物联网中压缩感知算法的云加速方法

张永平，张功萱，朱昭萌

(南京理工大学计算机科学与工程学院 南京 210094)

【摘要】为了减少采集的数据量，提出在物联网中引入“边采样边压缩”的新型采样方法——压缩感知。针对压缩感知理论中信号重建算法计算复杂度较高的问题，设计并实现了一个基于云平台和代码迁移的算法加速方案；该方案解决了代码并行化的自动翻译、算法向云端迁移、本地和云端执行同步等问题，对可并行化的算法，仅需要增加几个新定义的接口及插入一些描述性的注释，就可以利用云资源实现算法的加速；实验表明，该方案是可行的、有效的。该文还研究了基于物联网资源的云加速方法，提出了基于云加速方案、结合多核/多CPU方法和GPGPU方法，能充分利用已有物联网资源的混合压缩感知算法加速框架，并初步设计了理论运行流程。

关 键 词 算法加速; 云计算; 压缩感知; 物联网; 并行化
中图分类号 TP391 文献标志码 A doi:10.3969/j.issn.1001-0548.2014.03.017

Cloud Acceleration Method for Compressed Sensing in Internet of Things

ZHANG Yong-ping, ZHANG Gong-xuan, and ZHU Zhao-meng

(School of Computer Science and Engineering, Nanjing University of Science and Technology Nanjing 210094)

Abstract In order to reduce the amount of data collected, the compressed sensing (CS), a new sampling method, is used for data acquisition and processing in Internet of Things (IoT). To overcome high computational complexity of CS algorithms, an acceleration scheme based on cloud platform and code migration is introduced in this paper. The scheme solved the automatically translated problem of parallelization code, the migration problem of algorithm, and the synchronization problems of local and cloud. It can use the resources from cloud environment to speed up algorithms by adding several interfaces and inserting some comments. In addition, this paper studies the method of cloud acceleration based on computing resources in Internet of Things, and put forwards an acceleration framework, in combination with multi-CPUs/multi-cores CPU and GPGPU parallelization, to speed up CS algorithm based on the cloud acceleration scheme.

Key words algorithm acceleration; cloud computing; compressed sensing; Internet of Things; parallelization

物联网(Internet of things，IoT)是物物相连的网络[1]，它通常配置了为数众多的数据采集设备，这些设备时刻都在产生着大量的采集数据。海量采集数据的存在严重影响了物联网的性能，研究“边采样边压缩”的采样方法已成为物联网应用的一个重要前提。压缩感知(compressed sensing，CS)[2]就是一种将数据采集和压缩同时完成的新型采样方法，可以在对信号采样的同时丢弃冗余信息，从而降低采集的数据量。

在压缩感知理论中，信号的重构是通过求解数值优化问题实现的，算法计算复杂度很高。如利用著名的压缩感知算法——BP(basis pursuit)算法重构信号时，当原始信号的长度为8 192时，重构操作的计算规模相当于求解一个8 192×262 144的线性规划问题[3]。算法的高计算复杂度会对其应用带来较为不利的影响，快速压缩感知方法的研究一直都是该领域的一个热点。

在物联网数据采集和处理中引入压缩感知方法，会减少采集的数据量、降低通信负担，但同时也会带来计算量的增加，这对物联网来说是非常不利的(特别是对实时性要求较高的物联网)。为了加速压缩感知算法，在物联网环境中，云计算技术[4]是一个很好的选择。云可以方便地利用网络中已有的计算资源来加速算法，这对物联网(特别是私有物联

收稿日期：2013－08－29；修回日期：2014－02－24
基金项目：江苏省973项目(BK2011022)；国家自然科学重点基金(61170035, 61272420)
作者简介：张永平(1979－)，男，博士生，主要从事物联网、压缩感知、云计算和分布式计算等方面的研究.

