# Practical cloud storage auditing using serverless computing

**摘要**

云存储审计研究致力于解决云端外包存储数据完整性的问题。近年来，研究者们提出了各种不同技术的云存储审计方案。尽管这些研究在理论上很精妙，但它们假设了一个理想的云存储模型，即假设云端能够提供所需的存储和计算接口。然而，这并不适用于主流的云存储系统，因为这些系统只提供读写接口，而不提供计算接口。为了弥补这一差距，本研究提出了一种基于无服务器计算的云存储审计系统，可用于现有的主流云对象存储。该系统利用现有的云存储审计方案作为基本构建模块，并做了两项调整。其一是利用云对象存储的读取接口来支持传统云存储审计方案中的块数据请求。其二是采用无服务器计算范式来支持传统所需的块数据计算。利用无服务器计算的特性，该系统实现了经济、按需的云存储审计。该系统还通过在嵌入认证标签以供后续审计时不修改数据格式的方式，支持主流云存储上层应用（例如文件预览）。我们对提出的系统进行了原型化，并开源到主流云服务——腾讯云。实验结果显示，提出的系统对于实际应用是高效且具有前景的。对于40GB 的数据，审计仅需约 98 秒，经济成本为每年 120.48 人民币，其中无服务器计算仅占 46%。相比之下，目前没有现有研究报告针对真实云服务的云存储审计结果。

**关键词：云存储审计； 无服务器计算； 对象存储； 可用性；**

## 1 引言

近年来，随着 5G 的普及和物联网的快速发展，社会数据总量迅速增加。无论是企业还是个人用户，都存在大量需要计算和存储的数据。云存储服务具有按需付费、易于共享和跨平台访问的优势。此外，云计算具有灵活部署和弹性扩展的优点。这些特点吸引着越来越多的个人、企业和政府机构采用云服务。

尽管云存储和云服务正在迅速发展，但安全问题也随之出现。据报道，云服务时常遭遇数据泄露和服务故障等问题。因此，用户开始担心其外包数据的安全性。为解决云数据安全问题，研究人员提出了云存储审计的概念，并进行了广泛研究。云存储审计解决了远程用户如何确保云端存储数据完整性的问题。这类研究的价值在于，即使云端试图掩盖数据损坏，用户仍然可以利用少量的数据证明来检测云端数据的损坏。

如图 1 所示，经典的云存储审计模型包括两个实体，即用户和云。在将数据外包给云端之前，用户首先使用某些秘密密钥对数据进行预处理。其目的是在外包数据中嵌入秘密认证信息。随后，用户将处理过的数据存储到云端。为验证外包数据的完整性，用户要求云端返回一个证明其真实性的完整性证明。在收到数据证明后，用户通过检查返回的证明/认证是否正确来获取外包数据的最新状态。

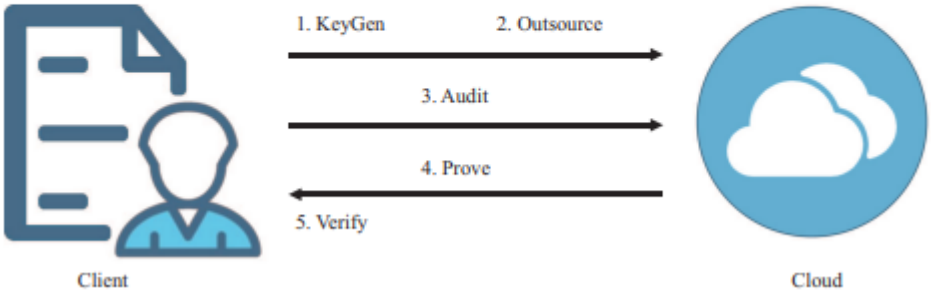


图 1 云存储审计模型

尽管在云存储审计的当前研究中取得了相当大的成就，但大部分研究旨在构建理论方案。这些理论方案假设了一个理想的云服务；即云端为存储提供了读取、写入和计算接口。然而，对于主流的云对象存储（COS）服务，现有理论云存储审计方案所需的计算接口并未提供。因此，对这些方案在当前主流云实践中的实际可用性的研究尚未进行。在实践中，除了架构和实现问题外，经济成本也是影响云存储审计方案实际吸引力的关键因素。

