

# 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的数据完整性问题。近年来，研究者提出了各种基于不同技术的云存储审计方案。尽管这些研究在理论上很高明，但它们通常假设一种理想的云存储模型，即假设云存储能够提供这些方案所需的存储和计算接口。然而，这一假设并不适用于主流云存储系统，因为这些系统通常只提供数据的读写接口，而不提供计算接口。为了弥补这一差距，本文针对现有主流云对象存储提出了一种基于无服务器计算的云存储审计系统。该系统将现有的云存储审计方案作为基础模块，并进行了两项改进：一是利用云对象存储的读接口来支持传统云存储审计方案中的块数据请求；二是采用无服务器计算模式来支持传统方案所需的块数据计算。通过利用无服务器计算的特点，该系统实现了经济实惠的、按需付费的云存储审计。此外，该系统在嵌入认证标签以支持后续审计时，并未修改数据格式，从而支持主流云存储的上层应用（例如文件预览）。本文以主流云服务——腾讯云为平台开发并开源了该系统的原型。实验结果表明，该系统高效且具有实际应用的潜力。对于 40GB 的数据，使用无服务器计算进行审计大约需要 98 秒，经济成本为每年 120.48 元人民币，其中无服务器计算部分仅占 46%。相比之下，现有研究尚未在真实云服务中报告相关的云存储审计结果。

**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，社会数据总量快速增长。云计算因其具备跨平台访问、共享便捷、灵活部署以及弹性扩展等优势，正逐步成为个人、企业和政府组织的首选数据存储和计算解决方案。然而，云存储与云服务的迅速普及也带来了新的安全挑战，用户对外包数据的安全性和完整性产生了广泛关注。

为了解决云数据的安全问题，研究人员提出了云存储审计的概念，并对其进行了广泛的研究。云存储审计旨在解决远程用户如何验证存储在云中的数据是否完整的问题。这类研究的价值在于，即使云服务提供商试图隐瞒数据丢失或损坏，用户仍然可以通过一个小规模的数据证明检测出云端数据的异常状态。然而，目前大多数云存储审计研究主要集中于理论方案的设计，这些方案通常假设云服务能够提供理想化的接口，包括存储数据的读写和计算操作。然而，主流云对象存储服务（COS, Cloud Object Storage）通常仅支持基本的读写接口，而不具备现有理论方案所需的计算能力。这一现实限制了理论方案在实际云环境中的直接应用。此外，经济成本也是影响云存储审计系统实际可行性和大规模应用的重要因素。

相比于传统的基于虚拟机的云计算模式，无服务器计算（Serverless Computing）无需用户管理云计算底层基础设施，并采用按需付费的模式，仅针对实际使用的资源收取费用。这种特性使其在在计算需求较低或不均匀的应用场景中显著降低了用户的经济成本，非常适合云存储审计的应用需求。无服务器计算的经济性和灵活性为云存储审计的实践化和大规模推广提供了重要的技术支持。

基于此，本文 [1] 设计并实现了一个基于无服务器计算的云存储审计系统。该系统以黑盒方式构建于现有的云存储审计协议之上，旨在推动云存储审计从理论研究向实际应用的转化。本文在腾讯云的对象存储（COS）和无服务器云函数（SCF，Serverless Cloud Function）环境中实现并开源该系统。由于主流云服务提供商（如 AWS、Azure、阿里云等）普遍提供标准化且相似的云服务（例如云对象存储和无服务器云函数），因此该系统具备在其他云平台扩展、复现的可行性。

在理论层面，本文拓展并丰富了当前云存储审计研究内容。一方面提出了当前理论研究需要解决的问题，另一方面提出了审计成本模型，有助于云存储审计协议的最优参数设置；在实践层面，随着《数据安全法》和《个人信息保护法》的实施，数据安全问题成为业界关注的重要问题之一，本文所提出的方法可以提高企业云中数据管理的合规性。

在此次研究中，我们将该云存储审计系统在微软云（Azure）上成功复现，并对系统的实际性能进行了分析。本研究的目标是为推动云存储审计的实际应用提供实用性参考和技术路径。

## 2 相关工作

### 2.1 云存储审计

云存储审计致力于解决云端外包存储的数据完整性问题，已有许多研究提出了多种方案和技术支持。在本节中，将介绍云存储审计的经典模型，以及两个开创性的云存储审计理论方案，最后介绍基于这两个理论基础的后续扩展与改进。

#### 2.1.1 云存储审计模型

如图1所示，经典的云存储审计模型包含两个实体：用户和云。在数据外包前，用户首先使用密钥对数据进行预处理。这样做的目的是在外包数据中嵌入秘密验证信息。完成处理后，用户将数据存储在云端。

