用于跨平台通信的轻量级高性能远程过程调用框架

哈坎-巴奇和艾哈迈德-卡拉
土耳其安卡拉 Tubitak Bilgem Iltaren

关键词：　远程过程调用、高性能计算、跨平台通信。

摘要：远程过程调用（RPC）被广泛用于构建分布式系统已有近 40 年的历史。针对不同的目的，有多种 RPC 实现方法。一些远程过程调用机制是通用系统，提供多种功能调用模式，因此并不强调性能。另一方面，少数 RPC 机制的实现则以性能为主要考虑因素。这类 RPC 实现集中于减少传输 RPC 信息的大小。在本文中，我们提出了一种新颖的轻量级高性能 RPC 机制（HPRPC），它使用了我们自己的高性能数据序列器。我们比较了我们的 RPC 系统与著名的 RPC 实现（gRPC）的性能，后者既能提供各种调用模式，又能减少 RPC 信息的大小。实验结果清楚地表明，就通信开销而言，HPRPC 的性能优于 gRPC。因此，我们建议将我们的 RPC 机制作为高性能和实时系统的合适候选机制。

1　引言

RPC 使用户能够调用驻留在另一个进程地址空间中的远程进程（Corbin，2012 年）。该进程可能运行在同一台机器上，也可能运行在网络上的另一台机器上。RPC 机制被广泛用于构建分布式系统，因为它能降低系统的复杂性和开发成本（Kim 等人，2007 年）。RPC 的主要目标是使远程过程调用对用户透明。换句话说，它允许用户像本地过程调用一样进行远程过程调用。

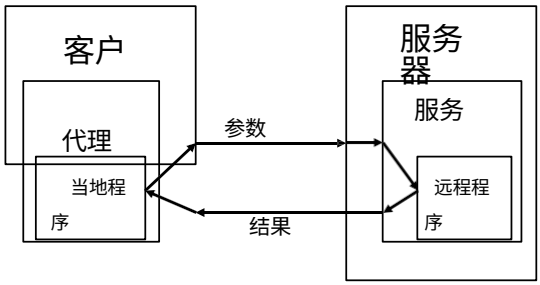
1976 年首次讨论了 RPC 的概念（Nelson，1981 年）。Nelson 在他的研究中探讨了 RPC 的设计可能性。在 Nelson 之后，又开发了一些 RPC 实现，如 Courier RPC（Xe-rox，1981 年）、Cedar RPC（Birrell 和 Nelson，1984 年）和 SunRPC （Sun Microsystems，1988 年；Srinivasan，1995 年）。SunRPC 在 Unix 系统上非常流行。随后，它被移植到 Solaris、Linux 和 Win-dows。SunRPC 被广泛应用于文件共享和分布式系统等多个领域（Ming 等人，2011 年）。

RPC 机制基本上是一种客户机-服务器模式（Ming 等人，2011 年）。客户端发起包含远程过程参数的 RPC 请求

通过代理。然后，请求被传送到一个已知的远程服务器。服务器收到请求后，会将请求委托给包含实际存储过程的相关服务。然后，服务使用收到的参数执行程序并产生结果。服务器将结果发送回客户端。此过程如图 1 所示。

图 1：RPC 客户端服务器通信。

高性能计算系统和实时系统的主要关注点之一是减少数据传输量。因此，想要进行远程过程调用的系统需要一种开销最小的轻量级 RPC 机制。此外，大多数 RPC 机制都是通过网络运行的，因此减小 RPC 数据包的大小非常重要。目前有几项研究



Bagci, H. 和 Kara, A.
用于跨平台通信的轻量级高性能远程过程调用框架DOI: 10.5220/0005931201170124
第 11 届国际软件技术联合会议（ICSOFT 2016）论文集 - 第 1 卷：ICSOFT-EA，第 117-124 页 ISBN：978-989-758-194-6

