# Q1论文试图解决什么问题？

智能手机未来增长的最大障碍之一是电池技术。随着处理器的速度越来越快，屏幕越来越清晰，设备也配备了更多的传感器，智能手机消耗能量的能力远远超过了电池提供能量的能力。不幸的是，电池的技术趋势表明，这些限制将继续存在，而能源仍将是手持移动设备的主要瓶颈。

# Q2这是否是一个新的问题？

解决手持移动设备的能源问题并不是一个新的问题。

# Q3这篇文章要验证一个什么科学假设？

MAUI通过细粒度的代码卸载来最大限度地节省能源的潜力，同时最小化应用程序所需的更改；这种方法可以节省移动设备能源消耗，提高移动设备性能。

# Q4有哪些相关研究？如何归类？谁是这一课题在领域内值得关注的研究员？

1. **程序划分**

①程序划分

在Spectra [9,10]中，程序员提供了关于如何分区给定不同忠诚度的应用程序的执行计划，这是一种特定于应用程序的服务质量度量。在运行时，Spectra 监视到远程服务器的连接，并选择最大化用户提供的实用程序功能的执行计划。Chroma [1,3]建立在来自光谱的思想之上，同时努力减轻程序员的负担。Chroma允许程序员使用声明性语言指定“策略”（程序如何利用基础设施资源的策略）。光谱要求用户为系统提供实用函数，而Chroma则依赖于一个名为Prism [2]的外部系统，通过跟踪用户来自动构造这些实用函数。光谱和色谱都借鉴了Odyssey[30]的想法，这是一个研究支持应用程序能够适应不同网络变化、CPU负载与电池情况的操作系统。它们另外一个共同的方法：Protium [41]，手动将应用程序分区到在移动设备上执行的本地查看器和执行应用程序逻辑的远程服务器。尽管MAUI的体系结构借鉴了所有这些系统的思想，但MAUI通过自动化程序分区所需的许多步骤，减轻了程序员的负担。

②代码迁移

在代码迁移的轻量级方法上有许多早期的工作，包括用于移动对象的系统，如Emerald [18], Network Objects [4], Obliq [5], Rover [17], 与Agent Tcl [12].。这些系统侧重于使代码和数据能在分布式系统中的节点之间轻松地移动。但这些系统都没有试图向程序员隐藏分布，也没有实现自动迁移。

③自动程序分区

Coign [16]提供了一个粗粒度的，能自动将DCOM应用程序分区到客户端和服务器的组件，这种方法无需修改源代码。Kremer等人[23]提出使用静态分析来选择远程执行的方法，以节省能源。在[28]中，作者提出将一个类c程序静态地划分成一个节点级nesC 程序集合，这些程序运行在传感节点上。Hydra [40]支持将计算卸载到专门的处理器，如gpu、nic和磁盘控制器上，它使用ILP来决定卸载什么代码。Wishbone [29]使用一种基于配置文件的方法来划分应用程序，指定为操作员的数据流图。MAUI将许多这些想法改造到今天的移动领域，其中移动设备和远程服务器使用不同的指令集架构（不像[16]），而MAUI支持的移动应用程序并不是通常在传感器网络上运行的数据收集应用程序。

④远程运行构建好的程序

另一种方法是将复制构建好的程序到移动应用程序中，并使用分布式协议来同步应用程序的副本。在这样的系统中，移动用户可以使用任何可用的副本来运行应用程序，无论它们是本地的还是远程的。staging[11]和flfluid replication[20]建议巧妙地使用“代理”（附近的不受信任和不受托管的公共机器）作为应用程序副本的分段服务器。Slingshot[39]通过添加动态实例化“有状态”应用程序副本的功能，扩展了早期的工作。所有这些系统还依赖于程序员来分区应用程序与构建代码。

⑤最接近MAUI的系统

最接近MAUI的系统是OLIE系统[13]。OLIE在运行时动态划分Java应用程序，此时程序员的负担很小。在运行时，OLIE监视网络条件并配置程序行为。然而，MAUI与OLIE一个主要的区别是：OLIE关注于克服移动设备的内存资源限制，MAUI的重点是降低能源消耗。