在实际应用方面，无服务器计算发展迅速。与基于传统虚拟机的云计算相比，无服务器计算不需要用户管理云计算基础设施。它只采用按需付费模式，对于用户计算需求较小的情况大大节约了成本。这个特性非常适合云存储审计，并由于其经济吸引力有助于其大规模应用。从研究角度来看，无服务器计算范式为云存储审计研究的原型验证提供了良好的解决方案。从云服务提供商的角度来看，应用云存储审计协议将有助于他们提供更好的云存储服务，并向消费者证明他们的云服务是可靠的。从云用户的角度来看，降低部署审计系统的经济成本和操作成本是有帮助的。

## 2 相关工作

### 2.1 云存储审计

在 2007 年，Juels 等人和 Ateniese 等人首次提出并解决了云存储审计问题。Juels 等人提出了“可检索性证明”方法，简称为 PoR。PoR 首先将数据分成块，并在原始数据块之间随机插入称为哨兵的秘密块。随后，PoR 将数据与哨兵一起混洗并上传到云端。在进行审计时，要求云端返回指定的数据块。可以以概率的方式证明 PoR 方案的安全性；其安全性基于秘密哨兵及其插入位置。

同时，Ateniese 提出了“可证明数据拥有权”方法，称为 PDP。与 PoR 类似，PDP 首先将数据分成块，并为每个数据块计算认证标签。与 PoR 使用随机插入和置换不同，PDP 中用户上传原始数据块和认证标签到云存储。进行审计时，用户要求云端以规定的方式计算一个聚合挑战数据块和一个认证标签。收到聚合证明数据后，用户使用类似于计算认证块的方法来验证外包存储是否完整。PDP 方案的安全性基于密码同态签名或同态消息认证的安全性。

在 Juels 和 Ateniese 的工作之后，研究人员根据不同的场景和需求扩展了云存储审计研究。首个跟进工作是支持数据动态性。PDP 和 PoR 的一个显著缺点是它们仅支持静态数据。无

论是 PDP 还是 PoR 解决方案，大多数云存储审计方案都将数据分成块。支持数据动态性的难点在于以下事实：为抵御重放攻击，认证信息必须与数据块的索引绑定，例如在 PDP 中计算认证标签和在 PoR 中的秘密块的位置信息。当数据动态变化时，基于原始索引的认证信息将失效并需要重新计算。

为解决这一问题，研究人员提出了各种方法。一个经典解决方案是使用“跳表”。它引入了“排名”概念，而不是直接使用数据块的索引以抵御重放攻击。同时，“排名”也包含数据块间相互位置的逻辑信息。在更新数据时，可以以较低的开销维护数据块之间的关系。另一个解决方案是使用类似 Merkle 哈希树（MHT）的认证数据结构。MHT 最初由 Merkle 提出；其结构特征被广泛用于结果验证。在这个方向上，Wang 等人提出了一种基于 MHT 认证结构的云存储方案。当数据更新时，服务器重新计算 MHT 树的根值，客户端验证旧值。同样，根值包含数据块之间索引的逻辑信息。如果验证通过，客户端接受新的根值并签名。除了这两种方法，数据动态性也可以通过建立动态维护数据块索引变化的表来实现。一般而言，这种方法更有效，但需要更多的存储开销；现有研究中也采用了这种方法。

随后，随着云服务应用规模的扩大，研究人员研究了支持公共审计和隐私保护的云存储审计。通过引入更有能力的第三方，可以更有效地处理繁重的审计任务，并减轻用户的审计负担。通常，这样的实体被称为可信第三方（TPA）。许多支持公共审计的 TPA 解决方案基于密码同态技术。在同态消息认证解决方案中，数据所有者请求 TPA 协助完成完整性审计任务，提供关键的元数据信息，包括消息认证标签和对应的公钥。

