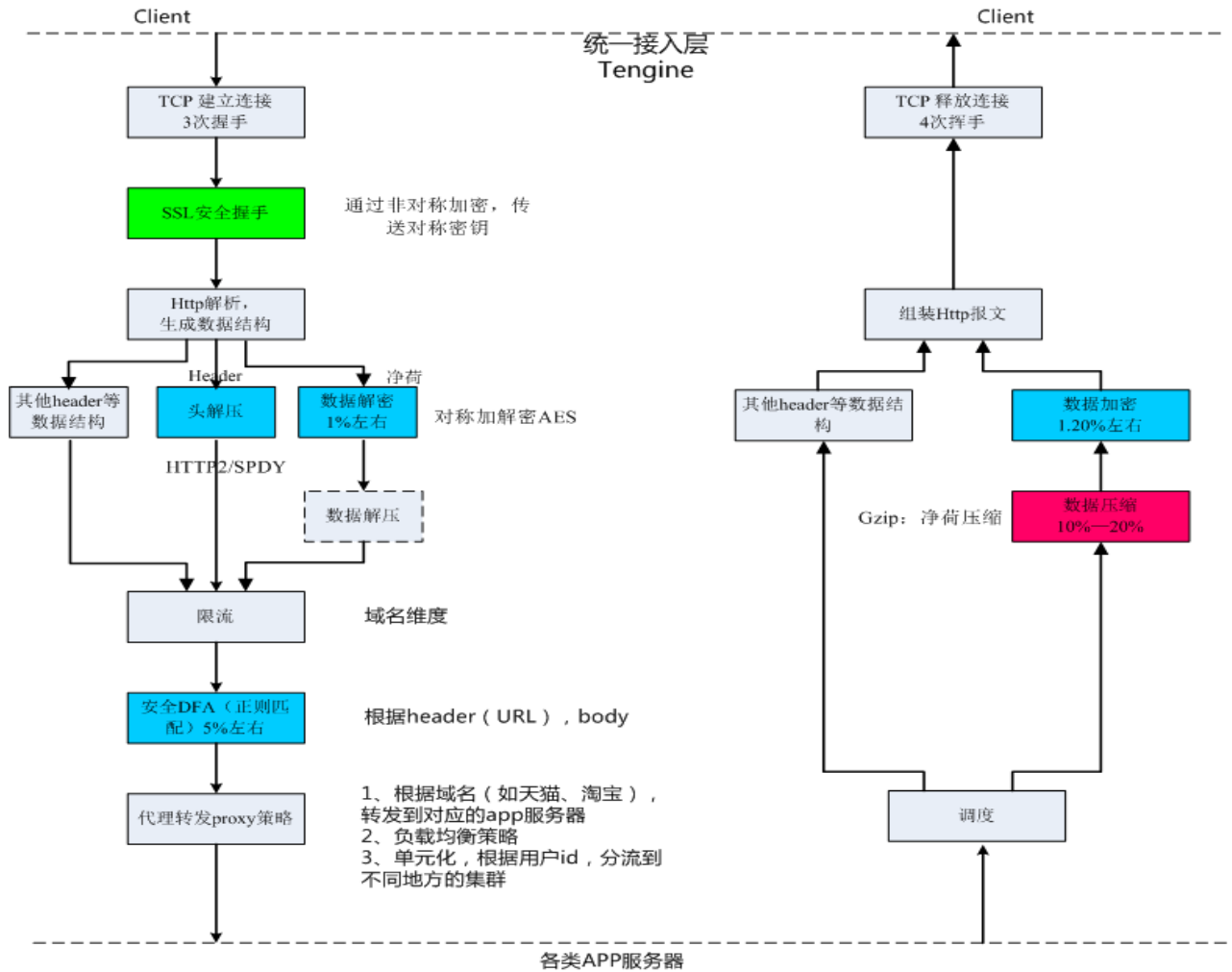
01

性能分析与调研

- 性能分析
- 业内解决方案
- 方案选取

Tengine性能分析

模块名称	功能介绍	CPU占比
Gzip	解压缩	15%-20%
Sinfo	流量镜像	10%
Limite flow	限流	8%
Lua	lua相关业务	8%
Security	防火墙	6%
其它众多模块	...	...