为验证外包数据的完整性，用户生成审计请求并发送给云端，要求其返回验证外包数据的完整性证明。云端根据审计请求生成证明后返回给用户，用户通过验证该证明来确认外包数据是否保持完整性。

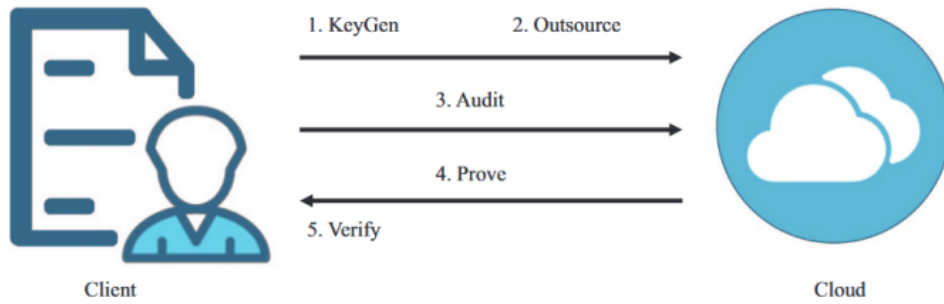


图 1. 云存储审计模型

### 2.1.2 早期云存储审计理论研究

云存储审计的核心目标是通过密码学技术验证存储数据的完整性，而无需将数据从云端下载回本地。早期的研究包括 Juels [2] 等人提出的可检索性证明 (Proofs of Retrievability, PoR) 和 Ateniese [3] 等人提出的可证明的数据持有性 (Provable Data Possession, PDP)，它们是现有云存储审计研究的理论基础：

- PoR (可检索性证明): PoR 将数据分割为多个数据块，并随机插入特殊的“哨兵块” (Sentinels)，然后将这些数据块随机打乱顺序后上传到云存储。在审计过程中，用户向云端发起挑战，要求返回指定的块数据，通过验证哨兵块的位置和内容，用户可以概率性地证明数据的完整性。PoR 的安全性依赖于随机插入的哨兵块及其位置，这种方法具有较高的效率，适合大规模数据存储。
- PDP (可证明的数据持有性): PDP 的工作流程与 PoR 类似，不同的是 PDP 不插入哨兵块，而是直接为每个数据块计算认证标签，然后与数据一同上传到云存储。用户在启动审计时，随机选择多个数据块，要求云端返回这些数据块的聚合块及对应的认证标签，并通过返回的证明数据来验证外包存储是否完好无损。PDP 的安全性依赖于基于同态签名或消息认证的安全性。

### 2.1.3 云存储审计的扩展与改进

在 PoR 和 PDP 的基础上，后续研究针对不同需求和场景提出了诸多改进和扩展方案：

1. 数据动态性支持：传统的 PoR 和 PDP 方案仅支持静态数据存储，不适用于动态数据更新场景。为解决此问题，研究者提出了基于跳跃链表 (Skip List) 和 Merkle 哈希树 (MHT) 的改进方案。这些方法允许用户动态插入、删除或修改数据块，同时维护数据块索引与认证标签之间的绑定关系，从而抵御重放攻击。
2. 可信第三方审计 (TPA)：为减少用户的计算开销，研究者引入了可信第三方 (Trusted Third Party, TPA)，提出由 TPA 代表用户完成数据完整性审计。这种方法通过同态加密技术保证了用户隐私，但也引入了新的安全挑战，例如如何防止 TPA 滥用权力或泄露用户的隐私数据。
3. 隐私保护审计：针对审计流程引入第三方带来的安全性风险，研究者提出了基于隐私的公共审计方案。

尽管这些方案在功能和安全性方面有显著改进，但它们大多仍停留在理论阶段，因为它们假定云存储服务支持用户定义的计算。然而，主流云对象存储服务（如 AWS S3 和 Azure Blob Storage）通常只提供数据的读写接口，而不提供相关计算接口，这一现实限制了许多现有研究方案的实际部署。

## 2.2 无服务器计算

近年来，无服务器计算技术逐渐成熟，并得到了广泛的关注与研究。主流云服务提供商（如 AWS、Azure、阿里云、腾讯云等）也相继推出了无服务器计算服务。无服务器计算是一种事件驱动的托管云计算模式，具备按实际资源使用量计费和弹性伸缩的特点 [4]，特别适合负载不均匀的应用场景。这种特性与云存储完整性检查中的挑战-响应审计模式高度契合。

