intel XEON

白皮书

基于英特尔® 至强® 可扩展平台，多重优化方案助力阿里巴巴 Noslate 性能加速

阿里云

“随着更多基于 Node.js 的业务应用被部署到云原生、Serverless 等新一代云服务架构中，阿里巴巴也在积极探索对 Node.js 的优化改进并推出了 Noslate 产品。为了让 Noslate 获得更优性能，我们与英特尔一起基于第四代英特尔® 至强® 可扩展处理器等产品开展了一系列针对性的优化，并取得了显著的性能提升。”

李三红
程序语言与编译器技术总监
阿里云

前言概述

活跃在服务端任务处理的 Node.js[1] 是阿里巴巴各业务应用运行时的基石之一，为了在云原生 (Cloud Native)、无服务架构 (Serverless) 等新一代云服务架构下进一步向用户提供更优的服务体验，阿里巴巴基于 Node.js 推出面向 Serverless 架构和云原生场景的 JavaScript 容器方案 Noslate。

作为阿里巴巴多年的深度合作伙伴，英特尔提供的包括第四代英特尔® 至强® 可扩展处理器在内的先进全栈软硬件产品与技术，一直是阿里云构建强劲云基础设施的重要基石。为了让 Noslate 在基于英特尔® 架构的云平台上获得更好的运行性能和更短的启动时间，英特尔与阿里巴巴一起，携手打造全新的多重优化方案，并在后续的验证测试中取得了预期的效果。

方案背景

在阿里巴巴围绕消费、云计算和全球化三大战略[2] 构建的商业版图中，无数的互联网应用正以高效、便捷的服务能力，为人们的生活出行、企业的生产经营带来良好体验。这其中，基于 JavaScript 引擎构建的 Node.js 是阿里巴巴应用服务端最重要的运行框架之一。据统计，自 2018 年以来，Node.js 在阿里巴巴的使用量平均年增长约为 37%，而部署到 FaaS (Function as a Service，函数即服务) 平台的活跃应用程序量平均年增长更达到 140%[3]。因此对 Node.js 开展有针对性的优化，一直是阿里巴巴推动服务品质，提升用户满意度的重要保证。

随着云原生、Serverless 等新一代云服务架构获得越来越多的部署，为使各类应用在新架构中具有更好的性能与弹性，进而满足业务在泛终端、全栈交付等领域的需求，阿里巴巴也借助长期的技术积累与创新探索，在 Node.js 的基础上打造了 Noslate 这一面向 Serverless 架构和云原生场景的 JavaScript 容器方案。

作为一款高效、弹性和完全可定制的 Serverless 运行框架，Noslate 可为云上各类工作负载提供大量的高级功能和优化组件，例如以下组件：

- **Node.js 发行版：** 即阿里巴巴 Node.js 发行版 Anode，其面向函数计算冷启动场景开展了优化，可降低 Node.js 用户代码加载耗时；
- **Noslate Workers：** 一种全新的轻量化容器方案，能有效平衡交付灵活度、资源和执行效率，从而高效应用于中心化的 SSR（Server-Side Render，服务端渲染）等轻量任务场景；
- **Noslate Debugger：** 具有基于 Corefile 的“快照”功能，可有效对问题进行离线诊断的工具，助力用户实现问题回溯和实时定位。

性能的持续优化与应用模式的不断成熟，让 Noslate 在阿里巴巴众多业务场景中承担越来越重要的角色，并逐渐成为主流的服务端负载之一。例如在某重要的全球购物节上，其已托管了数万个面向客户的服务器端渲染页面，在提升购物体验方面起到了关键作用。

而包括一系列英特尔® 至强® 可扩展处理器在内的英特尔先进产品与技术，一直以来都是阿里巴巴旗下阿里云的重要性能引擎。随着第四代英特尔® 至强® 可扩展处理器被引入阿里云最新第八代企业级弹性计算实例规格族 ECS g8i 中，在实例性能相比上一代提升 60% 之余[4]，阿里巴巴也希望通过与英特尔合作，借助一系列针对性的优化方案来提升 Noslate 在新一代云实例上的性能，从而让部署在阿里云上的众多应用从中受益。

借助英特尔在 Node.js 上的优化经验，双方携手制订的优化方案将围绕基于英特尔® 架构的平台及相关软硬件技术展开。后续的验证测试数据表明，优化方案能在真实工作负载下，令 Noslate 执行性能获得显著提升，启动时间得以大幅降低。