1. **进程和虚拟机迁移**

远程执行的另一种方法是为进程迁移提供操作系统支持，比如Sprite [8]和Amoeba [26]等系统。再往后，Zap [31]在进程迁移的时候具有检查点和重新启动的功能。最近，在虚拟机实时迁移方向，[7]能够迁移整个操作系统以及运行的所有应用程序，CloneCloud系统[6]和Cloudlets[37]将这种技术应用到移动设备环境中。所有这些方法都大大减轻了程序员的负担，这也是MAUI的目标之一。MAUI关注于节能，因此选择了一个更注重于物尽其用的代码卸载方法。MAUI的粒度更细，而不是迁移整个操作系统或者整个进程。

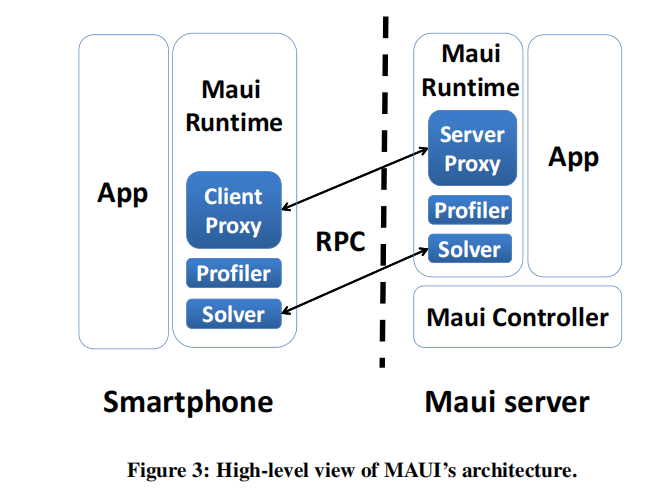
# Q5论文中提到的解决方案之关键是什么？

MAUI通过细粒度的代码卸载来最大限度地节省能源的潜力，同时最小化应用程序所需的更改。

# Q6论文中的实验是如何设计的？

**1.实验原理**

**1.1总体架构**



**1.1.1程序员如何标记远程执行的程序**

除了以下三类其它都是可以远程执行的程序：

1. 实现应用程序用户界面的代码；
2. 与I/O设备交互的代码，其中这种交互只在移动设备上有意义；
3. 与任何可能受到重新执行影响的外部组件交互的代码（例如从网上商店购买商品）

**1.1.2在不同的CPU架构上执行相同的代码**

MAUI支持 .NET平台支持的语言编写的应用程序，所有的CLR应用程序都被编译为CIL中间语言，而不管它们是用什么源语言编写的。因此，通过利用CLR的能力，MAUI获得了独立于当今智能手机之间的指令集架构的差异。

注：要开始运行应用程序，MAUI运行时必须确保MAUI服务器具有应用程序可执行文件的副本。MAUI运行时支持两个选项：1)MAUI服务器可以获得程序的副本可执行文件直接从智能手机（导致延迟和消耗能量），或2)MAUI运行时可以发送签名的可执行文件MAUI服务器然后从云服务下载实际的可执行文件。（虽然不知道后者服务器怎么做到的）

**1.1.3使用反射提取可删除的方法**

使用.NET CLR的自定义属性特性。属性是注释特定代码元素的元数据，它们包含在编译的.NET CLR可执行文件中。应用程序开发人员通过向每个安全的远程执行的方法添加“[可远程操作]”属性来修改应用程序的源代码。MAUI运行时使用.NET反射API [22,36]来自动识别开发人员将哪些方法标记为适合远程执行，只需通过在可执行文件中搜索那些标记为“[可远程]”属性的方法。

**1.1.4执行代码卸载**

* 在编译时，MAUI生成两个**代理**，一个运行在智能手机上，另一个运行在MAUI服务器上。这些代理的作用是**实现**由MAUI求解器所做出的决策。求解器根据来自MAUI分析器的输入来决定方法是否应该在远程或本地执行。
* MAUI中的状态和控制传输的粒度在方法级别。
* 对于将控制从本地智能手机转移到远程服务器的调用，本地代理在调用之前执行状态序列化，然后在调用之后对返回的应用程序状态进行反序列化。当当前在MAUI服务器上执行的一个可远程操作的方法调用一个不可远程操作的方法时，服务器端代理将执行必要的序列化，并将控制权传输回智能手机。
* MAUI运行时目前对多线程应用程序的支持有限，支持单线程。