在传统云存储审计的部署中，用户通常需要租用云虚拟机服务器。这种方法不仅增加了设备资源管理和维护的复杂性，还导致了高额的运行成本，即使虚拟机在大部分时间处于空闲状态。而采用无服务器计算模式，用户仅需专注于审计任务的逻辑实现，由云服务提供商根据具体审计任务动态分配资源，这种按需执行和按需收费的模式有效避免了闲置资源浪费，显著提升了成本效益。

无服务器云函数（SCF, Serverless Cloud Function）是无服务器计算的核心组件，支持根据用户定义的触发条件调用部署的程序。云服务会自动分配所需资源来运行用户程序，无需用户长期维护底层资源。使用 SCF 时，用户只需按照云服务提供商定义的函数接口规范设计和实现程序逻辑，其开发体验与在本地编写程序基本一致。主流云平台支持多种编程语言用于实现云函数，极大地提升了开发的灵活性。

SCF 的编程工作完成后，用户可以将其部署到云端，并通过配置触发器实现云函数的调用。触发器定义了云函数的运行条件，是启动云函数的关键机制。目前，云服务提供商提供了多种触发器选项以满足不同业务场景的需求，如 API 网关触发器、定时触发器以及 COS 触发器。触发器不仅启动云函数的执行，还将事件作为输入参数传递给云函数。事件通常是云服务提供商定义的标准化数据结构，例如 JSON 格式。触发函数时，用户还可以选择同步或异步执行模式。

## 3 本文方法

### 3.1 本文方法概述

在应用层上考虑，本文的想法是以可行、高效和经济的方式将现有的理论云存储审计协议与主流云服务相结合，通过四个方面进行实现。

第一，本文通过使用通用的云对象存储服务（COS）和无服务器云函数（SCF）服务来作为云存储审计协议所需的基础设施。所采用云服务应被现有主流云服务厂商所支持。更详细地说，本文使用 COS 托管用户的外包数据，使用 SCF 为理论云存储审计协议提供计算服务。云对象存储是当今使用最多的云存储服务之一，它已被所有主要云供应商支持。SCF 是一种弹性计算服务，近年来逐渐流行并被广泛应用。它减轻了用户手动管理云资源的负担，例如管理虚拟机或者裸机器。SCF 也均被主流云供应商所支持。

第二，本文从两个方面降低存储审计系统的成本。首先，本文不再使用传统的云虚拟机来

提供计算能力，而是使用 SCF 来支持传统存储审计方案中的“证明生成”阶段算法。SCF 具有比虚拟机更轻量的优点。它只对用户每次运行程序收取费用，而不向用户收取固定的、长期的（例如一年）租用虚拟机的费用。这样可以帮助用户显著降低成本。其次，本文将 COS 和 SCF 部署在同一云服务提供商的同一地理区域，以进一步降低成本。这一方法带来一些好处，同一区域的通信费用是免费的，因为它属于云厂商的局域网中，不涉及公网上的数据传输费用。此外，COS 和 SCF 也有更好的通信效率。云服务提供商一般会为自己的产品优化或提供专用的数据通信线路。这一优势还将提高系统的执行效率和稳定性，进一步提高系统的实际可用性。

第三，本文以模块化的方式设计系统，并将所提出系统的安全性归约到应用层理论审计协议。所提方案将理论上的 PDP 范式的云存储审计协议作为可以模块化插入审计系统的黑盒。因此理论上现有的其他研究方案均能够被支持。所提出系统的安全性将基于所采用的经过充分研究的理论审计协议的安全性。基于这种模块化设计，用户也可以控制选择合适的审计协议，并在将来需要时更改该协议。

第四，本文通过将认证信息分离为一个独立的数据对象，保持了用户数据的原始格式。这样做的目的是使得外包数据中原始数据和认证标签可以独立存储，使得所有现有的云上应用层功能都得到保留和支持。

## 3.2 系统架构和实现概述

### 3.2.1 系统架构

图2展示了基于无服务器计算模式和云对象存储（COS）的云存储审计系统的架构。它由三个主要实体组成，即用户、无服务器云函数 SCF 和云对象存储 COS。其中，SCF 与 COS 部署在同一云服务厂商的同一区域下。SCF 提供理论审计方案所需的计算服务，并且可以根据审计任务的规模灵活扩展。COS 通过其使用 HTTP 请求/响应的读写接口提供存储服务。

该云存储审计系统选择腾讯云作为后端云服务提供商，使用 Java 实现，包含两个可执行程序，一个在客户端，实现了 KeyGen、Outsource、Audit 和 Verify 算法。另一个以 SCF 的形式存在于云端，主要负责运行 Prove 算法。COS 上不运行任何程序，它只被用作提供云存储服务的基础设施，并支持上述两个可执行程序的数据访问。