优化方案

基于英特尔® 架构平台和 Noslate 的优化方案可分为 11 个优化项，其中优化项 1、2 是针对英特尔® 架构平台所具备的向量化硬件特性而专门设计，其它优化项则可用于英特尔及其它平台。但一些测试同样表明，这些优化项在基于英特尔® 架构的平台上有着更优的表现。

优化一：利用异步 Nginx 优化加速加解密性能

在信息隐私愈发受到重视的今天，互联网应用普遍都采用 HTTPS（Hypertext Transfer Protocol Secure，超文本传输安全协议）等更安全的网络协议来巩固数据安全。但这也带来了巨大的加解密计算工作量，进而对应用整体性能带来影响。英特尔® QAT（英特尔® QuickAssist Technology）面向 Nginx 加入了专门的优化，令其可工作在异步模式下，即应用无需等待加解密任务完成就能够继续下一步工作，这显然可以大幅提升应用中 HTTPS 通信的性能。值得一提的是，由于英特尔® QAT 能良好地工作在软件模式下，因此 Noslate 不需要加入专门的英特尔® QAT 驱动程序来实现此优化。在面向阿里巴巴真实工作负载的测试中，这一优化能帮助 ghost.js 提升其 HTTPS 工作负载的 TPS（Transactions Per Second，每秒事物处理量）性能 15% 以上[5]。

优化二：利用 SIMD 指令提升缓冲区换位操作的性能

与 Node.js 一样，在 Noslate 等构建的服务端应用中，未被应用处理的二进制数据会被暂时存放在缓冲区（Buffer）中，因此 Noslate 也定义了许多缓冲区交换 API 函数来对这些数据进行操作。例如需对缓冲区中的数据进行字节顺序调整时，就会用到 Buffer.swap16()、Buffer.swap32() 及 Buffer.swap64() 等函数。

传统上，这种交换操作只能采用顺序方式，在数据量较大的情况下显然缺乏效率。而优化项是利用基于英特尔® 架构的处理器所具备的 SIMD（Single Instruction Multiple Data，单指令

多数据流）硬件指令，包括英特尔® AVX-512 等，让上述函数的工作模式从顺序方式变为并行方式，从而有效提升交换效率。在实战中，长度超过 128 字节的缓冲区能通过这一优化项获得更佳优化效果。

优化三：借助透明大页技术实现性能加速

页表是应用中虚拟内存与物理内存相互转换的关键元素，但过大的页表，以及存放访问最频繁页面的 TLB（Transfer Look-aside Table，快表）也会带来过多的系统开销，影响整体性能。这一优化项是通过引入 Linux 系统的 THP（Transparent Huge-page，透明大页）功能，在大页优点（页表更小、页表遍历更快、内存开销更低等）的基础上进一步允许大页做动态分配，综合减少 Noslate 运行时的 TLB 开销。在实战中，需要通过下面的命令手动启用，然后构建二进制文件。

```
1. $./configure –v8-enable-hugepage
2. $make
```

优化四：对新生代空间大小参数调优

同 Node.js 一样，新生代空间（New space）是 Noslate 在内存中存储新对象和实现垃圾回收管理的区域，因此其大小设置也会对性能造成影响。空间设置越大，吞吐量越高，但耗费的内存也越大。方案中经过对比验证，将其设置为 128MB 来实现更优化。

```
1. $node --max_semi_space_size=128 app.js
```

优化五：默认打开 Noslate 压缩指针特性

这一优化项能使得 Noslate 中的堆对象更小，进而占用内存更小，降低缓存和 TLB 的压力，并由此提高性能。在实战中，需要通过下面的命令手动启用，然后构建二进制文件。

```
1. $./configure --experimental-enable-pointer-compression
2. $make
```

优化六：默认打开 Short Built-in calls 特性

这一优化能使 Noslate 应用更好地进行分支预测，通过长程跳跃（long-range jumps）的减少来提高工作负载执行时的 IPC。

优化七：利用 PGO 实现性能加速

这一优化是利用工作负载的 PGO（Profile Guided Optimization，配置文件引导优化）配置信息来重建并优化节点二进制文件。在实战中，是通过先执行一遍应用后收集启动阶段的热点数据并生成缓存文件，在后续的二进制文件构建过程中加载此缓存文件，大幅提升执行速度。