**1.1.5处理故障**

当智能手机在服务器执行远程方法时失去与服务器的联系时，MAUI将控制权发回本地代理。此时，代理既可以在本地重新调用该方法，也可以尝试找到一个替代的MAUI服务器，然后在新的MAUI服务器上重新调用该方法。因为程序状态只在方法的开始和结束时被转移，所以重新执行方法的一部分将不会影响程序的正确性。

**1.2MAUI分析器**

MAUI通过分析器的分析决定方法调用是应该在本地运行还是远程运行。

**1.2.1设备分析（探索如何描述设备应用的耗电量）**

①如何测量耗电量

我们将一个硬件功率计[25]连接到智能手机的电池上（见图5）（这里跟butterfly一样）

②合成基准测试

受JouleMeter [19]的启发。我们构造了一个同时使用整数和浮点算法的合成基准测试。我们多次运行合成基准，改变基准的运行时，并检测它来记录每个基准所需的CPU周期数。

③建立CPU周数与耗电量的模型

我们利用收集到的CPU利用率和相应的智能手机能耗样本，使用最小二乘线性回归建立一个简单的线性模型。

**1.2.2程序分析**

* MAUI分析器测量其状态传输需求、其运行时持续时间以及执行它所需的CPU周期数。状态传输需求包括程序执行所需数据以及执行完成之后需要返回的状态。
* MAUI使用该方法的持续时间和CPU周期来估计在智能手机上运行该方法所消耗的能量。

**1.2.3开销分析**

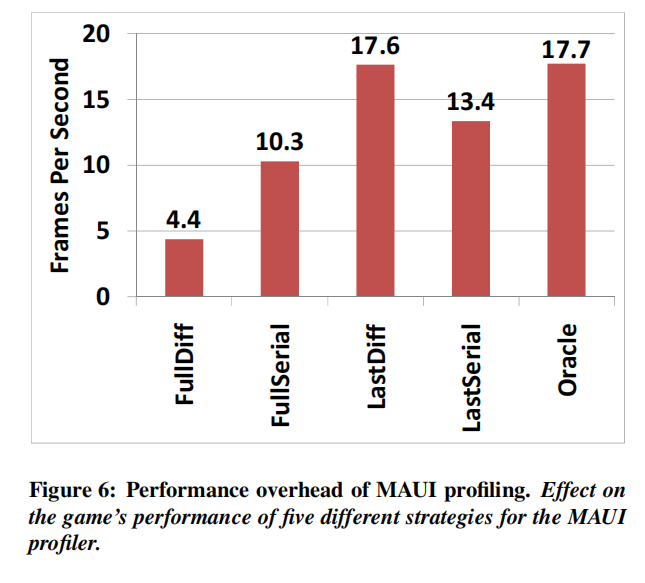
MAUI代理发送增量的状态，以减少状态传输的网络和能量开销。为了实现这一点，本地MAUI代理会跟踪已发送的状态，并且调用前计算增量。但为MAUI运行时**引入了意想不到的复杂性。**

①状态增量计算为状态的估计带来难度

MAUI**分析器观察**程序在执行时随时间变化的行为，并且MAUI**求解器**使用过去的程序行为作为应用程序未来行为的预测器。当方法A调用方法B时，MAUI分析器测量需要的状态的大小，并且无论B是否实际被远程移动，都执行这种测量。此时性能开销只是使用XML序列化器将状态转换为XML格式，然后度量缓冲区大小的成本。分析器观察A调用B的次数越多，它对未来行为的估计就越好。MAUI扰乱了它对程序的测量结果。如果MAUI最近卸载了一个方法，那么增量可能会很小，而如果MAUI在很长一段时间内没有卸载，那么这个增量可能会大得多