网)有着巨大的吸引力。此外，多核CPU/多CPU[5]和GPGPU(general purpose GPU)[6]并行计算也是很有效的加速方法，物联网中常存在一些这样的高性能计算设备。综上所述，研究以云计算技术为基础，综合使用多种加速方法，充分利用物联网中各种计算资源加速压缩感知算法非常有意义。

1 压缩感知理论

压缩感知[7-8]方法可以在某些基框架下用远少于Nyquist定理规定的测量值表示稀疏的或可压缩的信号，并在需要时能够从这些较少的测量值中重构原始信号。

一般地，根据稀疏表示理论，可假设信号 $s_{n\times 1}$ (为了说明的简便，假设信号是一维的)有某个稀疏表示 θ(θ是可压缩的)：

$$s=\boldsymbol{\Psi}\theta \quad 或 \quad \theta=\boldsymbol{\Psi}^{\mathrm{T}}s \tag{1}$$

式中，$\boldsymbol{\Psi}_{n\times n}$是$s$的稀疏变换基。然后利用观测矩阵 $\boldsymbol{\Phi}_{m\times n}$($m\ll n$)对系数$\theta$进行压缩：

$$y=\boldsymbol{\Phi}\theta=\boldsymbol{\Phi}\boldsymbol{\Psi}^{\mathrm{T}}s \tag{2}$$

这里的$\boldsymbol{\Phi}$是特别设计的，它必须与$\boldsymbol{\Psi}$线性无关；原始信号s与观测数据y之间的压缩比为n:m。

重构信号即求解式(2)，由于$m\ll n$，欠定方程式(2)无法直接求解。根据文献[3]的研究，当$\boldsymbol{\Phi}$与$\boldsymbol{\Psi}$线性无关时，可以通过求解1-范数数值优化问题，即：

$$\min\|\theta\|_1 \quad \text{s.t.} \quad y=\boldsymbol{\Phi}\theta=\boldsymbol{\Phi}\boldsymbol{\Psi}^{\mathrm{T}}s \tag{3}$$

获得s的优化近似解$\hat{s}$，式(3)是一个凸优化问题，可以利用优化方法来求解。

2 云加速方案

优化算法的计算复杂度一般都很高，通常压缩感知方法中的信号的重构时间很长。本节研究利用云技术加速压缩感知算法的设计方案。

2.1 方案设计

云计算技术可以通过网络为用户分配计算资源、提供按需服务，以达到用户满意的加速效果。OpenStack是一个流行的云服务框架，易于实现并具有良好的可扩展性，可以部署、运行于标准的硬件平台。压缩感知算法的云加速方案就是基于OpenStack架构的，它可以利用云平台为用户提供适量的计算资源加速Python语言[9]编写的算法。云加速方案之所以面向Python语言，一是因为Python语言功能足够强大，二是因为OpenStack本身是用Python语言开发的。

在云加速方案中，采集的数据通常以压缩的形式存在。当用户要求重构信号时，计算复杂度不高的一般代码将在本地执行，而计算复杂度较高的重构算法将被自动迁移到云端；在云端，云平台根据用户的要求(如需要计算资源数量、算法运行时间的限制等)提供适量的计算资源，以实现算法的加速服务，并返回最终结果。计算的结果可以是重构的原始信号，也可以是根据重构的信号获得的一些结论。

云加速方案需要解决两个问题：代码的迁移及并行化处理。将计算复杂度较高的代码迁移到云端可以利用云环境中丰富的计算资源，而并行处理是云计算的前提。

2.2 算法迁移

云加速设计方案的算法迁移是基于函数的，它定义了一个“*#remote*”的标签，如果某个函数被该标签标记，则该函数将被自动迁移到云端执行。如下面代码中的Mig函数。

```
#remote
def Mig(参数):
    函数体
```

一般地，函数的迁移有两种实现方法：1) 将函数全部打包并发送；2) 将函数对象和变量序列化后打包并发送。第一种方法易于实现，但会发送很多冗余信息，增加数据传输负担，这与引入压缩感知方法的目的相背。第二种方法传输的数据量较少，但实现复杂，需要充分了解编程语言特性。云加速方案采用第二种方法来实现函数迁移，通过重写用于序列化和反序列化的dumps和loads函数，实现了对Python语言编写的函数迁移，比现有的序列化方法功能更加强大。

2.3 循环的并行化

如果一个循环可以并行处理[10]，就可以使多个操作同时向前推进，从而加快执行速度，但并非所有的循环都可以并行化。在云加速方案中定义了标签“*#parallelize*”，只有被这个标签标记的循环才会并行处理，如以下的for-循环将被并行化：

```
#parallelize
for 1 in …
```

Python语言中，list模式提供了方便的创建列表的方法，而列表很容易实现并行化。云加速方案的并行化思想就是把循环语句转化为list模式，通过设计的专门转换接口，可以自动转换标记了“*#parallelize*”标签的循环语句。