优化八：利用 BOLT 加速性能

这一优化是使用 BOLT（二进制优化和布局工具）对 Noslate 中的 Anode 二进制文件进行二进制优化。

优化九：优化容器的处理器资源分配

这一优化是在 docker 运行命令行中添加下述选项：

```
1. --cpuset-cpus=
```

这可以为部署在容器中的应用加入处理器限制（即为容器中的应用指定相应的处理器核心来保证处理效能）。

优化十：优化 Noslate 主线程的处理器亲和性

这一优化将设置 Noslate 主线程所在处理器核心与当前启动时的处理器核心之间的相关性。这一设置能够减少在高处理器负载情况下，由于线程迁移而导致的 TLB 和缓存未命中情况。

优化十一：利用字节码缓存，减少函数式计算框架的冷启动时间

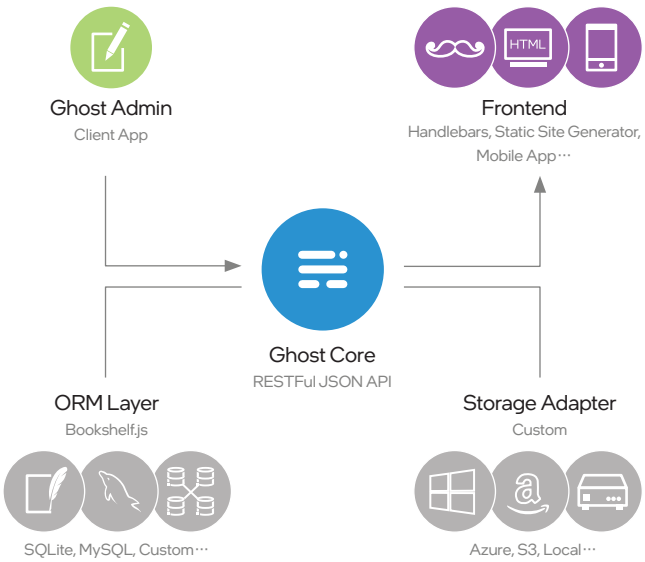
这一优化能够让 Noslate 工作负载在第一次运行时，通过 V8 API 将字节码（bytecode）存储到磁盘中。然后在后续运行中复用这些字节码，从而减少部分解析和编译工作，减少函数式计算框架的冷启动时间。

方案验证

为验证上述 11 个优化项的效果，阿里巴巴与英特尔一起基于第四代英特尔® 至强® 可扩展处理器，在云实例上开展了交叉测试，并覆盖了 Ubuntu、Anolis 等不同的操作系统。测试方案基于英特尔开发的 Node.js 性能测试场景展开，这些测试项都来自现实世界开源的 Node.js 应用代码或 Node.js 官方的性能测试案例。测试场景包括：

1. Ghost.js

Ghost.js (https://ghost.org/) 是一个流行的开源 (MIT 许可证) Headless Node.js CMS (Content Management System，内容管理系统)，可用于内容发布和博客设计。如图一所示，作为一个现代的、解耦的 web 应用程序，Ghost.js 由 Ghost Core (Ghost 核心)、Ghost Admin (Ghost 管理)、Frontend (前端) 等组件构成。



图一 Ghost.js 架构示意图

Ghost.js 工作负载包括一个 Ghost.js 应用服务器 (由 node.js 运行)、Nginx web 服务器、MariaDB 数据库服务器，以及用于压力测试的 ApacheBench 客户端。同时这一工作负载会包括 http 和 https 两种版本。对于 http 工作负载，ApacheBench 使用“http://”作为连接 Nginx 的协议，而 https:// 则用于 https 工作负载。https 工作负载使用 TLSv1.3、TLS_AES_256_GCM_SHA384 加密套件和 secp384r1 ssl_edch_scurve 算法。

工作负载所有的相关组件都会被打包到一个 docker 映像中。因此从客户端测量到的吞吐量能很好地反映出单实例性能。测试中，服务器上将启动多个 docker 实例 (数量通常与服务器逻辑核心相同)，在服务器的处理器负载达到 90% 以上后，测量服务器的每秒吞吐量，所有实例的吞吐量之和就是总性能。(值得注意的是，此配置仅用于简化测试，可能无法反映实际情况)