同态聚合器方法工作原理类似。数据所有者请求 TPA 计算认证信息，让 TPA 完成审计任务。引入第三方带来了好处和新问题；即第三方的安全性使用户数据隐私成为新的挑战。例如，在使用同态技术的方案中，TPA 可以通过对原始块的线性组合来获取原始数据信息。为解决这一问题，研究人员提出了许多基于隐私的公共审计方案。此外，现有研究还利用用户身份简化了存储完整性审计的密钥管理。近期的理论研究也探索了支持云存储审计的新技术和新方法。

总之，现有研究已将云存储审计扩展到支持多功能性，并具有不同的安全基础。然而，这些研究都是基于理论的，因为它们假设云存储服务支持用户定义的计算。但这一假设在实际世界中不成立。现有的主流云存储服务是 COS 服务，它并不提供这样的接口。这导致现有方案在实际应用中效果不佳。此外，现有方案并未考虑用户成本，在实际使用云服务时，用户成本是一个重要的影响因素。在本工作中，我们旨在从实际应用的角度解决这些实际存在的差距。

## 2.2 无服务器计算

近年来，云无服务器计算技术逐渐成熟并得到广泛研究。主流云服务提供商也开始提供无服务器计算服务。 无服务器计算是由事件驱动的托管云计算服务。它具有细粒度的用户管理特性，并根据实际资源使用情况和弹性扩展实现按需计费。这非常适合云存储完整性检查中的挑战-响应审计模式。在部署传统的云存储审计时，需要租用云虚拟机服务器。这种方式被认为很繁琐，因为可能无法充分发挥云服务器的潜力。使用无服务器计算，云服务提供商会根据需要为一次审计任务分配资源，实现按需执行和按需计费。

# 3 本文方法

## 3.1 本文方法概述

Figure 2 展示了基于无服务器计算范式和 COS 的提出的云存储审计系统架构。它由三个主要实体组成，即用户、SCF 和 COS。其中，SCF 和 COS 部署在同一云服务的同一区域。SCF 提供所需的计算服务，可按需计算和灵活扩展。COS 通过其读/写接口使用 HTTP(S)请求/响应提供存储服务。

系统工作如下：首先，用户在本地计算机上使用 KeyGen 算法生成一个秘密密钥。然后，使用 Outsource 算法处理数据。处理后的数据有两部分，即原始数据和每个数据块对应的认证标签。用户随后使用云的 API 将这两部分作为两个独立对象发送到 COS。这样就完成了系统的设置。

之后，用户可以进行定期的存储审计。要发出审计查询，用户执行 Audit 算法并计算挑战，然后使用云 API 将其发送到 SCF。用户根据 SCF 的输入规范，将挑战数据作为 HTTPS 请求的正文内容发送。云无服务器计算 API 网关统一处理接收到的 HTTPS 请求。网关进一步将 HTTPS 请求的内容转换为 SCF 的输入事件，并触发云函数。

一旦触发，SCF 开始执行。首先，SCF 从输入事件中解析挑战数据。随后，SCF 通过 COS API 从云存储下载所需数据块的指定数据和认证标签。之后，SCF 运行 Prove 算法生成审计证明。随后，SCF 将证明以指定的数据格式返回给最初向 SCF 发送审计请求的用户。客户端接收到 SCF 返回的 HTTPS 响应，然后解析审计证明数据。最后，用户执行 Verify 算法检查云数据是否完整。

基于系统架构，我们提供了我们实现的云存储审计系统的更多技术细节。我们选择腾讯云作为代表性的后端云服务提供商。它是亚马逊、Azure 和阿里云等主流云服务提供商之一。它的云服务与其他云服务相似。我们注意到任何主流云都可以使用。