工作重点是减少 RPC 机制的开销（Bershad 等人，1990 年；Satyanarayanan 等人，1991 年）。

其他 RPC 机制侧重于提供不同类型的调用模式，如异步和并行远程过程调用（Satyanarayanan 和 Siegel，1990 年；Ananda 等人，1991 年）。然而，调用模式数量的增加会给系统带来额外的开销。随着新的高级功能被添加到 RPC 机制中，RPC 数据包的大小也会增加。这是因为协议控制数据越来越大。

gRPC 是一种语言中立、平台中立的开源 RPC 机制，最初由 Google 开发（Google，2015b）。它采用了协议缓冲区（Protobuf）（Google，2015a），这是 Google 的纯开源机制，用于结构化数据服务。gRPC 的目标是在提供多种调用模式的同时最大限度地减少数据传输开销。例如，每个参数都要向服务器传送一个标识符。因此，它不适合以最小数据传输开销为目标的系统。

在本文中，我们介绍了一种轻量级、高性能的远程过程调用机制，即高性能远程过程调用（HPRPC），其目的是最大限度地减少数据传输过量。我们采用了自己的结构化数据序列化机制，名为 Kodosis。Kodosis 是一种代码生成器，可实现高性能数据序列化。我们工作的主要贡献如下：

- 提出了一种新颖的轻量级高性能 RPC 机制。
- 介绍了一种可实现高性能数据序列化的代码生成器。
- 我们对 RPC 机制和 gRPC 的性能进行了比较，结果表明，就数据传输量而言，我们的 RPC 机制优于 gRPC。

本文接下来的内容安排如下：下一节简要介绍与 RPC 机制相关的研究。第 3 节介绍 HPRPC 的详细信息。第 4 节给出了实验结果。最后，我们以一些讨论结束本文。

2 相关工作

自 20 世纪 70 年代中期以来，RPC 机制被广泛用于构建分布式系统。文献中有多种 RPC 实现。在本节中，我们将简要介绍与我们的工作相关的 RPC 机制。

XML-RPC（UserLand Software Inc.，2015）将HTTP作为传输协议，将XML作为交换和处理数据的序列化技术，从而实现远程过程调用（Jagannadham等人，2007）。由于 HTTP 可用于各种编程环境和操作系统，而且 XML 是一种常用的数据交换语言，因此，XML-RPC 很容易适用于任何给定的平台。XML-RPC 是整合多种计算环境的理想选择。不过，它并不适合直接共享复杂的数据结构（Laurent 等人，2001 年），因为 HTTP 和 XML 会带来额外的通信开销。在（All-man，2003）一文中，作者指出，由于 XML-RPC 数据包的大小和编码/解码 XML 的开销增大，XML-RPC 的网络性能比 *java.net* 低。

简单对象访问协议（Simple Object Access Protocol，SOAP）（W3C，2015）是一种广泛应用于网络信息交换的协议。与 XML-RPC 类似，SOAP 一般采用 HTTP 作为传输协议，XML 作为数据交换语言。XML-RPC 更加简单易懂，而 SOAP 则更加结构化和规范化。SOAP 的通信开销比 XML-RPC 高，因为它在要传输的信息中添加了额外的信息。

在（Kim 等人，2007 年）一文中，作者提出了一个灵活的用户级 RPC 系统，名为 FlexRPC，用于开发高性能集群文件系统。FlexRPC 系统可根据请求率动态改变工作线程的数量。它提供客户端线程安全，并支持多线程 RPC 服务器。此外，它还支持各种调用模式和两大传输协议（TCP 和 UDP）。不过，FlexRPC 的这些额外功能可能会造成额外的处理和通信开销。

近年来，高性能计算（HPC）开始使用 RPC 机制。Mercury（Soumagne 等人，2013 年）是专为高性能计算系统设计的 RPC 系统。它提供异步 RPC 接口，允许传输大量数据。它没有实现更高级别的功能，如多线程执行、请求聚合和流水线操作。Mercury 的网络实现是抽象的，因此可以轻松移植到未来的系统中，同时还能实现以下功能

有效利用现有的运输机制。

gRPC（Google，2015b）是最近广泛使用的 RPC 机制，它解决了这两个问题：提供各种调用模式并减少 RPC 消息的大小。它使用 Protobuf（Google，2015a）进行数据序列化。Protobuf 旨在最大限度地减少需要传输的数据量。由于 gRPC 是最近推出的著名 RPC 框架，而 Protobuf 是成熟的序列化机制，因此我们将 HPRPC 系统的性能与 gRPC 进行了比较。比较结果见第 4 节。