业内解决方案

- SSL硬件加速
  - 腾讯
  - 百度
  - CloudFlare
- Gzip硬件加速
  - 无已知

注： 图片来源 https://blog.cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details/

Tengine硬件加速——HTTPS卸载篇

HTTPS现状简介

虽然全站HTTPS已经是一个老生常谈的话题，但是国内为何能做到的网站却还是屈指可数？原因简单总结来说有两点，首先使用HTTPS后使得网站访问速度变**“慢”**，其次导致服务器CPU消耗变高、从而机器成本变**“贵”**。

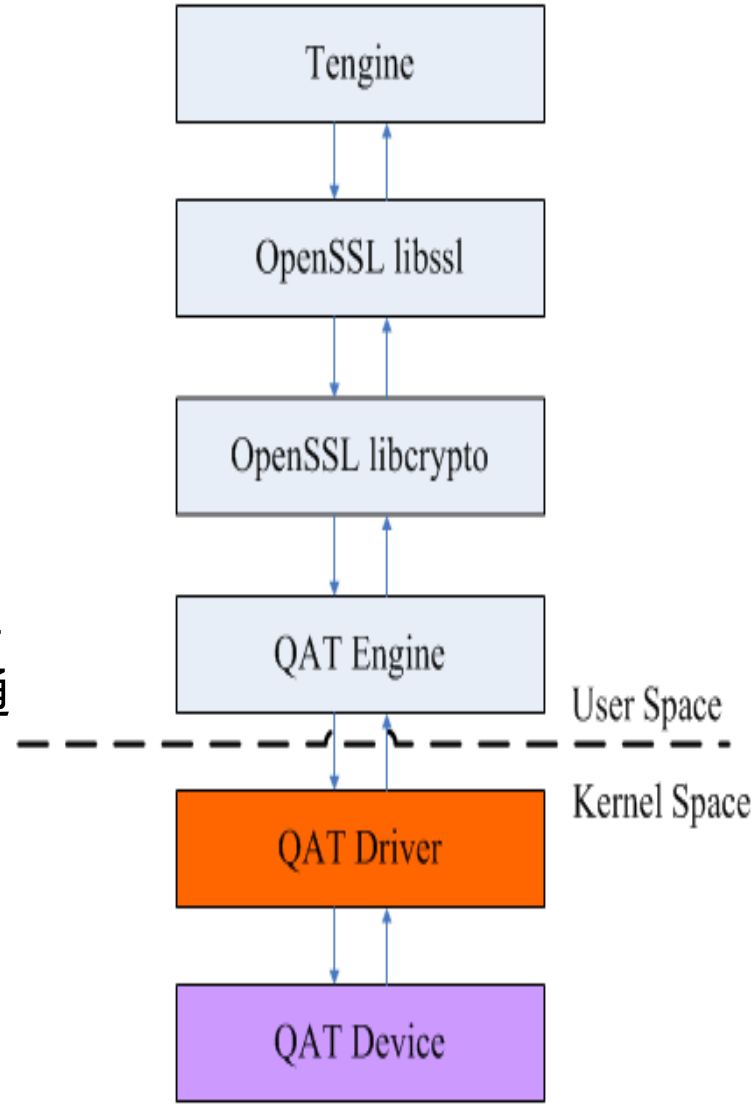
软件优化方案：如Session复用、OCSP Stapling、False Start、dynamic record size、TLS1.3、HSTS等。