2. Web Tooling

Web Tooling (https://v8.github.io/web-tooling-benchmark/) 是一个性能测试套件，如图二所示，其关注由 JavaScript 执行的各类常用 web 开发工具的工作负载，并在执行多次后对结果进行评估 (几何平均数方法)。

```
Running Web Tooling Benchmark v0.5.3…
-------------------------------------
        acorn:  5.85 runs/s
        babel:  4.52 runs/s
  babel-minify:  4.48 runs/s
      babylon:  3.75 runs/s
        buble:  2.34 runs/s
         chai:  8.77 runs/s
 coffeescript:  3.30 runs/s
      espree:  1.87 runs/s
     esprima:  2.50 runs/s
       jshint:  4.50 runs/s
        lebab:  4.94 runs/s
     postcss:  2.47 runs/s
     prepack:  3.43 runs/s
    prettier:  3.33 runs/s
  source-map:  4.49 runs/s
      terser:  7.20 runs/s
  typescript:  3.39 runs/s
   uglify-js:  3.38 runs/s
-------------------------------------
Geometric mean:  3.84 runs/s
```

图二 运行 Web tooling 测试基准

在测试中，每个 docker 实例中都有一个 Node.js 进程运行 Web Tooling 基准测试，服务器上将启动多个 docker 实例 (数

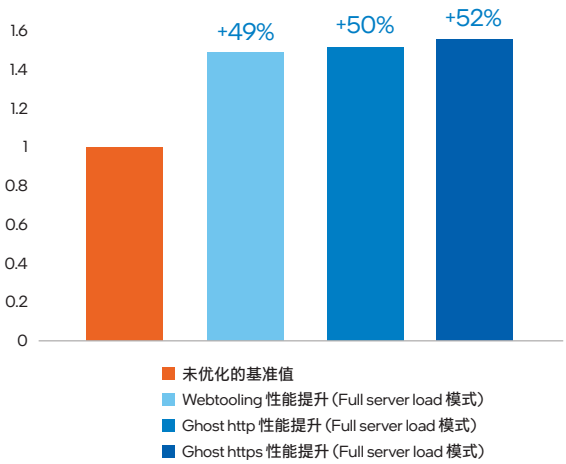
量通常与服务器逻辑核心相同），在服务器的处理器负载达到 90% 以上后，计算每个 docker 实例的几何平均数乘以实例数作为性能结果。

3. Function Computing Cold Start up（函数式计算冷启动时间）

这个测试场景将在 Node.js 上连续启动一个 FC（Function Computing，函数式计算）框架，并在用户函数执行后结束。性能分数定义为启动时间（延迟）的平均值。在测试中，将在服务器上启动多个相同的性能测试容器，当服务器的处理器负载达到 90% 以上后，测量每个容器中函数式计算框架的平均启动时间。（值得注意的是，这一测试场景主要是面向云服务提供商，帮助其优化 FC 框架代码，减少启动延迟）

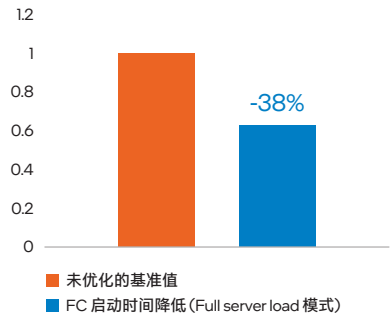
结合以上不同测试场景，在每个工作负载都加入所有可用优化项后，阿里巴巴与英特尔基于阿里云最新的第八代企业级 ECS 实例 g8i（配置第四代英特尔® 至强® 可扩展处理器）开展了测试。测试结果中，Full Server Load 意味着对运行相同工作负载的多个节点实例（基于超线程数的 64 个实例）进行测试。测试结果如下[6]：