我们使用 Java 实现了所提出系统的原型。实现包含两个可执行程序。一个在用户端。它实现了 KeyGen、Outsource、Audit 和 Verify 算法。另一个在云上，以 SCF 的形式。它实现了 Prove 算法。SCF 在用户触发后执行。为了调用 SCF，我们使用了云 API 网关作为触发器。它首先从用户程序中以 API 调用接收审计挑战数据。然后解析挑战数据并调用 SCF。SCF 继续运行 Prove 算法。我们在 COS 上没有使用任何程序。它被用作提供云存储服务的基础设施，并支持两个可执行程序的数据访问。

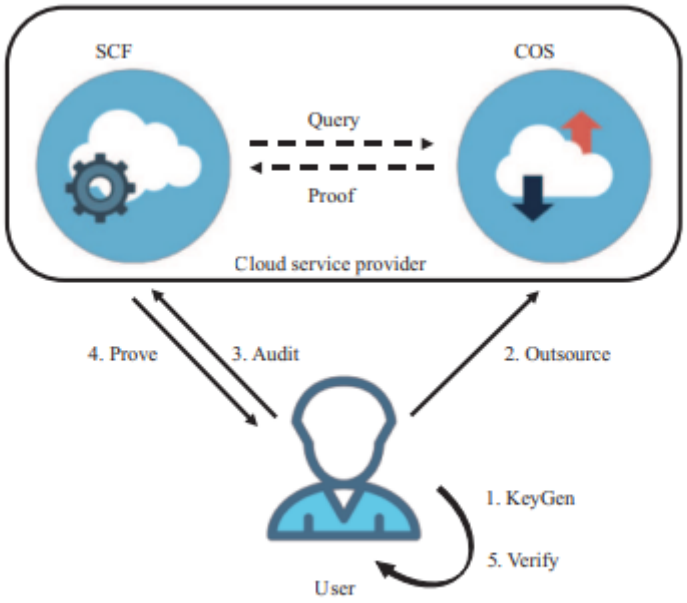


**Figure 2** (Color online) System architecture.

图 2. 系统架构

### 3.2 成本分析

在实际应用中，用户期望了解云存储审计系统的成本。一个价格合适的系统更受欢迎。为了更好地理解成本，我们分析成本组成部分，并根据实际的收费标准确定成本表达式。主要的收费项如表 1 所示：

表 1：

| 符号 | 说明 | 单位 |
|---|---|---|
| pru | SCF 资源使用费 | 人民币/GB·秒/月 |
| piv | SCF 调用费 | 人民币/10000 次请求 |
| ppn | SCF 公共网络下行流量费 | 人民币/GB |
| j | 每天的审计次数 | 次 |
| rm | SCF 运行内存 | GB |
| rt | SCF 运行时间 | 秒 |
| l | 挑战数据长度 | 无单位 |
| ds | 外包数据的大小 | GB |

| | | |
|---|---|---|
| puf | 标准存储使用费 | 人民币/GB/月 |
| prf | 标准读/写请求费 | 人民币/10000 次请求 |
| D | 源数据大小 | GB |
| n | 源数据的块数 | 无单位 |
| λ | 认证标签的大小 | GB |

# 4 复现细节

## 4.1 与已有开源代码对比

由于复现的目的是为了比较不同云的差异，所以第一个差异为复现代码基于 AWS 云的 SDK for JAVA v2 进行实现。

在复现后的测试过程中，发现源代码并不完全支持论文里对数据规模的要求，即在给 jvm 分配 4GB 的内存下，只支持 3GB 左右的源文件进行外包审计，故对源代码进行改进。经过改进，可以支持比源代码大 6.5 倍的源文件进行外包审计。

## 4.2 实验环境搭建

需要在源代码基础上额外引入亚马逊云的依赖，如图 3 所示。当然不建议这么做，最好将需要的各个控件依赖分别导入，具体实现已在复现代码中。

```
<dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>aws-sdk-java</artifactId>
    <version>2.21.22</version>
</dependency>
```