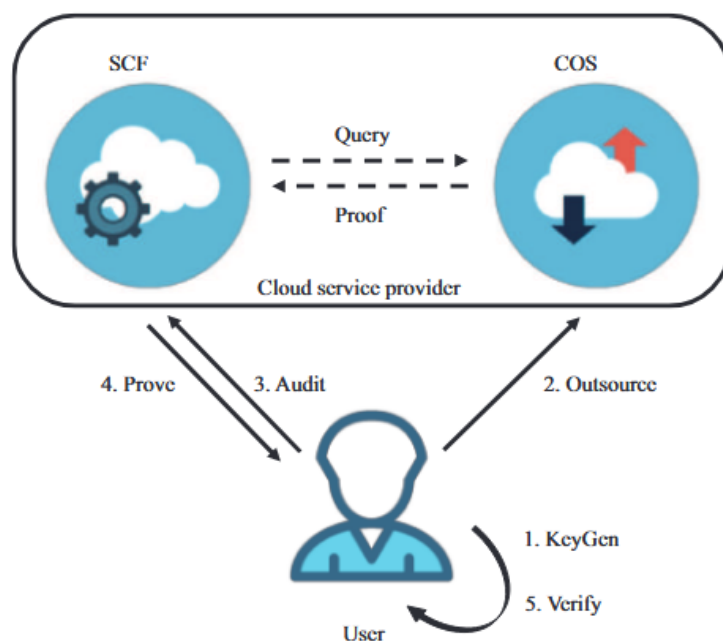


图 2. 系统架构

### 3.2.2 系统的工作原理与流程

首先，用户在本地客户端使用 KeyGen 算法生成密钥。然后，使用 OutSource 算法对数据进行预处理。处理后的数据包括两部分，即原始数据和每个数据块对应的认证标签。之后，用户在系统客户端中通过云 API 将这两部分作为两个独立对象上传到 COS。至此，系统初始化完成。

之后，用户可以定期执行审计。要发出审计请求，用户首先需要执行 Audit 算法计算一个挑战值，然后将挑战值以 HTTPS 请求的方式发送到 SCF，触发云函数的执行。

云函数被触发后，SCF 即开始执行。首先从客户端发送的请求中解析挑战数据。随后，SCF 通过 COS API 从云存储中下载指定的被挑战数据块及其认证标签。之后，SCF 执行 Prove 算法生成审计证明，并以指定的数据格式将证明返回给请求审计的客户端。客户端接收 SCF 返回的 HTTPS RESPONSE，然后解析证明数据。最后，用户在系统客户端执行 Verify 算法，检查云数据是否完好无损。若结论为数据被破坏，则需进行进一步的数据恢复操作。

### 3.3 底层理论审计方案

本文在云存储审计系统的实现中，采用了一种改进的理论云存储审计方案作为底层支持。这一方案具有一定的前沿性，并支持部分数据恢复功能，从而提升了系统的实用性。更重要的是，该方案的安全性已得到验证。图3展示了本系统采用的理论云存储审计方案的具体细节。

- Setup (public parameters)  
 $GF(q)$ ,  $(n, k)$  Reed Solomon error correcting code
- KeyGen( $1^\lambda$ )  $\rightarrow K$   
 $s = (s_1, \dots, s_{n-k}) \leftarrow GF(q)^{n-k}$   
 $K_1$  for PRF  $F_{K_1}(\cdot) : \mathbb{Z} \rightarrow GF(q)^{n-k}$   
 $\alpha_1, \dots, \alpha_n \xleftarrow{R} GF(q)$ ,  $[C, H]$  as in Eq. (1)  
 $K = (K_1, s, \alpha_1, \dots, \alpha_n)$
- Outsource( $F, K$ )  $\rightarrow F'$   
 $F \leftarrow (m_1, m_2, \dots, m_{n_F})$  where  $m_i \in GF(q)^k$   
 $\sigma_i \leftarrow (m_i C^{-1} H) \circ s + F_{K_1}(i)$   
 $// \circ$  denotes vector entrywise product  
 $F' = (m_i, \sigma_i)$  where  $i = 1, \dots, n_F$
- Audit( $K$ )  $\rightarrow q$   
 $i_j \xleftarrow{R} [1, n_F]$  for  $j = 1, 2, \dots, l$   
 $c_j \xleftarrow{R} GF(q)$  for  $j = 1, 2, \dots, l$   
 $q = (i_j, c_j)$  where  $j = 1, \dots, l$
- Prove( $F', q$ )  $\rightarrow (\chi, \Gamma)$ :  
 $\chi \leftarrow \sum_{j=1}^l c_j \cdot m_{i_j}$   
 $\Gamma \leftarrow \sum_{j=1}^l c_j \cdot \sigma_{i_j}$
- Verify( $\chi, \Gamma, q, K$ )  $\rightarrow \delta$   
 $\delta \leftarrow \Gamma \stackrel{?}{=} (\chi \cdot C^{-1} H) \circ s + \sum_{j=1}^l c_j \cdot F_{K_1}(i_j)$