②状态增量计算对交互式应用程序的性能存在影响

为了表征这种影响，我们进行了以下实验。我们使用了五种不同的策略来分析一个交互式应用程序（一个视频游戏）



使用增量来发送状态或许可以节省能量，但也可能导致帧率的下降，因此应该从中找寻一种平衡。

**1.2.4网络分析**

①PSM

* 对于使用PSM和像MAUI这样的系统有重要的影响。例如，对于较短的rtt（在我们的实验中低于75 ms），禁用PSM可以节省能量，因为传输要短得多，而且设备在传输期间消耗的总体能量更少。然而，当RTTs接近100 ms时，启用PSM可以节省能量，因为无线发送方在等待来自接收方的ack时正在睡觉。同样的模式也发生在高于100 ms的时候——在缓慢启动时，rtt被有效地舍入到200 ms。
* 不幸的是，我们不得不在当前的MAUI实现中禁用PSM，因为视窗移动没有公开一个允许常规应用程序启用或禁用PSM的API，我们只能通过系统设置打开和关闭它。

②网络测量

没有必要使用专门的工具来单独测量无线链路的往返时间、带宽和数据包丢失。相反，我们的实验表明，以下非常简单的技术效果很好：

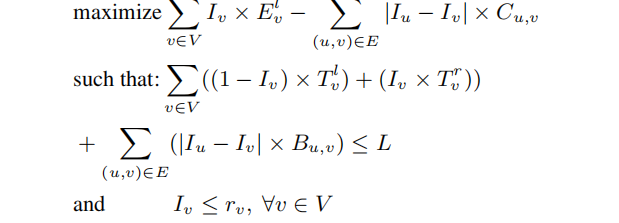
TCP协议下发送10 KB的数据到MAUI服务器，并测量传输持续时间，以获得平均吞吐量。**每次MAUI卸载一种方法时，分析器利用这个机会获得网络特征的最新估计（顺便发送）**；然后将这些最近的估计与当前估计进行平均。最后，如果一分钟内没有执行传输，MAUI通过向服务器发送10 KB来进行另一次测量，以获得新的估计值。

**1.3MAUI求解器**

MAUI求解器使用由MAUI分析器收集的数据作为全局优化问题的输入，该问题决定了哪些可远程操作的方法应该在本地执行，哪些应该远程执行。求解器的目标是找到一种程序划分策略，使智能手机的能源消耗最小化，同时受到延迟的限制。

**1.3.1求解器策略计算**

我们现在描述全局优化问题的公式。在高级级别上，程序的执行行为被建模为一个带注释的调用图，并且图和所有注释都由MAUI分析器作为求解器的输入提供（上一节描述了如何估计这些输入参数）。在延迟的限制下，我们使用一个线性程序求解器来寻找最优的分配策略，以最小化智能手机消耗的能量。为了减轻程序员的负担，我们提供了一个默认的延迟约束：总执行延迟L不能超过网络延迟的5%。也就是下面的0-1线性规划式子。



**2.实现与评估**

**2.1涉及的软硬件**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实现 | 手机 | HTC Fuze 智能手机 | | .NET Compact Framework v3.5 | |
| MAUI server | 双核电脑  Windows 7 | 3 GHZ CPU | 4 GB RAM | .NET Framework v3.5 |
| 评估 | * 服务器配备了一个NDIS中间驱动程序，插入数据包排队延迟，以控制智能手机和服务器之间的路径的RTT。 * 使用连接到智能手机电池上的硬件功率计[25]来测量手机上的能量。 | | | | |

**2.2评估对象**

1. 一个人脸识别应用程序
2. 一个高度互动的视频游戏
3. 一个国际象棋游戏。
4. 一个基于语音的实时语言翻译器（西班牙语到英语）。

前三个应用程序都是既可以运行在手机上也可以运行在电脑上的。虽然这些应用程序相对简单，但它们每个都包含10个可远程方法。

语言翻译器必须使用远程执行才能在智能手机上运行，因为它的内存资源要求超过了智能手机在本地提供的要求

**2.3宏基准**

**2.3.1MAUI为移动应用程序节省了多少能源？**

自变量：对前三个对象；原始方法与不同网络情况下使用MAUI方法

因变量：消耗的能量（为了体现效果，当运行超过3G时强行远程运行）

**2.3.2MAUI提高了多少移动应用程序的性能？**

自变量：对前三个对象；原始方法与不同网络情况下使用MAUI方法

因变量：执行的间隔时间

**2.3.3MAUI可以运行资源密集型的应用程序吗？**

对对象四进行实验

自变量：时间

因变量：CPU1、CPU2、RAM消耗量

**2.4微基准**

**2.4.1MAUI的求解器的开销是什么？**

我们使用了优化器来记录解决整数线性规划（ILP）问题的每个实例所需的多长时间。

**2.4.2MAUI以全视野来确定是否卸载程序是有必要的吗？**

* 构建了一个朴素的求解器作为MAUI的替代方案：只有当一个方法的远程执行（即传输其状态的成本）消耗的能量比其本地执行更少的能量时，它才会被卸载。
* 对朴素求解器与MAUI求解器运行时的能量消耗进行了对比