3 建议的工作

高性能远程过程调用（HPRPC）框架的设计有三个主要目标：

- 轻量级和简单的架构
- 高性能运行
- 跨平台支持

3.1 建筑学

HPRPC 的设计灵感来自（Google，2015c），但它完全是由我们的再搜索小组根据我们的目标和经验实现的。它依赖于三个主要组件和一个控制器（图 2）。*RpcChannel* 是通信层，负责数据传输。*RpcClient* 负责在通道上发送客户端再请求，*RpcServer* 负责处理从通道传入的客户端请求。另一方面，*RpcController* 是主控制器类，负责将客户端/服务器的信息导入/导出通道。我们将在下文中深入探讨这些类的功能。

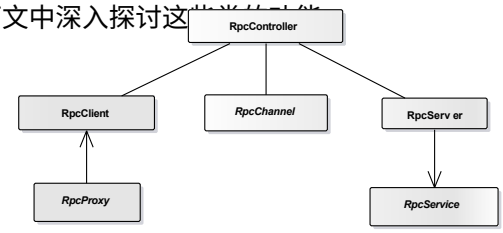


图 2：HPRPC 框架概述。

识别信息，并返回一个*二进制写入器对象*来传输参数。调用者将方法参数/返回值序列化到 *BinaryWriter* 对象，并调用 *EndCall* 方法完成远程过程调用。另一方面，接收 *RpcChannel* 会反序列化 *RpcHeader* 结构，并将其与 *BinaryReader 对象*一起发送给控制器，以便读取参数/返回值。控制器根据 *RpcHeader* 信息调用客户端或服务器消息处理方法。

目前，*RpcChannel* 仅针对 Tcp/Ip 通信层实施。不过，该架构允许任何其他包含管道或共享内存机制的实现方式。

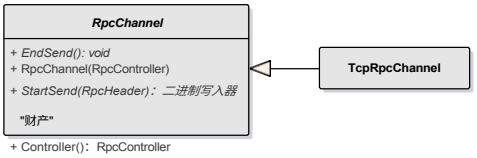
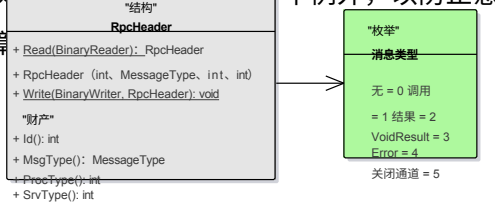


图 3：RpcChannel 类。

RpcHeader 是一个由 4 个字段组成的简单结构（图 4）：

- 标识呼叫的唯一 ID
- 标识有效载荷的信息类型
- 服务类型
- 调用程序的类型

头结构体（Header struct）指定了序列化/解序列化方法，这些方法使用 4 字节的前言（preamble）来保护信息的一致性。如果前一条信息未被正确读[illegible]个例外，以防止意外故障



3.1.1 频道

RpcChannel 实现了发送/接收信息的双工传输机制（图 3）。客户机/服务器通过调*用 Start- Call* 方法来发起请求，然后请求一个 *RpcHeader*，该 *RpcHeader 可用于发送或接收信息。*

图 4：RpcHeader 结构。

3.1.2 客户

RpcClient 是代理用来发送方法调用请求并等待响应的类（图 5）。它包含不同的 *XXXCall* 方法，用于启动远程过程调用并处理其响应。调用请求会返回一个 *PendingCall* 对象，该对象指定了返回值的票据。调用者在 *PendingCall 对象*上等待。当收到服务器的响应时，将根据远程过程的响应返回值或抛出异常。

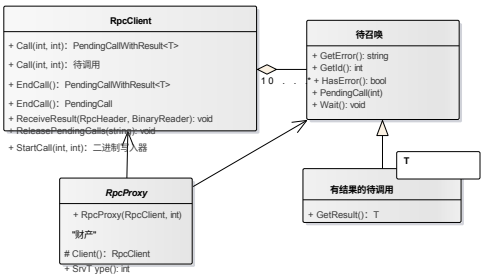


图 5：RpcClient 类。

代理是自定义生成的类，可将用户请求转为 *RpcClient* 调用。对于每个方法调用，代理类都会调用 *RpcClient* 的 *XXXCall* 方法，并等待 *PendingCall* 对象。这种机制为 HPRPC 框架的用户引入了透明度。图 6 显示，在 *ConnectionTesterProxy* 上实现了 IConnectionTester 接口。用户调用该接口的 *KeepAlive* 方法，无需处理 HPRPC 框架的细节。