在Python语言中，内嵌函数map常用来构造迭代，但其不适合于并行化，将其重写为pmap函数。

在pmap函数中定义了一个继承list模式的新类，它使程序在迭代时不必等待本次迭代全部结束就可以继续后面的迭代运算，从而实现并行化。

云加速方案是使用Python语言实现的，对Python语言实现的算法，只需要增加一些接口和标签，就能自动并行化指定的循环并将标记的函数迁移到云端进行处理。

3　验证算法及修改

3.1　选择验证算法

在压缩感知理论中，信号重构通过求解数值优化问题实现。根据重构时所使用优化方法的不同，有不同的压缩感知算法。云加速方案的验证，主要使用了BP和OMP(orthogonal matching pursuit)两种算法。BP算法[3]通过基追踪优化方法获得原始信号的全局最优逼近解，算法的重构精度较高，但执行速度较慢。OMP算法[11]通过在每次迭代时得到一个新的近似估计，不断逼近原始信号，最终获得原始信号的局部最优解，算法易于实现且执行速度较快，但重构精度一般。BP算法和OMP算法都是经典的压缩感知算法，BP算法在实验室应用较多，而在工程中OMP算法比较常用，很多压缩感知算法都是从这两个算法延伸而来的。

3.2　算法的修改

为了方便并行化及利用更多的计算资源，云加速方案验证过程中处理的基本数据对象是$m\times n$的信号采样，重构函数(BP函数或OMP函数)的每次调用可以重构信号的一列。这样，对于一个$m\times n$的信号采样，通过调用n次重构函数就可以获得原始信号。为了减少相互之间的关联性，将每次重构函数的调用均作为一次独立操作，不考虑采样数据列之间的关系。对于$n\times 1$、$m\times n\times k$等形式的采样数据，如果需要使用本文方案，可以对数据做简单处理转为$m\times n$形式。

由前面的描述可知，要利用云加速方案实现算法加速，需要为重构函数增添一些说明和注释，主要就是“*#remote*”和“*#parallelize*”两个标签。添加两个标签后的BP算法和OMP的代码形式为：

```
#parallelize
  for i in xrange(n):
      X[:,i]=BP(Y[:,i].reshape((-1,1)),phi,k)
#remote
  def BP(y,phi,k):
      函数体
```

和

```
#parallelize
  for i in xrange(n):
      X[:,i] = OMP(Y[:,i].reshape((-1,1)),phi,k)
#remote
  def OMP(y,phi,k):
      函数体
```

根据云加速方案的设计，当执行for-循环语句时，因前面有“*#parallelize*”标签，系统将同时启动n个进程，每个进程调用一次BP或OMP函数；而函数BP和OMP因前面存在“*#remote*”标签，系统将自动收集函数相关信息并序列化，然后迁移到云端执行，利用在云端申请的计算资源加速算法。

4　实验和分析

4.1　实验环境

云加速方案的验证环境可以分为本地和云端两部分。在本地使用的是PC机，性能参数为Intel(R) Core(TM) 2 Duo 双核处理器(E4600，2.4 GHz)、2 GB内存、64位、10/100 Mb/s网卡，运行Windows 7操作系统。在云端搭建了OpenStack IaaS平台，它管理一个服务器集群，每个刀片服务器性能参数为Intel Xeon四核处理器(2 GHz)、24 GB内存、64位、1 000 Mb/s网卡，运行Ubuntu服务器版操作系统。