**2.4.3状态增量在减少MAUI的数据传输开销方面效果是否有效？**

对比使用状态增量与不使用状态增量两种方法在迭代次数增加的情况下的状态转移的量

**2.4.4 MAUI求解器是否能够适应不断变化的网络条件和CPU成本？**

1)如何测量：

扩展了游戏的功能，使用了一个场外的物理引擎。由于物理计算是CPU密集型的，当导弹的数量变大时，CPU成本显著增长。

2)什么时候测量：

我们在两种特定场景下检查了求解器的行为： 在游戏开始后不久，当没有主动发射导弹时；在游戏运行一段时间后，敌人最多可以发射5枚主动导弹。

1. 本质：

两个变量：CPU计算量（与总计算时间相关）、网络延迟

根据求解器0-1线性规划的限制：总计算时间不超过网络延迟的L%，如果观察到卸载策略根据这两个变量的不同而发生变化就能够说明可以适应。

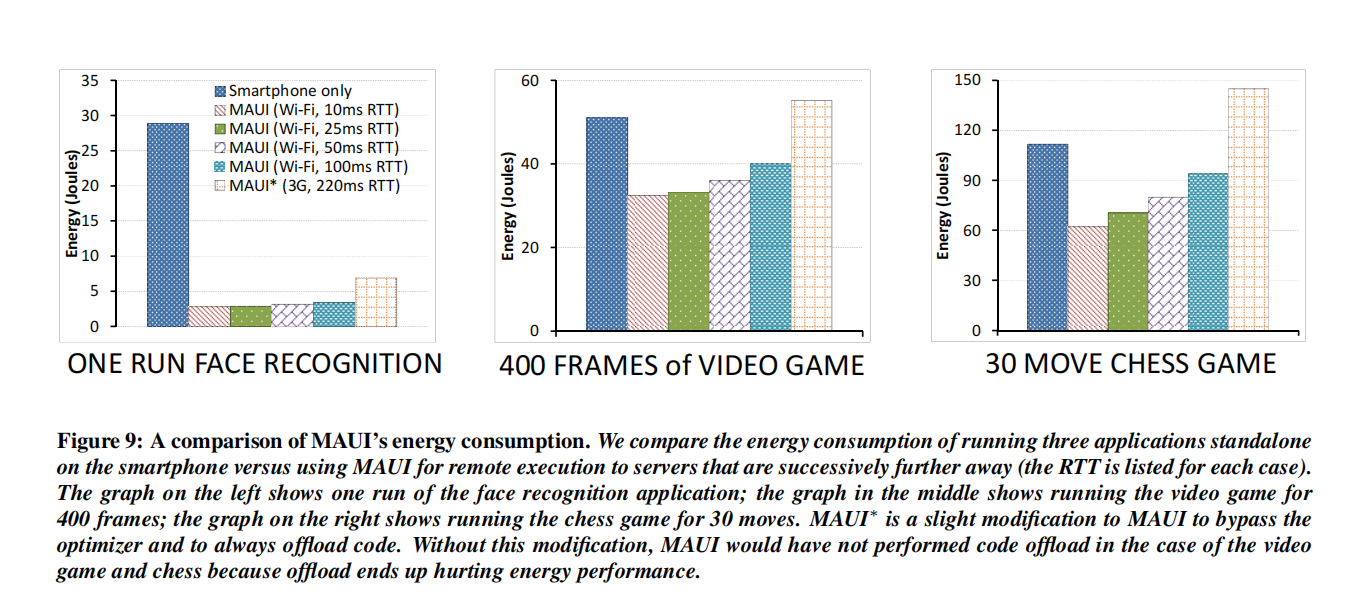
# Q7用于定量评估的数据集是什么？代码有没有开源？

代码：https://github.com/dotnet/maui