图 3. 理论审计方案

该方案基于有限域  $GF(q)$  上的代数运算, 其中  $q = 2^8$ 。此外, 方案中引入了 Reed-Solomon 纠错码, 设定参数为  $(n, k)$ , 其中  $n$  为码长,  $k$  为消息长度, 每块消息具有  $n - k$  的冗余校验码。为支持随机化验证, 方案采用伪随机函数  $F_{K_1}(\cdot)$ , 其作用是将输入映射到有限域  $GF(q)$  上的一个元素中。实现中使用高级加密标准 (AES) 算法作为伪随机函数的具体实现。

矩阵  $C$  和  $H$  用于嵌入 Reed-Solomon 纠错码, 定义如下:

$$[C, H] = \left[ \begin{array}{ccc|ccc} \alpha_1^0 & \cdots & \alpha_k^0 & \alpha_{k+1}^0 & \cdots & \alpha_n^0 \\ \alpha_1^1 & \cdots & \alpha_k^1 & \alpha_{k+1}^1 & \cdots & \alpha_n^1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{k-1} & \cdots & \alpha_k^{k-1} & \alpha_{k+1}^{k-1} & \cdots & \alpha_n^{k-1} \end{array} \right].$$

在数据外包阶段, 用户首先将原始数据  $F$  划分为多个数据块  $F = \{m_1, m_2, \dots, m_{n_F}\}$ , 其中每个数据块  $m_i$  的长度为  $k$ 。接着, 用户生成密钥  $K_1$  并计算数据完整性标签  $\sigma_i$ , 公式如下:

$$\sigma_i \leftarrow (m_i \cdot C^{-1} H) \circ s + F_{K_1}(i)$$

其中  $s$  是随机生成的向量。标签  $\sigma_i$  本质上是纠错码和消息认证码的结合。计算完毕后, 用户将  $\{m_i, \sigma_i\}$  上传至云端。

在验证阶段, 用户随机生成一个审计查询  $q = \{(i_j, c_j)\}_{j=1}^l$ , 其中  $i_j$  为被选中数据块的索引,  $c_j$  为对应的权重系数。用户将查询  $q$  发送至云端后, 云端分别计算数据线性组合  $\chi$  和认证标签组合  $\Gamma$  作为响应:

$$\chi = \sum_{j=1}^l c_j \cdot m_{i_j}$$

$$\Gamma = \sum_{j=1}^l c_j \cdot \sigma_{i_j}$$

云端返回  $(\chi, \Gamma)$  给用户。用户接收到结果后，验证以下等式是否成立：

$$\Gamma \stackrel{?}{=} (\chi \cdot C^{-1}H) \circ s + \sum_{j=1}^l c_j \cdot F_{K_1}(i_j)$$

若等式成立，则数据完整性通过审计；否则，判定数据存在损坏。

在具体实现中，使用  $GF(2^8)$  作为有限域的好处是有限域上的元素大小正好是 1 个字节，而 1 个字节是 Java 语言的基本数据类型。这样可以使得系统的计算效率非常高，因为不需要涉及大整数的运算。此外，在设置 Reed-Solomon 纠错码时，使用  $(n = 255, k = 223)$ 。即将源数据划分为多个区块，每个区块为 223 字节，标签大小为 32 字节。挑战长度设为  $l = 460$ ，以获得 99% [5] 的审计准确率。

### 3.4 系统评估

图4展示了该系统的计算开销，验证了所提系统的实际可用性。从中可以发现 KeyGen、Audit 和 Verify 算法的计算成本稳定且较小。数据预处理的计算时间随着数据集的大小线性增加。对于无服务器云函数，计算成本以对数方式增加。测试结果在 17.5s 到 69.2s 之间。实验结果中增加的时间来自于数据集变大时 COS 的平均数据请求时间的增加。同一数据集的评估结果波动很大，造成该结果的一个原因来自挑战数据的随机性。另一个原因是 SCF 的执行时间还受到云服务供应商和网络的影响。但是，整体来看 SCF 计算花费的时间较少。