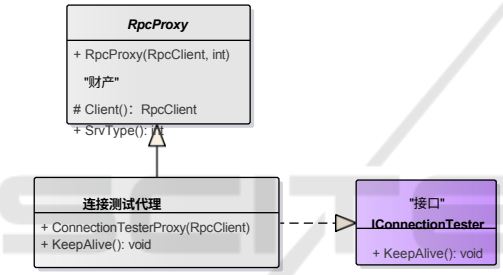


图 6：RpcProxy 类。

3.1.3 服务器

RpcServer 负责处理客户端请求（图 7）。根据 *ServiceType* 属性，它调用相关 *RpcService* 的 *Call* 方法。然后分析其 *ReturnValue（返回值），如果*出现错误，则分析客户端的 ex- ception 信息。

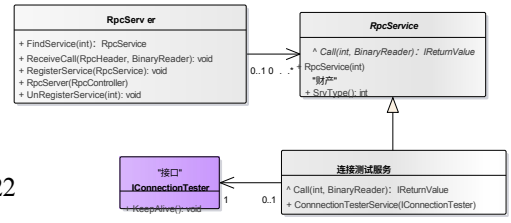


图 7：RpcServer 类。

服务是自定义生成的类，可超越 *RpcService* 类的*调用*方法。它们使用适当的参数调用其内部服务实例的相关方法。服务类不像代理那样实现服务方法，而是在它们拥有的对象上调用服务。如图 7 所示，当客户端调用 *KeepAlive* 方法时，*RpcServer* 会将此调用指向 *ConnectionTesterSer-vice*，后者随后会调用自己对象的 *KeepAlive* 方法。

RpcService 类的 *Call* 方法会返回一个 *IReturnValue* 对象，该对象保存存储过程调用的返回值。如图 8 所示，返回值可以是 void 或 *ReturnValue* 类的类型化实例。它们由 *RpcServer* 处理，以传输方法结果。

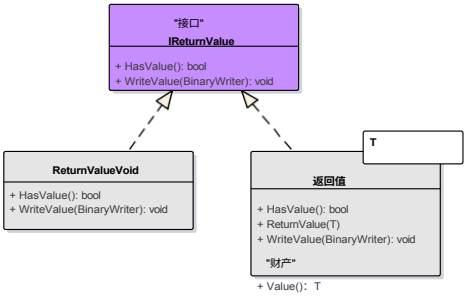


图 8：RpcReturnValue 类。

3.2 高性能

*高性能*运行是 HPRPC 的主要目标之一。为实现这一目标，我们重点关注两个主要领域：

- 设计轻巧
- 优化数据序列化

3.2.1 轻型设计

服务是自定义生成的类，可超越 *RpcService* 类的*调用*方法。它们使用适当的参数调用其内部服务实例的相关方法。服务类不像代理那样实现服务方法，而是

RPC 的主要目的是通过程序调用在两个端点之间轻松建立连接。异步调用、多用户/多服务器、异构平台等任何其他功能都是为满足特殊用户需求而附加的功能，但由于其额外的控制机制和数据交换成本，可能会降低性能。为了实现 HPRPC 的*高性能*目标，我们跳过了这些功能，以提高整体性能。我们的*轻量级设计的*基本特征概述如下：

- 同步呼叫：我们的客户可以一次调用一个程序。
- 单客户端/单服务器：我们的客户端和服务器类允许使用单一端点。
- 同质平台：尽管我们支持跨平台通信，但端点必须是同质的。因此，它们的字节顺序（小/大端）、字符串编码等都应相同。

我们通过使用 *PendingCall* 类实现了异步调用架构。今后，我们计划在性能损失有限的情况下，有选择地添加异步调用。

3.2.2 数据序列化

过程调用通常包括一些需要传输的附加数据，如方法参数和返回值。因此，RPC 机制的性能与其数据序列化/解序列化机制的性能高度相关。在 HPRPC 中，我们借助我们研究小组开发的名为 *KodoSis* 的代码生成器实现了数据序列化。它具有以下功能：