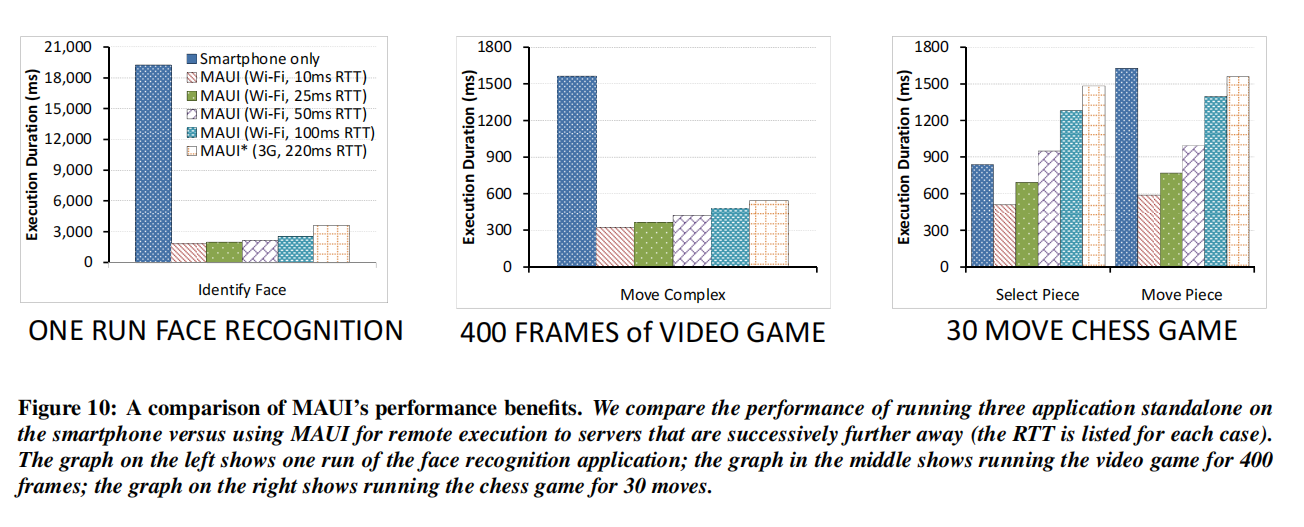
# Q8论文中的实验及结果有没有很好地支持需要验证的科学假设？

**2.3宏基准**

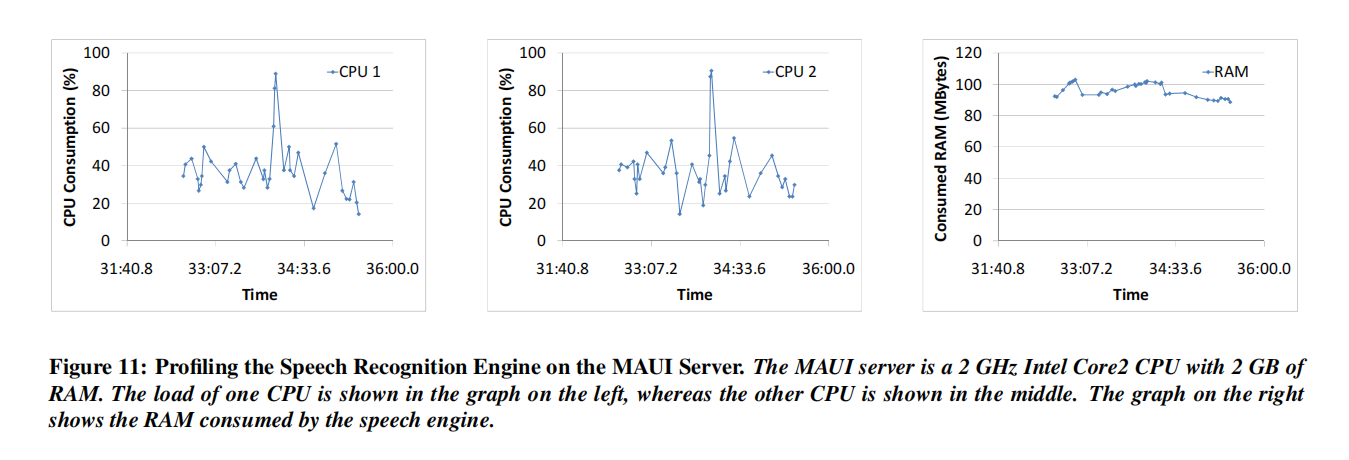
**2.3.1MAUI为移动应用程序节省了多少能源？**



**2.3.2MAUI提高了多少移动应用程序的性能？**



**2.3.3MAUI可以运行资源密集型的应用程序吗？**

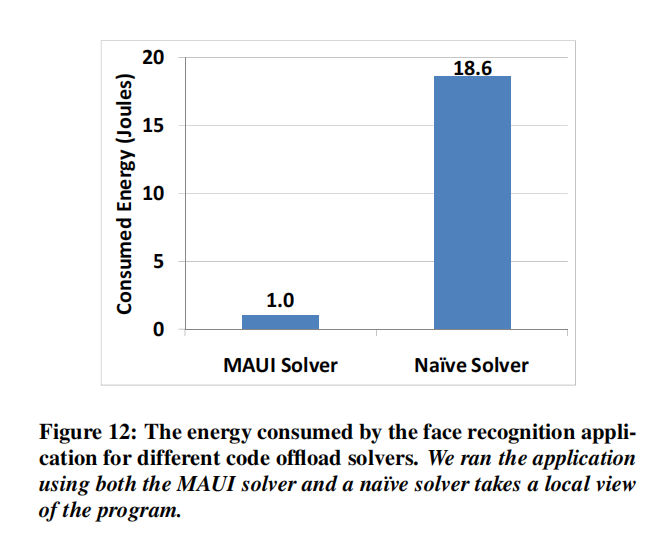


**2.4微基准**

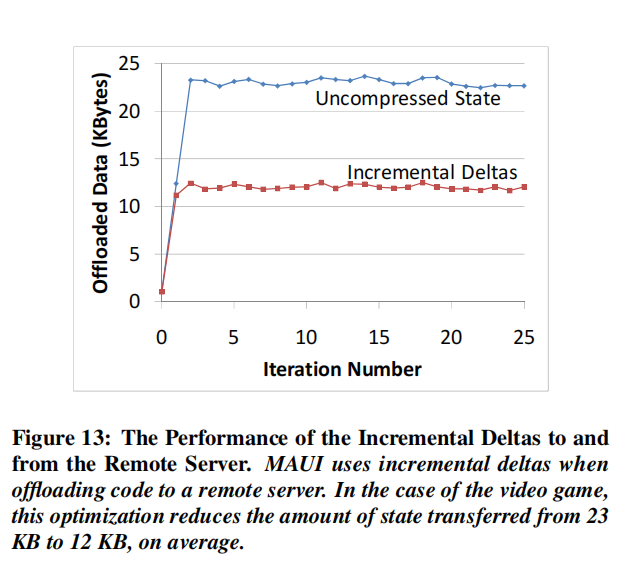
**2.4.1MAUI的求解器的开销是什么？**

求解器平均需要18 ms来求解象棋应用的调用图，而视频游戏需要46 ms。

**2.4.2MAUI以全视野来确定是否卸载程序是有必要的吗？**



**2.4.3状态增量在减少MAUI的数据传输开销方面效果是否有效？**



**2.4.4 MAUI求解器是否能够适应不断变化的网络条件和CPU成本？**

当没有导弹被激活时，只有当到MAUI服务器的往返延迟小于10 ms时，MAUI才会卸载DoLevel方法。当延迟大于10 ms时，将不卸载任何方法。在游戏进行后，敌人有5枚主动导弹，当延迟小于30 ms时，MAUI将卸载DoLevel方法，当MAUI服务器的延迟在30 ms到60 ms之间时，它只会卸HandleMissiles方法，不会卸载DoLevel方法。这是因为卸载HandleMissiles方法所需的状态转移比卸载DoLevel方法所需的要小得多。最后，当延迟超过60 ms时，求解器决定不卸载任何东西。

# Q9这篇论文到底有什么贡献？

MAUI利用托管代码的好处来减少程序员处理程序分区的负担，同时最大限度地提高卸载代码的能源效益。

# Q10下一步呢？有什么工作可以继续深入？

文章中并没有具体地表明这一个问题，但根据MAUI表现不好的地方，或许下一步可以寻找PMS在优化性能与节省能源上的平衡。