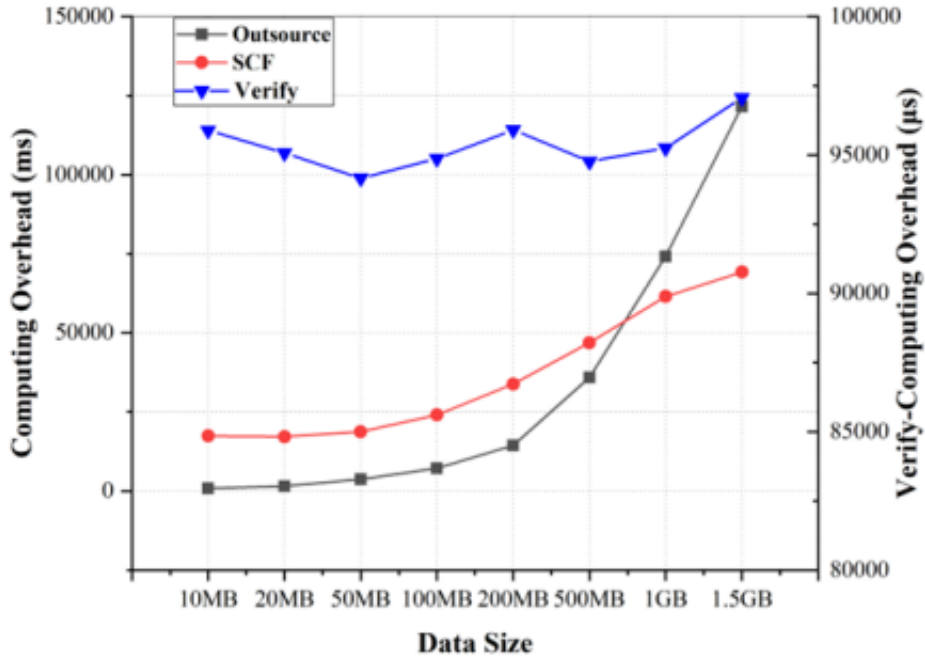


图 4. 主要计算步骤开销

图5展示了该系统的运行成本，评估表明使用 SCF 是具有性价比的选择。方案实现使用腾讯云作为云服务提供商，在实际系统中实施了最大 40GB 数据的审计策略。该条件下系统全年总费用仅为 120.332 元，其中 SCF 占 46%。此外，随着外包存储数据的增加，无服务器计算函数在系统中的成本占比显著降低。



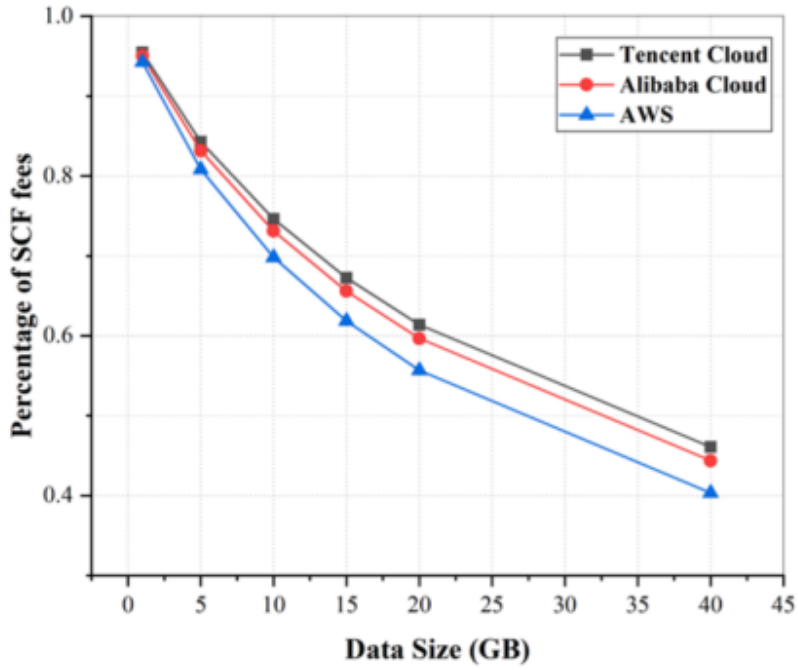


图 5. 系统在主流云服务中的运行成本

## 4 复现细节

### 4.1 与已有开源代码对比

论文开源代码主页：<https://github.com/szu-security-group/IntegrityCheckingUsingSCF>

本次工作的核心任务是将论文开源代码在腾讯云上实现的基于无服务器计算的云存储审计系统在另一主流云平台——Microsoft Azure 上进行复现。因此，原系统设计并实现的核心流程函数包括 GenKey、Outsource、Audit、Prove、Verify 等并不需要进行较大的修改，只需要对一些细节进行针对性的调整。而需要大量修改甚至重建的部分为有关特定云平台、云服务的内容，包括以下部分：