但软件层面如何优化也无法满足流量日益增长的速度，加上CPU摩尔定律已入暮年，使得专用硬件卸载CPU密集型运算成为业界一个通用解决方案。

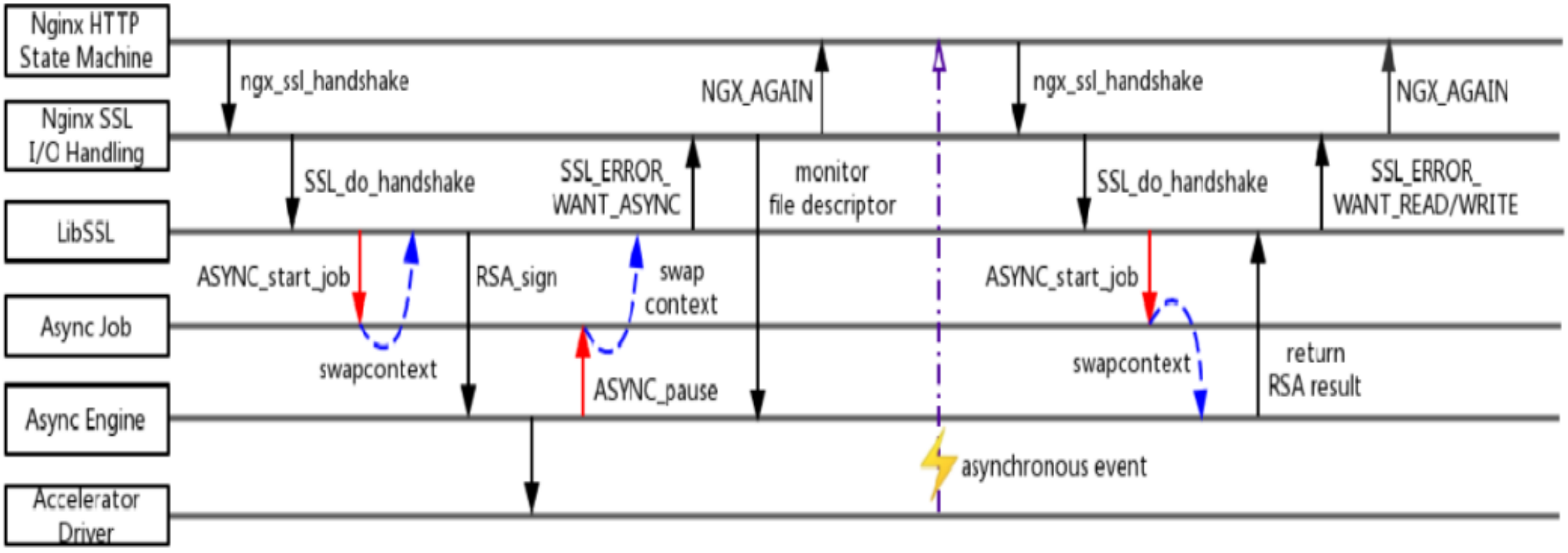
HTTPS硬件加速方案

Tengine基于Intel QAT的异步加速方案，其总体框架如右图所示。由三部分组成Tengine的ssl_async指令、OpenSSL + QAT Engine及QAT Driver。

其中Tengine通过适配OpenSSL-1.1.0的异步接口，将私钥操作卸载至Intel提供的引擎(QAT engine)中，引擎通过QAT驱动调用硬件完成非对称算法取回结果。



HTTPS硬件加速原理介绍



注：图片来源Intel® QuickAssist Technology (QAT)