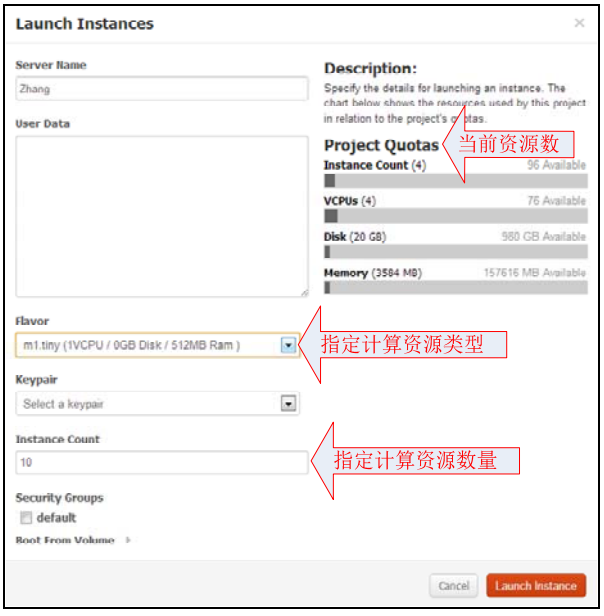


图 1　云加速方案资源申请示意图

在云加速方案中，将云端的计算资源设计为VM实例，每个实例都是相同的，包括1CPU和512 MB RAM。当申请计算资源时，以VM实例为基本单位，每次可以根据用户需求申请一个或多个实例。图1

是申请计算资源时的请求页面，左侧“Flavor”选项可以指定计算资源类型，其中的“tiny 1 VCPU/0 GB Disk/512 MB Ram”即为压缩感知云加速方案设计的实例；“Instance Count”选项是申请计算资源的数量，这需要用户指定；右侧的“Project Quotas”选项中列出了当前用户可申请使用的计算资源总数。

为了验证云加速方案，本文设计了两组实验，分别用于验证方案的正确性和加速效果。在这两组实验中，资源的分配只能在用户启动重构算法时自行指定，即用户需要指明申请的计算资源的数量。在以后的工作中可以考虑由用户设定信号的重构条件(如用户指定算法执行时间的限制)，系统分配符合用户条件的计算资源数量的方法。

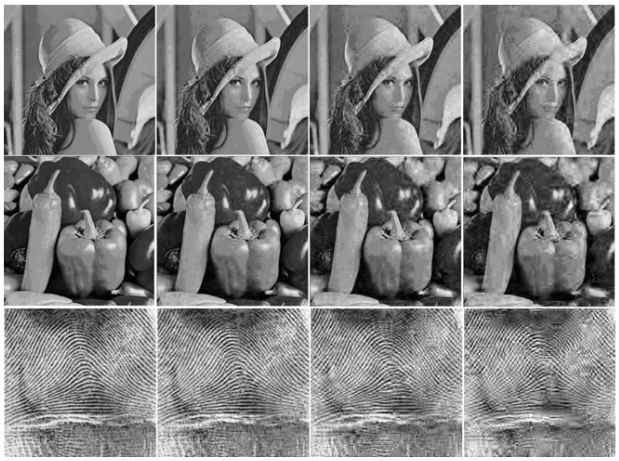
4.2 正确性验证

在云加速方案的设计中，将算法从本地迁移到云端执行，取消了采样数据的列相关性，本组实验是为了验证这些改动并不会影响重构效果。实验中选取了图像处理理论常用的3个标准测试图像Lena.bmp(人物)、Peppers.bmp(自然景色)和Finger.bmp(指纹)，对比其分别在云端和本地重构的效果，如表1和图2、图3所示。

表 1 本地和云端重构图像 PSNR 比较

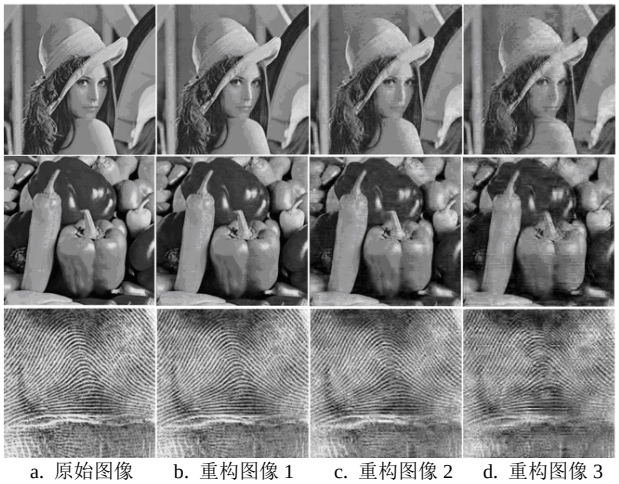
图像	算法	代码执行位置	压缩比(n:m)		
			2:1	3:1	4:1
Lena.bmp	BP	本地	32.066 7	27.921 1	25.096 9
		云端	32.168 7	27.776 8	24.920 9
	OMP	本地	30.070 6	26.526 3	24.196 6
		云端	30.010 6	26.572 3	24.417 0
Peppers.bmp	BP	本地	31.628 7	26.820 2	23.144 6
		云端	31.645 0	26.767 6	23.554 9
	OMP	本地	30.021 8	25.9295	22.115 2
		云端	29.793 8	25.869 6	22.028 6
Finger.bmp	BP	本地	22.666 1	18.594 2	16.981 1
		云端	22.611 8	18.712 1	16.755 8
	OMP	本地	20.028 1	16.882 3	15.657 2
		云端	19.979 3	17.124 4	15.528 7