■ 吞吐量获益（Ubuntu 操作系统）



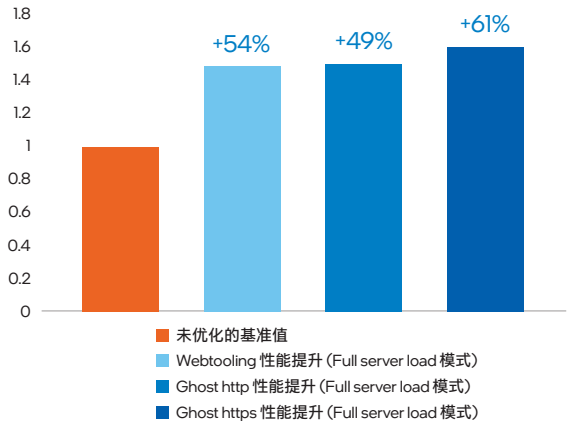
图三 Ubuntu 操作系统中优化方案带来的吞吐量获益（归一化）

■ FC 启动时间降低（Ubuntu 操作系统）



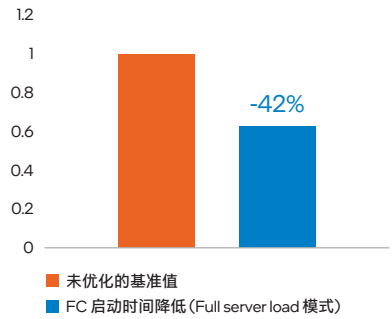
图四 Ubuntu 操作系统中优化方案带来的 FC 启动时间降低（归一化）

■ 吞吐量获益（Anolis 操作系统）



图五 Anolis 操作系统中优化方案带来的吞吐量获益（归一化）

■ FC 启动时间降低（Anolis 操作系统）



图六 Anolis 操作系统中优化方案带来的 FC 启动时间降低（归一化）

综合来看，上述 11 项优化项在基于第四代英特尔® 至强® 可扩展处理器的平台上都能为 Noslate 工作负载带来显著性能提升，在不同的操作系统中，不同测试场景下的吞吐量获益可达 49% 至 61%，而 FC 启动时间也能分别降低 38% 和 42%。

展望

阿里巴巴与英特尔携手开展的优化方案，被证明可帮助 Noslate 有效提升其在云平台，尤其是基于英特尔® 架构的云基础设施上的性能表现，这让阿里云上的 Node.js 开发者能够以更低的成本获得更加稳定可控的 Node.js 运行环境，并使最终用户获得更佳的服务体验。

面向未来，双方计划以龙蜥社区为合作载体，进一步在 Node.js、WebAssembly 等技术领域上，围绕第四代英特尔® 至强® 可扩展平台的特性开展优化，并构建标准化、有代表性的性能测试集来提升真实工业级框架的性能。优化方向包括但不限于：

- 面向英特尔下一代产品架构，进一步优化 Node.js 应用的执行性能；
- 推动 Node.js 持续发展，增强其在 Serverless 等技术领域的应用场景；
- 开源 Node.js 性能测量基准（benchmark），并持续加入更多测试案例；
- 推动龙蜥社区在 NodeJS，WebAssembly 等技术领域的 SIG（Special Interest Group，专项兴趣小组）活动。

[1] Node.js 是一个基于 Chrome V8 JavaScript 引擎构建的，开源、跨平台的 JavaScript 运行框架，可用于各类业务平台服务端构建，详情请见：https://nodejs.org/zh-cn

[2] 详细信息请参见阿里巴巴官网介绍：https://www.alibabagroup.com/about-alibaba-businesses

[3] 数据援引自白皮书《Optimize Node.js for Noslate on Intel platforms》，详情请参阅：https://github.com/noslate-project/node-benchmark/blob/intel-whitepaper/paper/Intel_Optimization_Noslate_SPR_Update.pdf

[4] 数据来源于阿里云，如欲了解更多详情，请联系阿里云：https://www.aliyun.com/

[5] 如欲了解更多性能配置详情，请参阅白皮书《Optimize Node.js for Noslate on Intel platforms》，https://github.com/noslate-project/node-benchmark/blob/intel-whitepaper/paper/Intel_Optimization_Noslate_SPR_Update.pdf

[6] 测试配置，硬件配置：测试平台：阿里云第八代企业级 ECS 实例 g8i；处理器：英特尔® 至强® 铂金 8475B 处理器（32 内核，64 虚拟处理器）；内存：256GB；BIOS 版本：449e491；Microcode：0x1；操作系统：Ubuntu 22.04.1 LTS/Anolis OS release 8.8；Kernel 版本：5.15.0-58-generic for Ubuntu/5.10.134-13.an8.x86_64 for Anolis；Docker 版本：23.0.1。软件配置：Noslate：V16.19.1；Ghost.js workload V4.4.0；Nginx：V1.18.0；Async Nginx *：V0.4.5；QAT Engine *：V0.6.5；OpenSSL Library *：V1.1.1j；IPP Crypto Library *：2020u3；Ipsec Multi-Buffer Library *：V0.55。（其中标*软件仅使用在 Ghost https 测试场景中）