图 3:引入 AWS SDK for JAVA 依赖

运行环境和原文有所不同，原文是 AMD 4800U（1.8 GHz）CPU 和 16 GB 内存。本文是 intel i7-8750H(2.2GHZ)和 16GB 内存。但是原文没有提到给 JVM 分配了多少内存。如果按照 JVM 默认取值，也应该是 4GB 理论分配内存(实际 jvm 可用内存略小)。

## 4.3 创新点

1：原文在腾讯云上实现，完成了基础理论。本文考虑多云环境，在亚马逊云上实现，可以通过本代码在不同云上的表现来比较不同云的性能，是多云环境使用的一个侧面印证

2：原文在代码实现上只支持处理与 JVM 同等大小的源文件外包处理。本文综合考虑运行时间和源文件大小进行改进，可在快于原文 2 倍计算的速度下将源文件的大小提升至 6.5 倍。

# 5 实验结果分析

## 5.1 0-1.5G 计算时间对比

图 4 是原文的计算时间，图 5 是依据原文计算方法仅在不同云环境下执行的计算时间（Verify time 没有画在图中，时间是原文的两倍。代码相同，差异是由于机器的计算性能不如原文所致），可以看到数量级和趋势两者大致相同，尽管本文略快，但是考虑到可能是不同云环境执行时间不同导致。整体趋势和数量级相同说明云环境的性能大致相同且原文和本文都是正确测量的结果。不过值得注意的是，原文中为云环境分配了 1GB 的执行内存，本文只分配了 256MB 的运行时内存。