HTTPS硬件加速性能对比

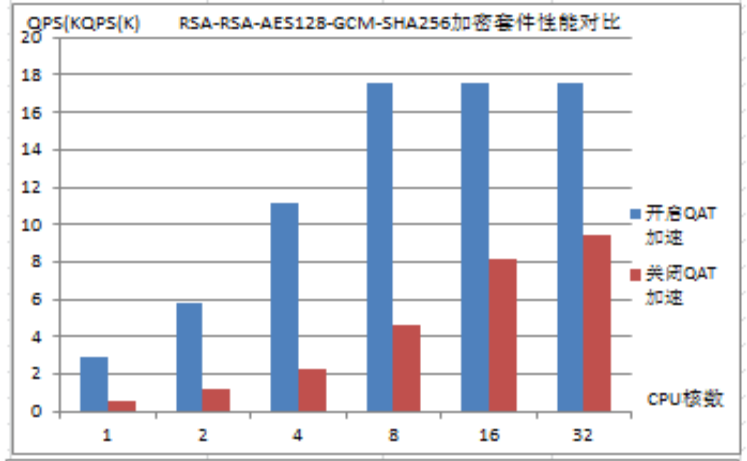
Tengine启用ssl_async QAT加速：

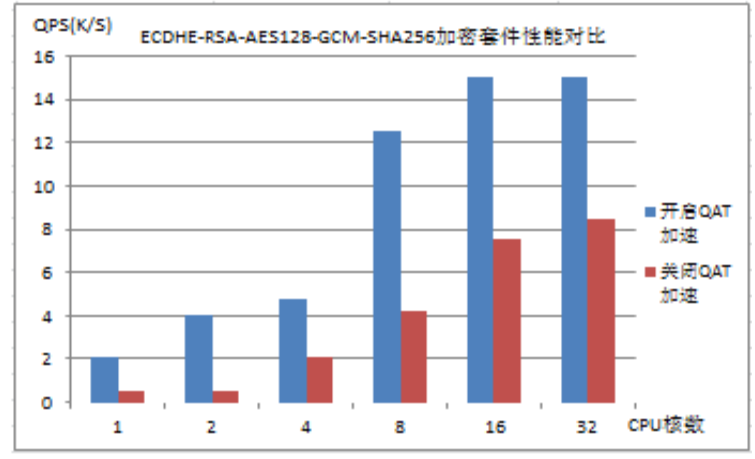
RSA套件提升**3.8**倍(8核)

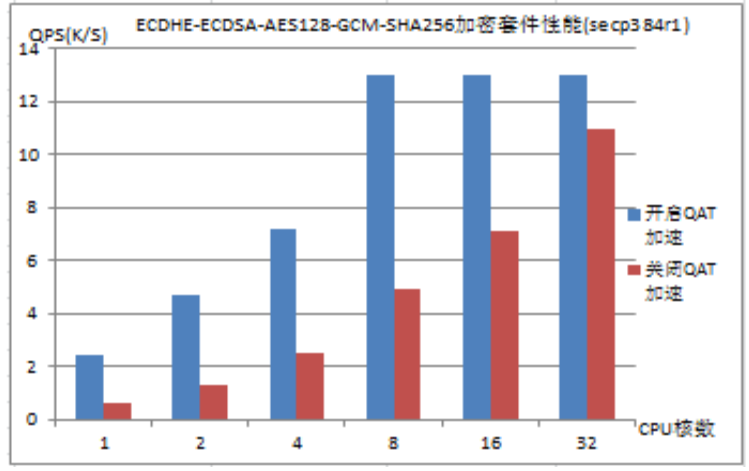
ECDHE-RSA提升2.65倍(8核)

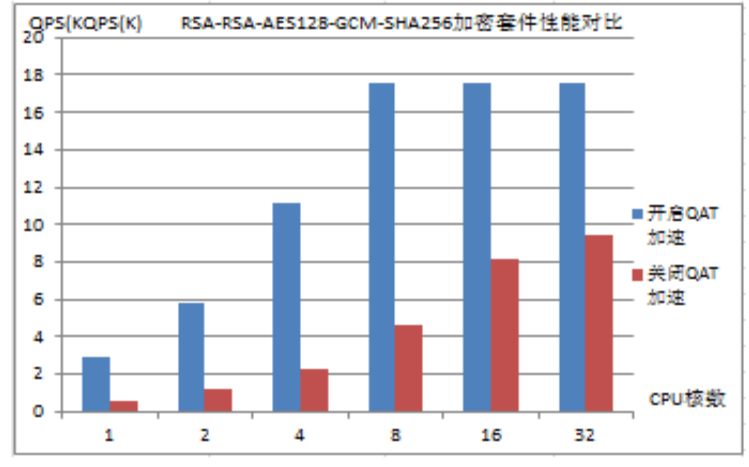
ECDHE-ECDSA(P-384) 提升**2**倍(16核)