- 一致性：我们的数据结构在单一配置文件中定义，并由 *KodoSis* 为所有终端和平台生成，这使我们能够使用一致的数据结构。
- 完整性*KodoSis* 可为所有数据结构生成序列化/解序列化例程，无需任何自定义代码。
- 最小开销：数据序列化过程中不需要信息标签，因为两端都知道数据的完整结构。
- 性能*KodoSis* 序列化例程使用特殊机制，特别是针对 Blit 表类型的列表，以提高性能。

3.3 跨平台支持

我们的架构已在多种语言和平台上实现。同一架构的类名和方法名完全相同，可在多种语言和编译器上实现和测试。所有实施方案都可以相互通信，而无需了解其他平台的情况。

在 Windows 操作系统中，我们使用 .NET Framework 支持 C# 语言，使用 Microsoft Visual C++ 编译器支持 C++ 语言。在类似 Linux 的操作系统中，我们使用 *gcc* 编译器支持 C++ 语言。

在 C++ 中，我们使用了 *共享 ptr*、*线程*、*mutex* 等 C++ v11 结构，以提高健壮性并简化代码。在一些不支持 C++ v11 的旧系统（gcc v4.3）中，我们使用了 Boost Framework（Boost，2015 年）的等价类。

4 评估

如前所述，HPRPC 的目标是尽量减少

RPC 客户端和 RPC 服务器之间的通信开销。为了实现这一目标，它利用 Kodosis 将通过 RPC 通道传输的数据量降至最低。我们进行了几项实验来评估我们提出的工作。在这些实验中，我们比较了 HPRPC 和 gRPC 的 性能。gRPC 是著名的开源 RPC 机制，最初由 Google 开发。它采用 Protobuf 进行数据序列化。与 Kodosis 类似，Protobuf 也旨在最大限度地减少数据传输量。

4.1 评估方法

实验中选择了四种不同的操作。每种操作都使用不同类型的请求和响应。在评估 XML-RPC 时也使用了 *Average* 和 *GetRandNums* 操作（Allman，2003 年）。这些操作的详情如下：

- ***HelloReply SayHello（HelloRequest 请求）***
  此操作发送一个包含字符串类型参数的 *HelloRequest* 信息，并重新生成一个包含请求参数的 "Hello "连接版本的 *HelloReply* 信息。此操作代表一个*小请求/小回复*事务。
- ***DoubleType 平均值（数字 Numbers）***
  此操作返回包含 32 位整数的 *Numbers* 列表的平均值。*数字列表*的大小并不固定。我们测试了 RPC 机制在*数字*列表数量不同的情况下的 性能。此操作代表了*大再查询/小响应*事务。
- ***Numbers GetRandNums (Int32Type Count) 在***
  给定所需随机数的情况下，该操作会返回一个 32 位随机数列表。我们测试了不同*计数*参数下 RPC 机制的性能。此操作代表一个*小请求/大响应*事务。
- ***大型数据　发送 RcvLargeData　(大型数据)***
  该操作接收大量数据，并返回相同的数据。*LargeData* 有不同的版本，包含不同数量的参数。此操作代表*大请求/大响应*事务。

在实验中，我们测量了 gRPC 和 HPRPC 每个操作的事务处理时间。所有实验重复 1000 次，并记录总的事务处理时间。在大数据实验中，我们还测量了 RPC 数据包的大小，以比较 gRPC 和 HPRPC 的交易时间。

通过 RPC 通道传输的数据量。

4.2 实验结果

在第一个实验中，我们测量了 *SayHello* 函数的事务处理时间，这是一个*小再查询/小响应*操作。gRPC 和 HPRPC 1000 次操作的总交易时间分别为 305 毫秒和 279 毫秒。HPRPC 的性能比 gRPC 高 9.3%。这是由于 Protobuf 在 *HelloRequest* 消息中标记了字符串参数，而 Kodosis 在序列化过程中没有使用标记。因此，与 HPRPC 相比，gRPC 需要传输的数据总量会增加。在本实验中，我们只在请求和响应报文中使用单个参数。当我们增加单个报文中的参数数量时，我们还将检验两种 RPC 机制的性能。实验结果以大数据测试的形式呈现。