1. CloudAPI.java：涉及云服务配置及初始化、云 API 的调用，主要用于实现对云平台 COS 服务中存储的文件的块读写访问。
2. ScfHandle.java：即无服务器云函数的实现部分，用于定义 SCF 在云端进行审计证明的计算流程等行为。
3. 客户端向 SCF 发送请求、接收响应的部分：不同的云平台支持的云函数触发方式及代码规范存在差异，需要进行针对性的调整。

在本次复现过程中，我在云函数触发机制的选择上做出了一定调整。原代码在腾讯云平台上实现该云存储审计系统时，采用的是 API 网关 (API Gateway) 作为触发方式。然而，Azure 平台并不支持通过 API 网关触发云函数执行的方式。因此，在 Azure 上的复现实现中，我改为采用 HTTP 触发器 (HTTP Trigger) 来实现相同的功能。

具体来说，API 网关是一种常用的管理和触发接口，充当客户端与云函数之间的中间层，负责处理 HTTP 请求并将其转发至对应的云函数，同时还支持请求路由、负载均衡、安全验

证等高级功能。相比之下，Azure Functions 中的 HTTP 触发器则直接响应 HTTP 请求以实现云函数的调用，而无需经过 API 网关的额外处理。在功能实现方面，HTTP 触发器同样能够响应来自客户端的请求，从而完成响应的功能需求。这一替换确保了复现的核心逻辑得以保持一致，同时适配了 Azure 平台的实际限制。

## 4.2 实验环境说明

在本次实验中，使用配置了 Intel 12th i7-12700 (2.10GHz) CPU 和 16GB 内存的计算机来运行客户端程序。

Azure Functions 与 Azure Blob Storage 的区域 (Region) 均配置为 East Asia，以便提高通信效率与降低成本。

## 4.3 使用说明

要运行该系统，首先要在 Microsoft Azure 上准备 Azure Blob Storage 服务和 Azure Functions 服务。有关云服务的配置和使用详情可参阅 Azure 官方文档。

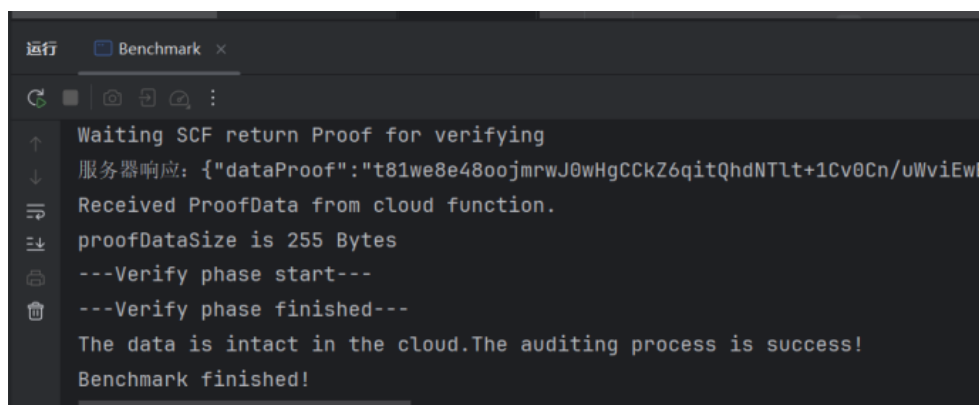
随后，需要在项目目录的 Properties.properties 文件中填入 Azure Blob Storage container 的名称及连接字符串，其中连接字符串中包含了相关的认证信息；并在项目的 Maven 配置文件 pom.xml 中填写 Azure 的 functionName。以上具体信息可从 Microsoft Azure 控制台中找到。

此外，还需替换项目代码中有关文件路径的部分。首先将“filePath”替换为需要存储外包数据并用于后续审计的目标目录。再将“uploadSourceFilePath”、“uploadParitiesPath”、“performanceResult”、“proofdata”分别替换为本地的实际可访问路径，这些文件用于在客户端本地保存审计执行过程中的相关信息。

所有配置完毕后，即可运行主程序：在 IntelliJ IDEA 中，找到 Benchmark 类，这是项目的入口，执行该类以运行系统。

## 5 实验结果分析

图6为该审计系统成功验证云中外包数据完整性的运行结果



```
运行 Benchmark x
Waiting SCF return Proof for verifying
服务器响应: {"dataProof": "t81we8e48oojmrwJ0wHgCCKZ6qitQhdNTlt+1Cv0Cn/uWviEWB
Received ProofData from cloud function.
proofDataSize is 255 Bytes
---Verify phase start---
---Verify phase finished---
The data is intact in the cloud. The auditing process is success!
Benchmark finished!
```