ECDHE-ECDSA(P-256) 8核达到QAT硬件处理峰值16k左右，只有**23%**的性能提升









Tengine硬件加速——GZIP卸载篇

方案对比及选择

Intel QAT卡（1）

ASIC模式

应用场景

压缩及解压缩

SSL数据加解密

实现

QAT卡提供zlib加速算法

评估

zlib shim适配度好

QAT性价比高

智能网卡（2）

实现

提供压缩API给host，压缩数据返回host，由host封包发送（同QAT）

约定压缩flag，host发送未压缩报文，网卡收到后压缩、业务处理、封包发送

评估

实现复杂

仅offload，并无加速

性价比不高

FPGA卡（3）

成本相对高

开发成本

资源成本

性能没有ASIC模式高

评估

无开发资源支持

02

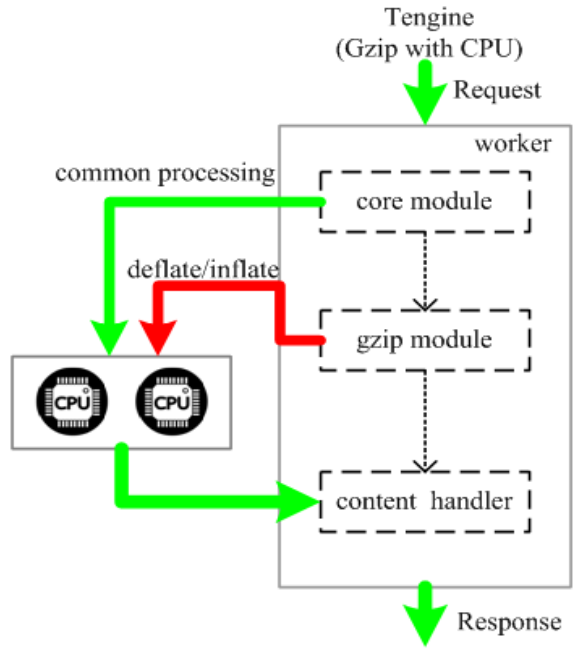
加速方案实施

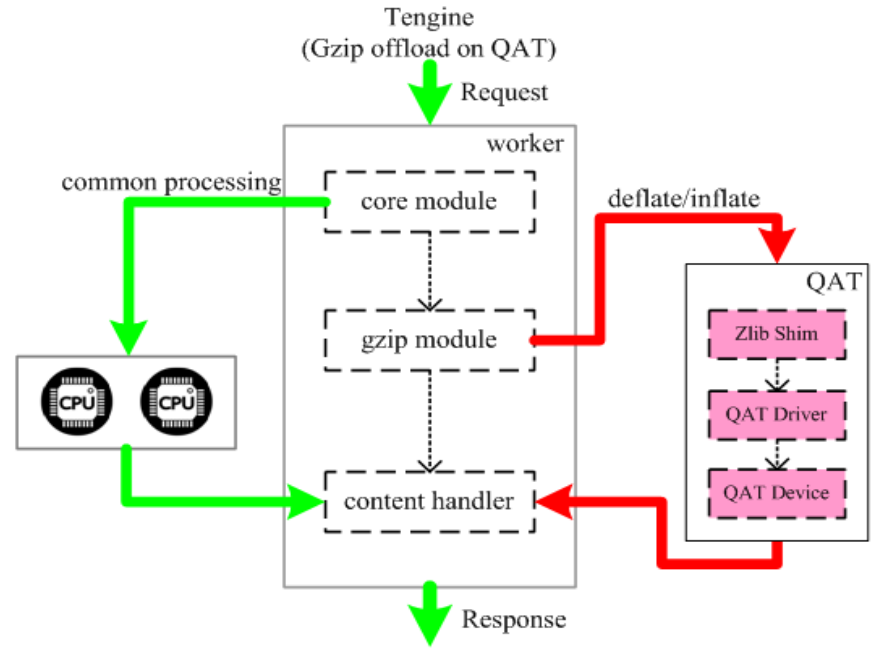
- Gzip加速架构
- 遇到的问题
- 如何解决

Tengine Gzip 硬件硬件加速架构

Tengine上层通过Zlib Shim适配标准zlib库，然后通过QAT Driver提供的相关接口打通Zlib与底层硬件的通信。

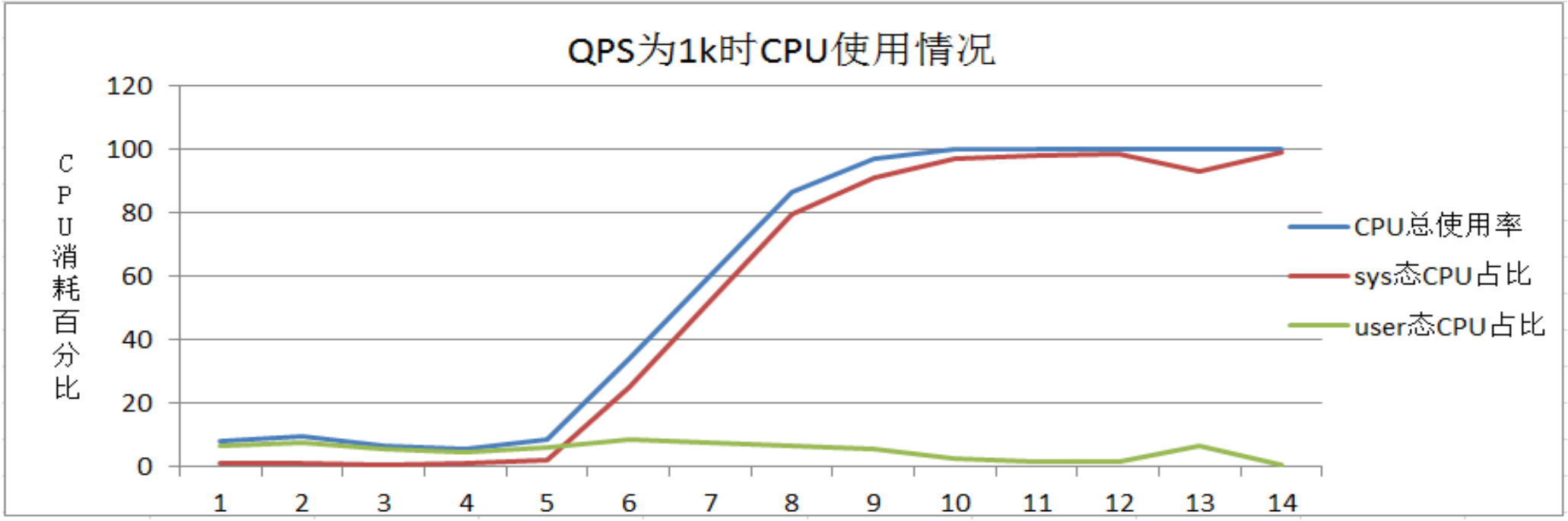
QAT加速卡也顺应了虚拟化的潮流，其采用SRIOV技术，可在虚拟机之间高效共享PCIe设备，当前DH895XCC系列芯片最高可支持32个虚拟机共享QAT，从而达到充分利用硬件资源。





遇到的问题

使用的第一版驱动Intel-Qat2.6.0-60，当QPS为1k左右时CPU很快打满（注：正常情况下QPS为1k时，CPU消耗6%左右），且CPU消耗中90%以上都是消耗在内核态。



问题分析

```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 99.88   38.202119       10256      3725           ioctl
  0.10    0.036755           8      4812           epoll_wait
  0.00    0.001465           0     23217           epoll_ctl
  0.00    0.001323           0      6001        20 write
  0.00    0.001015           0      2565           writev
  0.00    0.000903           0      4312           close
  0.00    0.000780           3       271           mmap