从表1可以看出，BP算法的重构精度要优于OMP算法，但这两种重构算法的本地执行和云端执行对重构图像的信噪比(power signal to noise ratio，PSNR)影响不大，一些细微的差别可能仅是因为重构时使用的随机观测矩阵不同造成的。从图2、图3可知，BP和OMP算法在云端重构的信号的视觉效果是可以接受的。需要进一步说明的是，3个图像Lena.bmp、Peppers.bmp、Finger.bmp的重构效果有一定的差别，Lena.bmp的重构效果最好、Finger.bmp最差，这不是云加速方案的原因，而是压缩感知算法本身对平滑图像重构能力更好造成的。



a. 原始图像 b. 重构图像 1 (n:m=2:1) c. 重构图像 2 (n:m=3:1) d. 重构图像 3 (n:m=4:1)

图 2 BP 算法的云端重构效果



a. 原始图像 b. 重构图像 1 (n:m=2:1) c. 重构图像 2 (n:m=3:1) d. 重构图像 3 (n:m=4:1)

图 3 OMP 算法的云端重构效果

4.3 加速效果验证

本组实验将用于测试云加速方案对压缩感知算法的加速效果，对重构算法的所有实验分两种情况。

1) 使用256×256大小的人物图像Lena.bmp测试计算资源数量不同时的加速效果，这里申请的计算资源数量从1～16，每一个计算资源数量在不同时间段总计执行100次，求出平均的计算时间和加速比。

2) 在相同的计算资源数量下，分别测试方案对大小为128×128、256×256、512×512、1 024×1 024的Lena.bmp人物图像的加速效果，这里比较了1个、2个、4个、8个和16个计算资源的情况。测试结果如图4(BP算法)和图5(OMP算法)所示，图中“本地”的意思是直接将算法放在一个计算资源上执行，而不需要序列化和代码迁移的过程。

在图4、图5所示的柱状图中，第1列描述算法的执行时间，对比两个柱状图，BP算法的执行时间远远大于OMP算法，这与OMP算法的计算复杂度小于

BP算法是相符的。从图中可以看到，当申请的计算资源数量从1～16不断增加时，BP和OMP两个重构算法的执行时间都逐渐减少，而加速比(柱状图的第2列)逐渐增加，即加速效果越来越好。柱状图的第2～5列描述不同数量计算资源的加速比，它随着计算资源数量的增加而不断增加。当申请的计算资源数量相同时，加速比通常随着信号的增大而逐渐增加，这是因为当信号尺寸比较小时，系统的额外耗费在执行时间中所占的比例比较大。