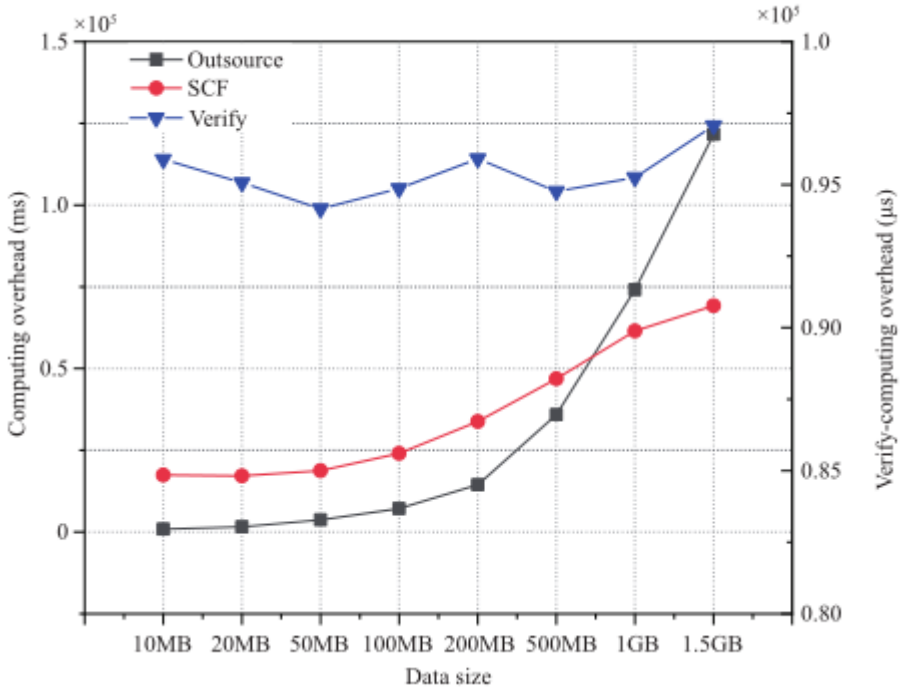


图 4 原文的计算时间

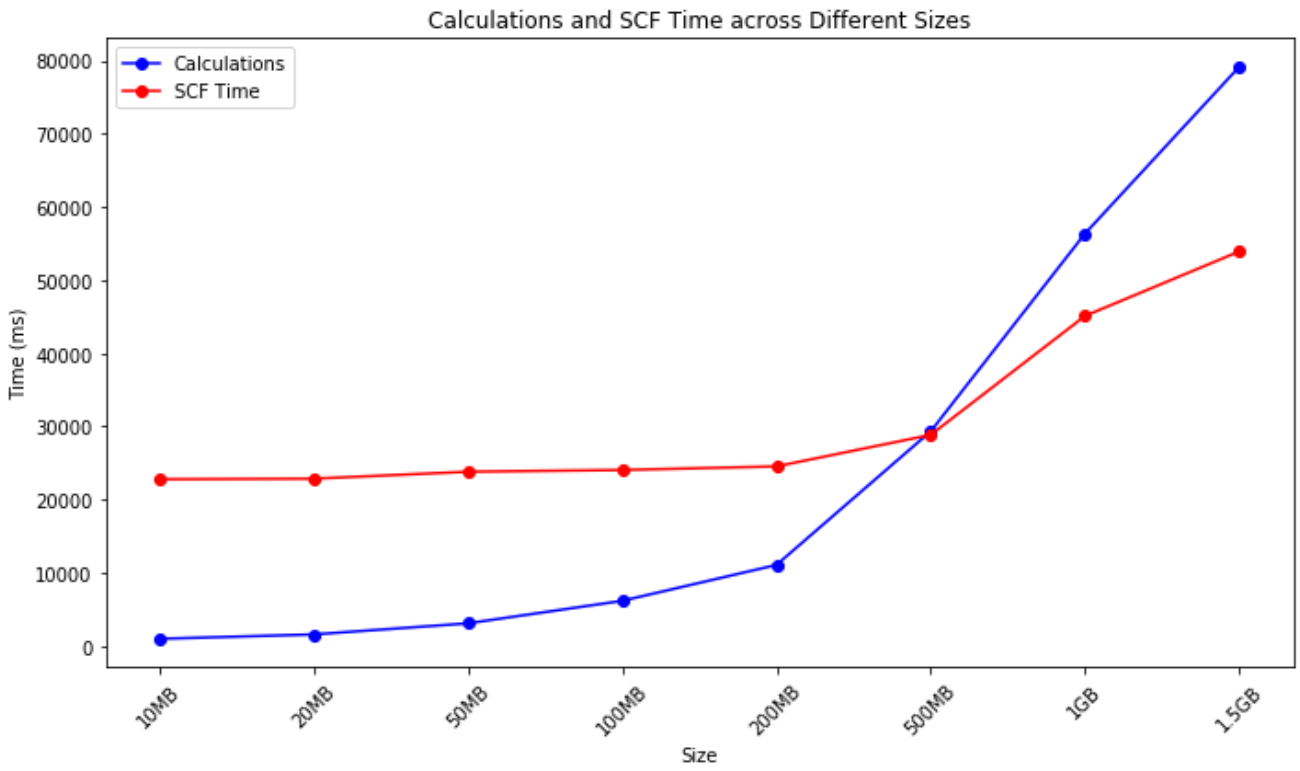


图 5 复现代码计算时间

## 5.2 价格对比

考虑到原文所说的实用性，所以需要对当下 AWS 云和腾讯云，以及当下腾讯云和论文腾讯云的收费价格进行比对。表 2 是对比结果。我们可以得到两个结论

1：地区服务有整体价格趋势。2：但也各有区别

| | **存储¥/G/月** | **读请求/1k** | **写请求/1k** | **SCF 调用/百万** | **SCF 调用/GB*h** | **SCF 流出/GB** |
|---|---|---|---|---|---|---|
| 以前腾讯云 | 0.118 | 0.001 | 0.001 | 1.33 | 0.399888 | 0.8 |
| 现在腾讯云 | 0.118 | 0.001 | 0.001 | 1.33 | 一致 | 0.8 |
| 腾讯(东京) | 0.148 | 0.001 | 0.001 | 1.33 | 一致 | 0.8 |
| 腾讯(孟买) | 0.156 | 0.001 | 0.001 | 1.33 | 一致 | 0.58 |
| 腾讯(香港) | 0.156 | 0.001 | 0.001 | 1.33 | 一致 | 1.0个个个 |
| AWS(东京) | 0.1785个 | 0.0026个 | 0.0336个 | 1.4282个 | 0.4286个 | 0.8141个 |
| AWS(孟买) | 0.1785个 | 0.0029个个 | 0.0357个个 | 1.4282个 | 0.4286个 | 0.7805 |