  0.00    0.000520           0      2640        35 open
  0.00    0.000496           0      4281      1817 read
```

函数名	DSO	热度	总体	调出
compact_zone	[kernel]	88.096%	93.525%	5.430%
_spin_unlock_irqrestore	[kernel]	4.744%	4.744%	0.000%
json_parser_string	tengine	0.377%	0.562%	0.185%
re2::DFA::AddToQueue	libwaf.so	0.377%	0.403%	0.026%
finish_task_switch	[kernel]	0.311%	0.311%	0.000%

然而并不是那么顺利

测试发现CPU节省效果不佳(用户态CPU降低和增加的内核态CPU相当)，发现使用QAT后，部分系统函数CPU占比变高，如下图所示(注：左边的是使用QAT后各系统热点函数)open、ioctl、futex执行时间占比高达8.95(注：3.91 + 2.68 + 2.36)，而未使用版本对应占比时间才0.44(注：0.24 + 0.14 + 0.06)。

% time	seconds	usecs/call	calls	errors	syscall
25.25	0.082034	2	40936		epoll_wait
11.86	0.038518	1	55330	23974	read
11.58	0.037625	0	136519	486	write
8.87	0.028823	0	98213		close
6.86	0.022274	0	191799		epoll_ctl
6.66	0.021646	0	117704	26842	recvfrom
5.89	0.019120	1	30988	8	writev
3.91	0.012713	0	56529	6	open
3.77	0.012242	0	26626	55	sendto
2.86	0.009279	0	35863	16362	readv
2.72	0.008837	0	19002	19002	connect
2.68	0.008721	0	74858		ioctl
2.36	0.007683	0	15879	399	futex
1.78	0.005786	0	19002		socket
1.73	0.005625	0	22519		accept4
0.54	0.001760	0	22813		getsockopt
0.46	0.001494	0	16726		getsockname
0.18	0.000586	0	6018		setsockopt
0.02	0.000054	0	667		fstat