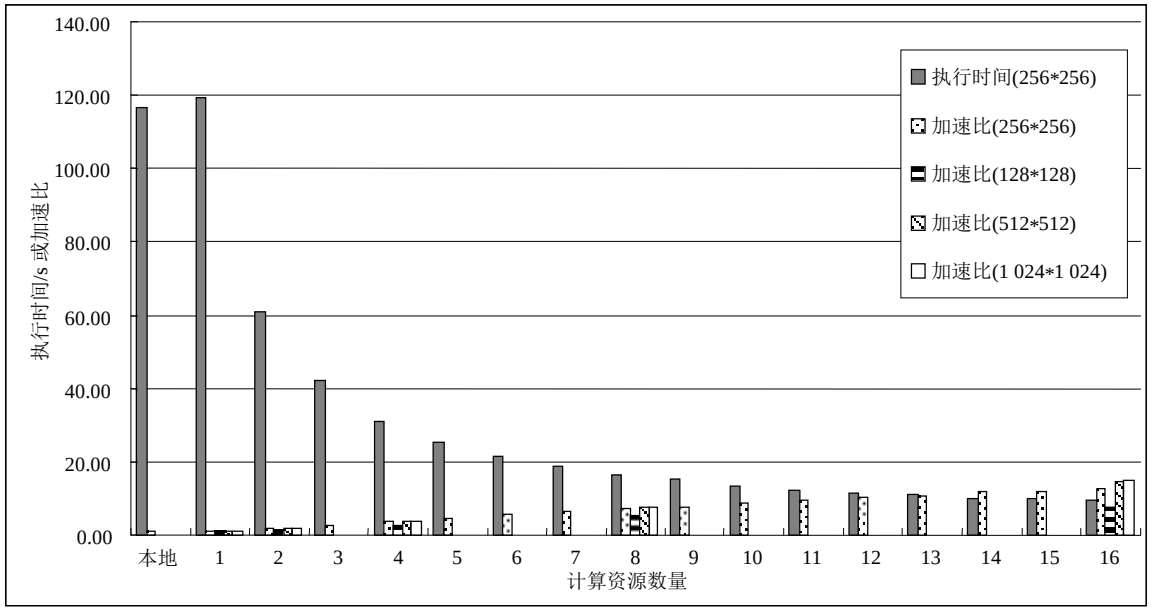


图 4　云加速方案对 BP 算法的加速效果

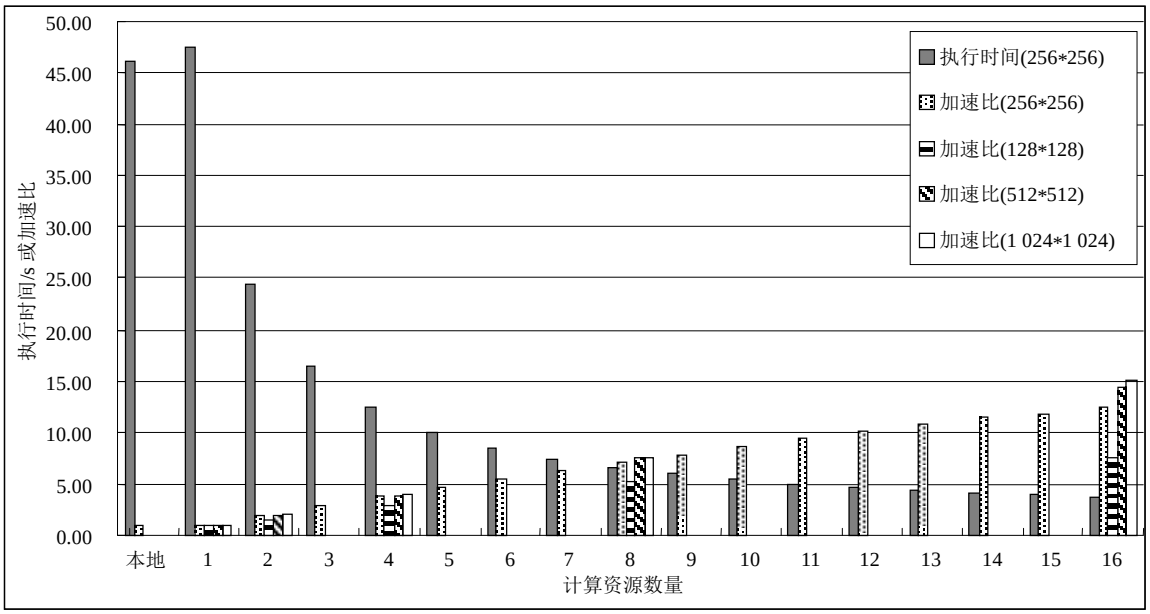


图 5　云加速方案对 OMP 算法的加速效果

5　物联网中云加速方案的应用

5.1　多核/多CPU、GPGPU加速和云加速

通过4.2和4.3节的实验可知，云加速方案可以正确地重构原始信号、并极大地提高重构算法的执行速度。但云加速方案设定每个计算资源是单CPU，在方便实现的同时没有考虑物联网中计算资源的多样性。

事实上，在物联网中同样会存在一些高性能计算资源，如多核/多CPU和GPGPU等系统或设备，而当前硬件设备价格的不断下降为高性能计算设备的增加带来了福音。通常申请一个这样的高性能计算