| | | | | | | |
|---|---|---|---|---|---|---|
| AWS(香港) | 0.1785个 | 0.0029个个 | 0.0357个个 | 1.9995个 | 0.5892个个 | 0.8569个 |

## 5.3 改进代码后的执行效率

| | 10MB | 50MB | 200M | 500M | 1GB | 5GB | 10GB | 20GB | 40GB |
|---|---|---|---|---|---|---|---|---|---|
| 原版 | 1.06s | 3.12 | 12.54 | 31.42 | 61.55 | × | × | × | × |
| 改版 1 | 1.47s | 4.09 | 16.97 | 40.62 | 92.08 | 470.3 | 1096.4 | × | × |
| 改版 2 | 1.12s | 2.81 | 8.99 | 21.26 | 41.41 | 247.6 | 503.6 | × | × |

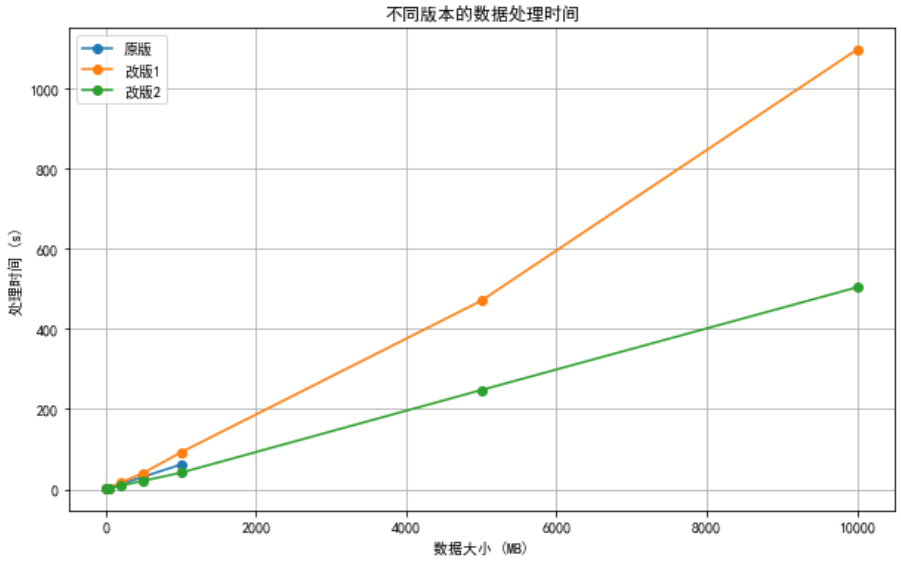


图 6 改版后的处理时间测量结果

## 5.4 改进代码后的瓶颈监测

改版后的代码的瓶颈在于将对原数据的审计计算结果，即额外开销统一读取到内存中，想要解决这个问题，一个容易想到的思路是依次处理。然而，无服务器计算也是用到了这一部分对外包的数据执行计算。这部分留给展望，我们需要先观测瓶颈是否与理论分析一致。根据测量结果我们可以看到，与理论分析一致。在爆栈前一段时间 20GB 源文件下的堆的开销为 2.72GB,与 20GB 文件理论上的开销 2.92GB 相近，所以理论分析正确