第二项实验使用 *Average* 函数来衡量两种 RPC 机制的性能，该函数代表了大请求/小响应的传输过程。图 9a 显示了 *Average* 操作测试的结果。在本实验中，客户端针对不同数量的 32 位整数从 RPC 服务器调用相应的 *Average* 函数。当输入数为 10 时，HPRPC 的性能比 gRPC 高 8.5% 。当输入大小增加到 10000 时，性能差距逐渐拉大，并达到最高值。此时，gRPC 的总传输时间比 HPRPC 高 21.6%。当输入大小超过 10000 时，性能差异会更大。这一结果提示我们，当数据量增加时，HPRPC 的性能会有所提高。

在*平均*函数实验中，我们只增加了请求信息的大小。为完整起见，我们还测量了响应信息大小增加时的性能。*GetRandNums* 函数的实验结果如图 9b 所示。不出所料，实验结果与之前的实验结果类似。在这种情况下，性能差异进一步加大。当从 RPC 服务器请求 10 个运行数据时，HPRPC 的性能比 gRPC 高 12.9%。当请求的随机数为 10000 时，两者的差距达到了 39.2%。

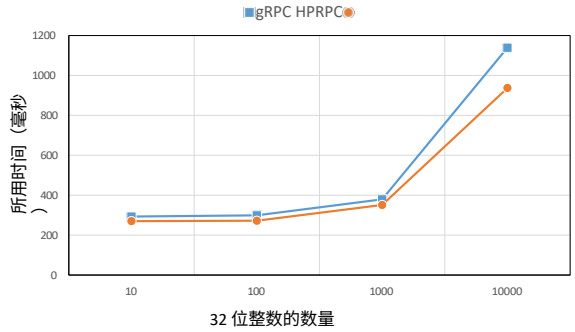
在 *Average* 和 *GetRandNums* 实验中，*Num-bers* 消息类型分别用作请求和响应类型。*Numbers* 数据结构将多个数字建模为同一类型的重复成员。

在接下来的实验中，我们将评估当消息类型由非重复成员组成时 RPC 机制的性能。在下一个实验中，我们将评估当消息类型包含非重复成员时 RPC 机制的性能。在本实验中，我们采用了不同版本的 *LargeData* 消息类型，如 LargeData128、LargeData256 等。与 LargeData 相连的数字表示信息类型中的成员数。例如，LargeData128 消息类型包含 128 个不同的 Integer 类型成员。

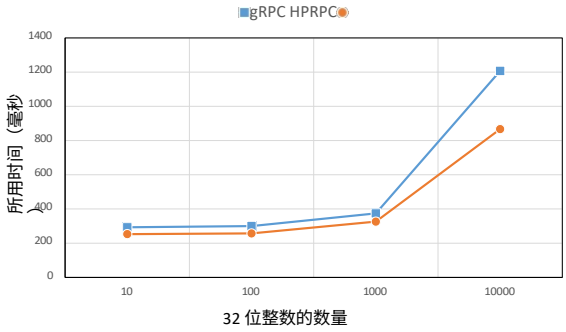
SendRcvLargeData 实验采用了 7 种不同版本的 *LargeData* 消息类型，以评估 RPC 机制在消息中成员数量不同的情况下的性能。在这些测试中，客户端从 RPC 服务器调用相应的 SendRcvLargeData 函数，服务器将相同的数据发送回客户端。实验结果如图 9c 所示。当成员数为 128 时，HPRPC 的性能是 gRPC 的近 2.7 倍。随着消息类型中成员数的增加，gRPC 的事务处理时间比 HPRPC 大幅增加。在成员数为 8192 的极端情况下，gRPC 消耗的事务处理时间约为 HPRPC 的 12.5 倍。这一结果清楚地表明，随着 RPC 消息中成员数量的增加，HPRPC 的性能要优于 gRPC。