设备，就能为算法带来很好的加速效果。

多核/多CPU方法实现简单，仅需要对算法做一些简单的并行化处理就可以实现算法加速；不需要远距离传输数据，节省了额外耗费；而且设备计算能力利用率比较高，加速能力强。GPGPU系统的计算核心数量都很大，可以同时启动更多的线程，平均每个核心的花费较低。如NVIDIA Tesla C2050就拥有448个核心，理论上可同时运行448个线程。文献[12-13]分别研究多核/多CPU和GPGPU并行加速的方法，获得了一些实验数据，发现在云加速方案中引入多核/多CPU和GPGPU可以带来加速效果的进一步提升，同时可以为方案提供的服务带来多样性，为云加速方案走出实验室带来极大的好处。

5.2 物联网中基于云的混合加速方案

多核/多CPU方法实现简单、资源利用率高，GPGPU系统的计算资源众多，云技术能够很好地利用现有计算资源、提高资源利用率，这使得它在需要更多计算资源时不用追加投资购买新的软硬件资源。如果能结合3种加速方法的优点，可以获得更好的实用效果，既获取更好的加速效果，又节省资金。图6设计了一个结合3种加速方法的压缩感知算法加速框架，基于物联网环境的该框架以云加速方案为基础，可以结合多核/多CPU和GPGPU加速方法，充分利用物联网中的已有计算资源加速压缩感知算法。

在图6中，假设传感器支持压缩感知方法，为了获取连入网络中物体的实时信息，这些传感器将不断地产生以压缩形式保存的采集数据。一般情况下，这些数据通过无线网络被发送到数据存储中心保存。当有用户申请使用某些采样数据时，这些数据将被重构为原始信号。为了提高算法的重构速度，系统将会根据用户提出的计算条件分配相应的计算资源加速重构算法。这里的计算条件，可以是用户申请的计算资源数量，也可以是用户所提出的重构信号的条件，如重构时间限制等。分配给用户的计算资源可能是一些PC机，如果用户的需求比较紧急，也可以提供计算能力比较强的高性能计算设备，如接入网络的多核/多CPU或GPGPU计算资源。

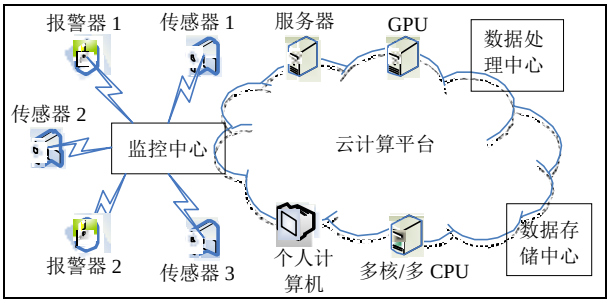


图 6 引入压缩感知后的物联网混合加速理论框架

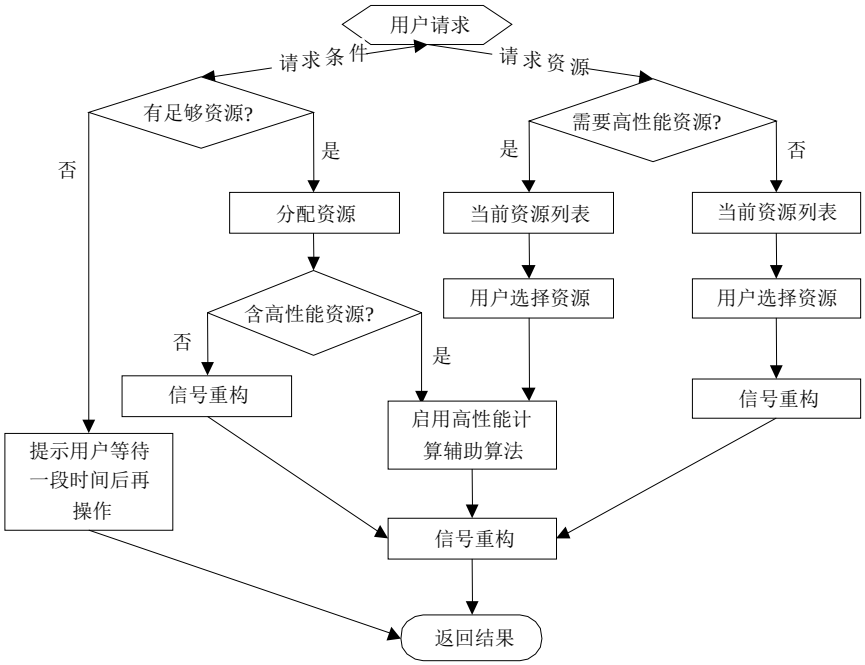


图 7 混合加速框架重构压缩感知的理论流程图

5.3 混合加速方案的重构流程

可以预期，在物联网环境中，建立基于云的混合加速框架，能够更加充分地利用物联网中的计算资源，为压缩感知算法提供更好的加速效果，但是这样会使系统中的计算资源类型变得复杂，不利于计算资源的分配。为此，需要重新设计计算资源的