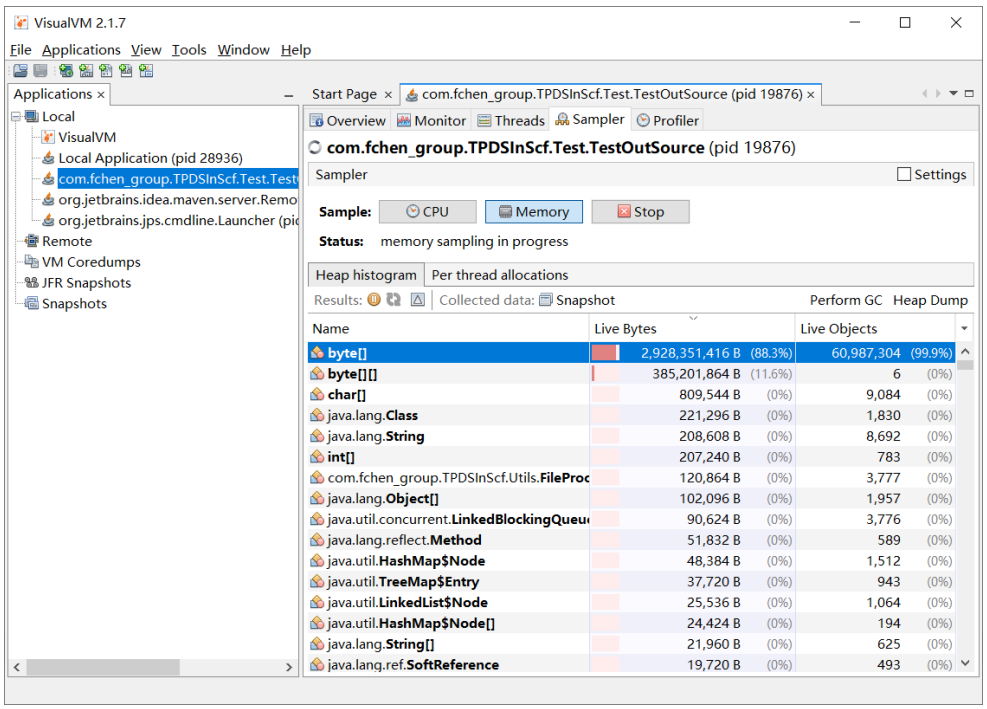


图 7 内存溢出前的监控结果

# 6 总结与展望

为了探究不同云环境下包括性能，价格，功能的不同，以原文入手对实验进行复现。将原文的代码从腾讯云迁移到亚马逊云上并探究表现的不同之处。总的来说，性能趋势相近，相互印证了原文的测量以及本文测量的正确性，但也有所不同，不同之处表现在：

1 价格不同，尽管云服务商在不同地区的收费有地区服务有整体价格趋势，但也各有区别。

2 执行性能不同，亚马逊云的 SCF 执行速度比原文略快

在复现的途中，发现了原文工程上的一个问题，即不能测量大小超过 JVM 内存大小的源文件。对源代码进行了改进，并进行了效率的优化。结果显示，改进后的代码最多可支持 6.5 倍 JVM 内存大小的源文件，同时执行效率为源代码的两倍。

但是在复现时还发现了其他问题，由于时间关系没有及时解决。首先，原文和本次复现都没有提到的上传速度，仅在云存储审计中我们可以忽视这个代价，因为只要云中文件没有损坏，我们只用执行一次上传。但在实际应用中，上传的时间如果过慢是不能接受的，这是云服务的使用体验中需要注意的事情。其二，本文还是只支持 6.5 倍的大小，理论上来说应该能够审计更大的文件才可以，所以需要进一步改进，但是考虑到代码执行效率(SCF 的收费和执行时间及内存开销都有关系)和此次复现工作的剩余时间，只能将这个作为一个展望。

总的来说，本次复现实现了基本目标，我们可以直观的观测到不同云环境下相同功能的表现差异和面临的问题，此次复现可以算作完成任务。

# 参考文献

[1]CHEN F, CAI J, XIANG T, et al. Practical cloud storage auditing using serverless computing[J].

[2]Jonas E, Schleier-Smith J, Sreekanti V, et al. Cloud programming simplified: A berkeley view on serverless computing[J]. arXiv preprint arXiv:1902.03383, 2019.