% time	seconds	usecs/call	calls	errors	syscall
27.18	0.057429	1	39629		epoll_wait
15.21	0.032140	1	49525	21799	read
13.11	0.027713	0	126857	1123	write
7.69	0.016259	0	184384		epoll_ctl
7.23	0.015276	0	37320		close
5.83	0.012319	0	28561	4	writev
5.66	0.011950	0	102401	23787	recvfrom
5.25	0.011099	0	24754	398	sendto
4.20	0.008881	0	36951	16154	readv
3.00	0.006334	0	17844	17844	connect
2.14	0.004520	0	17844		socket
1.44	0.003041	0	18878		accept4
0.70	0.001474	0	21008		getsockopt
0.53	0.001114	0	15833		getsockname
0.36	0.000770	0	5339		setsockopt
0.24	0.000507	0	17844		ioctl
0.14	0.000292	1	575	7	open
0.06	0.000124	0	668	130	futex
0.02	0.000040	0	568		fstat

一路走来，通过无数次的性能优化、功能测试，多次同Intel研发同学一起探讨之后，才使得QAT在功能、性能、架构方面等众多问题得以快速解决

运维与监控

压测与演练

- 高流量压测
- 压缩与解压缩并存
- 数据完整性检查

容灾保护

- 硬件资源缺乏自动切换软件
- 资源恢复后自动切回

可维护与监控

- 对资源指标进行实时监控及报警

部署与发布

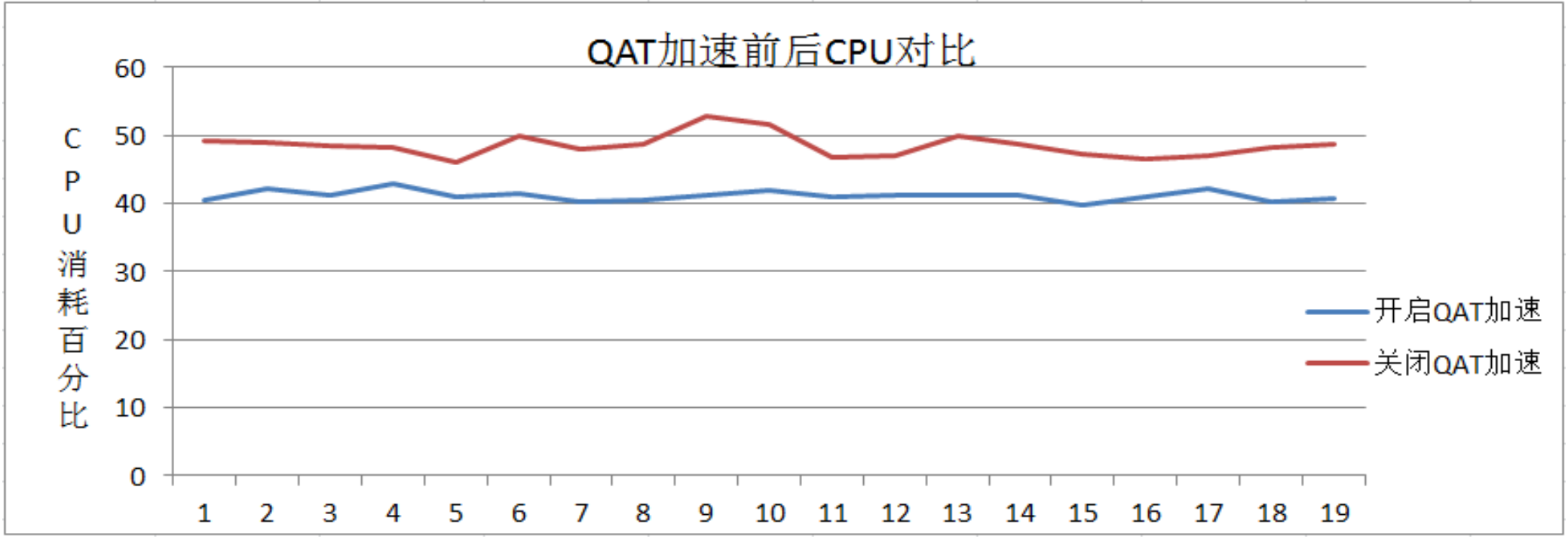
- 单rpm包、双二进制模式
- 自动识别

03

Tengine Gzip硬件加速效果

Tengine Gzip QAT 加速前后系统CPU对比

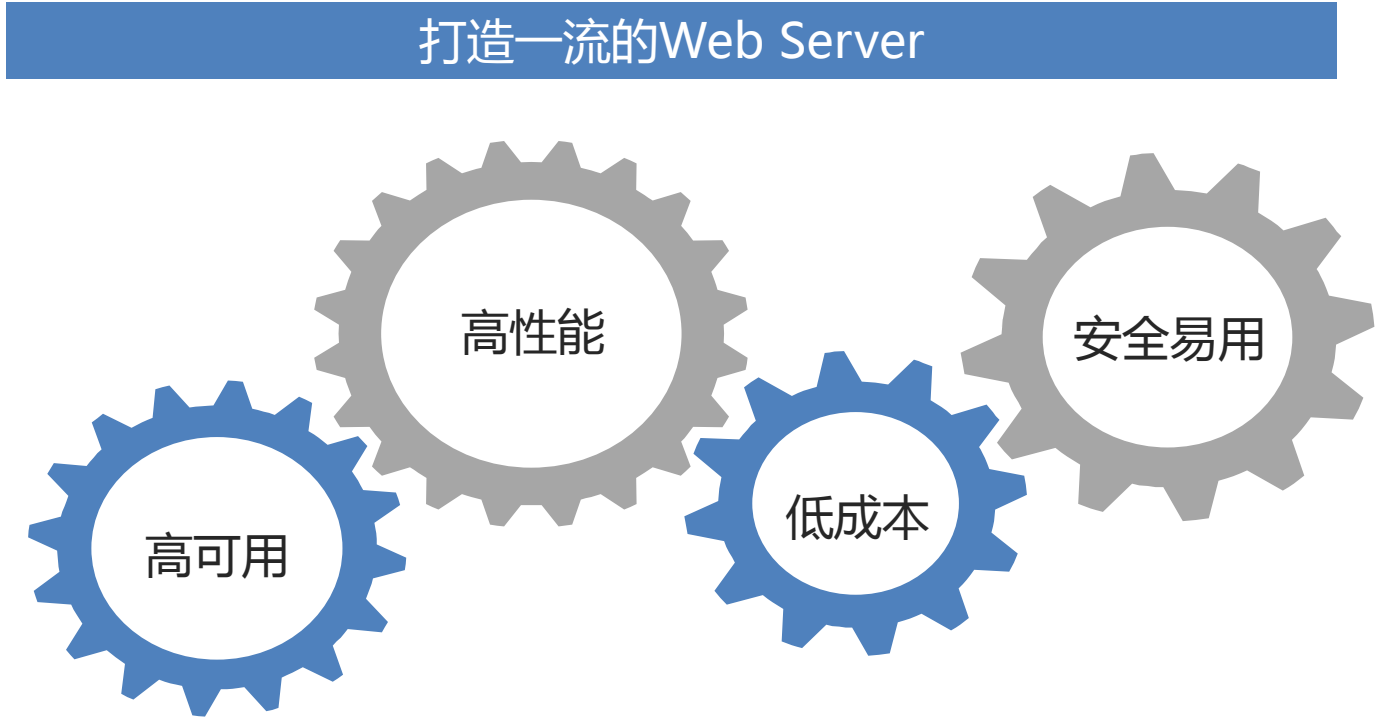
当qps为10k左右时Tengine Gzip使用QAT加速后CPU节省15%左右，且Gzip基本上完全卸载、随着其占比变高，优化效果将越好。（在2017年双11零点流量峰值，Tengine加速机器相比普通机器性能提升 **21%。** ）



04

展望及总结

展望及总结



Tengine首次采用硬件加速技术卸载Gzip，不仅带来性能上的提升，而且使得接入层在硬件加速领域再次打下了坚实的基础、为后续SSL+Gzip架构上整合做好了沉淀，同时也填充了业内接入层对Gzip采用硬件加速的空白，对此领域的发展具有一定的影响意义。

Tengine官网： http://tengine.taobao.org

Tengine Github： https://github.com/alibaba/tengine

Tengine Gzip 硬件加速： https://mp.weixin.qq.com/s/v_BsfSFp-LwiCOFKaMIS_g

Tengine Async OpenSSL 硬件加速： https://mp.weixin.qq.com/s/5OZOURttVdP8ye2lBXwwzg

Q?A:E