分配方法及算法流程，图7所示是设计的混合加速方案流程图。

通过图7的流程图可知，当用户申请计算资源时，可以直接指定需要的计算资源，也可以提出重构信号的条件由系统分配。分配计算资源时，如果有必要可以使用高性能计算设备，但需使用一些辅助方法，这些方法在文献[12-13]中已有介绍。

6 结 论

本文将压缩感知方法引入物联网的数据采集和处理，设计并实现了一个利用云技术加速压缩感知算法的方案。该方案可以加速Python程序，通过一些定义的标签和重写的Python语言内嵌函数，自动地将用Python语言写的算法迁移到云端执行，利用云资源加速算法。为了充分利用物联网中现有的计算资源，在将压缩感知算法的云加速方案搭建于物联网时，提出了基于云加速方案，使用多核/多CPU或GPGPU加速方法的混合加速理论框架，以期能够在利用压缩感知方法减少物联网中的采集数据规模的同时，获得更快的算法执行速度。当前，混合加速框架还处于理论设计阶段，接下来将研究其实现方法，以检验方案的可行性、实用性。

参 考 文 献

[1] ASHTON K. That ‘Internet of Things’ thing[EB/OL]. [2012-12-01]. http://www. rfidjournal. com/articles/ view?4986.

[2] ELDAR Y C, KUTYNIOK G. Compressed sensing: Theory and applications[M]. Cambridge: UK, Cambridge University Press, 2012.

[3] CHEN S S, DONOHO D L, SAUNDERS M A. Atomic decomposition by basis pursuit[J]. Siam Review, 2001, 43(1): 129-159.

[4] ARMBRUST M, FOX A, GRIFFITH R, et al. A view of cloud computing[J]. Communications of the ACM, 2010, 53(4): 50-58.

[5] PACHECO P. An introduction to parallel programming[M]. San Francisco, USA: Morgan Kaufmann Publishers, 2011.

[6] SANDERS J, KANDROT E. CUDA by example: an introduction to general-purpose GPU programming[M]. Boston: USA, Addison-Wesley Publishers, 2010.

[7] 王妮娜，桂冠，苏泳涛，等. 基于压缩感知的MIMO-OFDM系统稀疏信道估计方法[J]. 电子科技大学学报, 2013, 42(1): 58-62.
WANG Ni-na, GUI Guan, SU Yong-tao, et al. Compressive sensing-based sparse channel estimation method for MIMO-OFDM systems[J]. Journal of University of Electronic Science and Technology of China, 2013, 42(1): 58-62.

[8] CANDES E J, WAKIN M B. An introduction to compressive sampling[J]. IEEE Signal Processing Magazine, 2008, 25(2): 21-30.

[9] CAI X, LANGTANGEN H P, MOE H. On the performance of the Python programming language for serial and parallel scientific computations[J]. Scientific Programming, 2005, 13(1): 31-56.

[10] ALMASI G S, GOTTLIEB A. Highly parallel computing[M]. Redwood, USA: Benjamin-Cummings Publishers Co, Inc, 1989.

[11] TROPP J, GILBERT A. Signal recovery from random measurements via orthogonal matching pursuit[J]. IEEE Transactions on Information Theory, 2007, 53(12): 4655-4666.

[12] ZHANG Yong-ping, ZHNAG Gong-xuan, ZHU Zhao-meng, et al. Study on parallel compressed sensing for mass data in Internet of Things[C]//PDPTA’12. Las Vegas: CSREA Press, 2012: 571-576.

[13] YADAV K, MIYYAL A, ANSAR M A, et al. Parallel implementation of compressed sensing algorithm on CUDA-GPU[J]. International Journal of Computer Science and Information Security, 2011, 9(3): 112-119.

编 辑 漆 蓉