为了找出当消息中的成员数量增加时，gRPC 为什么需要如此多的事务处理时间，我们进一步分析了 *SendRcvLargeData* 实验的结果。在分析中，我们比较了 gRPC 和 HPRPC 传输的数据包大小。分析结果如图 9d 所示。分析结果表明，当信息中的成员数增加时，gRPC 往往比 HPRPC 使用更多字节。这是因为 gRPC 为报文中的每个成员都使用了唯一的标签。信息通过这些标签进行序列化，因此 RPC 数据包的大小也随之增大。此外，随着成员数的增加，用于代表标签值的字节数也会增加。标签在某些情况下是有用的，如启用可选参数，但标记每个参数会带来额外的通信和处理开销。因此，我们得出结论，标记参数对高性能系统并无用处。

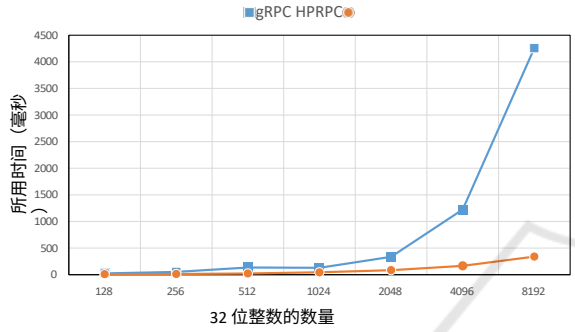
HPRPC 假定消息中的所有参数都是必需的，而且参数的顺序不会改变。因此，RPC 消息以相同的顺序序列化和去序列化，这就消除了在 HPRPC 中标记参数的需要。在高性能系统中，这一假设使 HPRPC 比其他 RPC 实现更具优势。这是由于



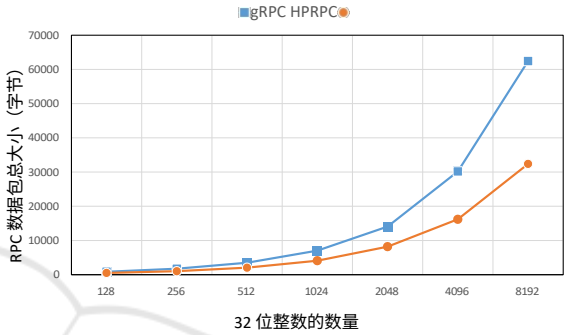
(a) 平均



(b) GetRandNums



(c) SendRcvLargeData



(d) 数据包尺寸大

图 9：实验结果。

这是因为 RPC 数据包的大小减小了，而且消除了标记的额外生产成本。*SayHello* 和 *GetRandNums* 的实验结果也验证了这一论点。

此外，Protobuf 还支持多种消息类型定义功能，如可选参数，这给 RPC 包的大小带来了额外的开销。另一方面，我们的 Kodosis 实现提供了一种简单的消息定义语言，只支持基本的必要功能。Kodosis 的这些功能有助于减少 RPC 数据包的大小，从而减少通信开销。

5 结论

HPRPC 是一种 RPC 机制，侧重于三个主要目标：轻量级的简单架构、高性能和跨平台支持。在本文中，我们通过相关的类图详细介绍了我们的架构，并介绍了一些性能评估结果，这些结果显示了我们相对于 gRPC 的优势。

为了评估我们的 RPC 机制的性能，我们将 HPRPC 与 gRPC 进行了比较。gRPC 是最近出现的一种著名 RPC 机制，它既能提供各种调用模式，又能减少 RPC 消息的大小。实验结果清楚地表明，HPRPC 优于 gRPC。

在所有测试用例中，gRPC 的通信开销都低于 gRPC。这是由于 gRPC 的高级功能在数据传输中带来了额外的开销。此外，gRPC 的底层序列化机制 Protobuf 会标记消息中的所有参数。因此，gRPC 信息在数据传输中往往会占用更多字节。随着消息中参数数量的增加，消息的大小会变得更大。另一方面，HPRPC 不仅提供了所需的所有基本功能，还确保了 RPC 报文大小的减小和处理开销的最小化。这一优势使 HPRPC 在高性能场景中表现更佳。我们的结论是，HPRPC 是一种适合在高性能系统中使用的 RPC 机制。

目前，我们使用 C# 和 C++ 语言在 Windows 和 Linux 操作系统上实现了 HPRPC。未来，我们计划使用 Java 实现我们的架构，并增加跨平台支持。此外，我们还计划在服务器端实现异步调用模式和并发执行请求。

参考文献

Allman, M. (2003).xml-rpc 评估。*ACM sig- metrics performance evaluation review*，30（4）：2-11.