图 6. 审计系统执行结果演示

根据系统记录的性能结果文件，如图7所示，审计一个 1703943 字节（1.62MB）的文件所需时间大约为 3.62 秒（图中时间单位为纳秒）。

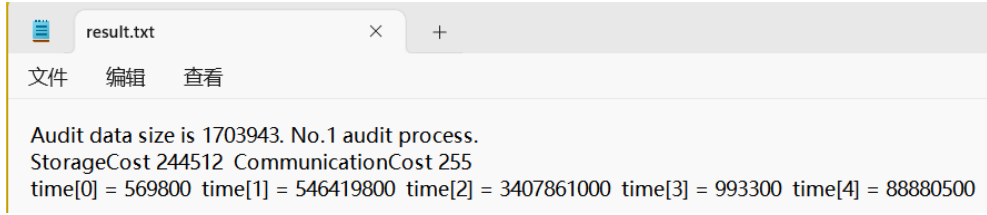


图 7. 系统性能记录文件

## 6 总结与展望

本次研究工作通过在 Azure 平台上复现基于无服务器计算的云存储审计系统，对原论文中提出的实用方案进行了全面的实践验证。复现工作利用 Azure Blob Storage 和 Azure Functions 来实现原系统中的云对象存储（COS）和无服务器云函数（SCF）的任务，并通过合理的设计克服了云平台之间的差异，成功验证了系统在不同主流云平台上的适应性和通用性。

在复现过程中，系统实现了云存储审计的核心功能，包括密钥生成、数据外包、审计挑战、证明生成以及结果验证等步骤，确保了用户能够有效验证外包数据的完整性。同时，我们通过在 Azure 平台上的实验分析，验证了无服务器计算在云存储审计中的高效性。

由于云平台的差异及底层服务器部署的地理位置的不同，Azure 平台中部分服务的性能可能与原系统在腾讯云上的实现存在差异，例如在数据处理延迟方面，仍需要进一步优化和调整。系统对于非专业用户的易用性和用户友好性方面也是未来可以进一步改进的方面。

此外，数据动态性也是该系统未来可以进行深入探索的方向。本系统将 PDP 审计协议作为黑盒，这意味着只要系统采用的 PDP 协议有能力支持数据动态，系统也就能支持数据动态。不过，虽然系统可以实现数据动态，但性能并不理想。这是因为 COS 没有通过提供应用程序接口（API）来原生支持字节级的数据插入和删除功能。例如，用户可以轻松地下载 COS 数据。但是，当用户想插入一个新的数据块时，唯一的方法是用户先下载外包数据进行修改，然后再上传新的整个数据。

尽管本次复现工作验证了系统在多云平台上的适应性和通用性，但如何进一步实现不同云平台上的标准化部署，确保系统功能在不同环境下的一致性，仍是一个重要的研究方向。可以考虑设计平台无关的抽象层，屏蔽底层云平台的差异，使系统能够在各种云环境中无缝迁移。

总之，本次复现工作验证了无服务器计算在云存储审计中的可行性、高效性与经济性，但在系统的跨平台一致性、性能优化及数据动态性方面等方面仍有改进的空间。随着云计算技术的不断发展，相信无服务器计算将在数据安全和云存储审计中发挥越来越重要的作用，为数据完整性保护提供更加灵活、经济和高效的解决方案。

## 参考文献

- [1] Fei Chen, Jianquan Cai, Tao Xiang, and Xiaofeng Liao. Practical cloud storage auditing using serverless computing. *Science China Information Sciences*, 67(3):132102, 2024.
- [2] Ari Juels and Burton S Kaliski Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597, 2007.
- [3] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 598–609, 2007.
- [4] Johann Schleier-Smith, Vikram Sreekanti, Anurag Khandelwal, Joao Carreira, Neeraja J Yadwadkar, Raluca Ada Popa, Joseph E Gonzalez, Ion Stoica, and David A Patterson. What serverless computing is and should become: The next phase of cloud computing. *Communications of the ACM*, 64(5):76–84, 2021.
- [5] Fei Chen, Fengming Meng, Tao Xiang, Hua Dai, Jianqiang Li, and Jing Qin. Towards usable cloud storage auditing. *IEEE Transactions on Parallel and Distributed Systems*, 31(11):2605–2617, 2020.