Ananda, A. L., Tay, B., and Koh, E. (1991).Astra-an asynchronous remote procedure call facility.*第 11 届国际会议，第*172-179 页。IEEE.

Bershad, B. N., Anderson, T. E., Lazowska, E. D., and Levy, H. M. (1990).轻量级远程过程调用。*ACM Transactions on Computer Systems (TOCS)*，8(1)：37-55.

Birrell, A. D. and Nelson, B. J. (1984).Implementing Remote Procedure Calls.*ACM Transactions on Computer Systems (TOCS)*，2（1）：39-59.

Boost (2015) 。 Boost Framework. http://www.boost.org.Ac- cessed: 2015-12-08.

Corbin, J. R. (2012).*分布式应用艺术：远程过程调用编程技术》*。Springer Science & Business Media.

Google (2015a). 谷歌协议缓冲区（Pro- tobuf）：谷歌数据交换格式。文档和开源版本。https://developers.google.com/protocol-buffers。访问时间：2015-12-08。

Google （2015b）. gRPC. http://www.grpc.io.访问日期：2015-12-08。

谷歌（2015c）。Protobuf-Remote： RPC Implementation for C++ and C# using Protocol Buffers. https://code.google.com/p/protobuf-remote. 获取时间：2015-12-28。

Jagannadham, D., Ramachandran, V., and Kumar, H. H. (2007)。Java2 分布式应用程序开发（socket、rmi、servlet、corba）方法、xml-rpc 和网络服务功能分析与性能比较。*通信与信息技术》，2007 年。ISCIT'07。国际研讨会，*第 1337-1342 页。IEEE.

Kim, S.-H., Lee, Y., and Kim, J.-S. (2007.(2007).Flexrpc: a flex- ible remote procedure call facility for modern cluster file systems.In *Cluster Computing, 2007 IEEE Inter- national Conference on,* pages 275-284.IEEE.

Laurent, S. S., Johnston, J., Dumbill, E., and Winer, D. (2001).*用 XML-RPC 编程网络服务》*。O'Reilly Media, Inc.

Ming, L., Feng, D., Wang, F., Chen, Q., Li, Y., Wan, Y., and Zhou, J. (2011)。无穷宽带上性能增强的用户空间远程过程调用。*光子学与光电子学会议 2011》*，第 83310K-83310K 页。国际光学与光子学会。

Nelson, B. J. (1981)."远程过程调用》CSL-81-9。技术报告，施乐帕洛阿尔托研究中心。

Satyanarayanan, M., Draves, R., Kistler, J., Klemets, A., Lu, Q., Mummert, L., Nichols, D., Raper, L., Rajendran, G., Rosenberg, J., et al. (1991).Rpc2 用户指南和参考手册。

Satyanarayanan, M. and Siegel, E. H. (1990).大型分布式环境中的并行通信。*Com- puters, IEEE Transactions on*, 39(3):328-348.

Soumagne, J., Kimpe, D., Zounmevo, J., Chaarawi, M., Koziol, Q., Afsahi, A., and Ross, R. (2013)。水星：启用远程过程调用以实现高性能

计算。*集群计算（CLUSTER），2013 IEEE 国际会议*，第 1-8 页。IEEE.

Srinivasan, R. (1995).RPC： Remote Procedure Call Proto- col Specification Version 2 (RFC1831)。*互联网工程任务组*。

Sun Microsystems, I. (1988).RPC：远程过程调用协议规范版本（RFC1057）。*网际工程任务组*。

UserLand Software Inc. (2015)。XML-RPC Specification. http://xmlrpc.scripting.com/spec.html. 获取时间：2015-12-15。

W3C(2015). SOAP Version 1.2 Part1：消息传送框架（第二版）。http://www.w3.org/TR/soap12/。访问日期：2015-12-
15.

Xerox, C. (1981).Courier：远程过程调用协议》。*施乐系统集成标准 038112*。