第 19 卷 第 1 期 上海大学学报(自然科学版) Vol. 19 No. 1
2013年2月 JOURNAL OF SHANGHAI UNIVERSITY (NATURAL SCIENCE) Feb. 2013
DOI: 10.3969/j.issn.1007-2861.2013.01.001

云计算中的弹性算法: 概要和展望

郭毅可[1,2], 韩 锐[1]
(1. 上海大学 计算机工程与科学学院, 上海 200444; 2. 帝国理工学院 计算机系, 伦敦 SW7 2AZ)

摘要: 近年来, 云计算已经成为一种支持按需(on-demand)提供计算资源的低成本传输模式. 在云平台上, 弹性的资源使用是一种基于“现用现付”(pay-as-you-go)的商业模式, 通过“按需”的原则来提供弹性的资源. 介绍一种新的弹性算法(elastic algorithm, EA), 即算法本身就是通过“现用现付”的方式组织起来. 在传统算法中, 计算是一个确定过程, 只会产生一种完整的结果或者没有结果. 与之对比, 弹性算法会随着资源的消耗而生成一组近似结果. 具体来说, 随着消耗的资源越来越多, 弹性算法能够保证产生更好质量的结果. 在这个意义上, 算法产生结果质量的好坏依赖于资源消耗的多少, 因此是具有弹性的. 最后, 正式定义弹性算法的必要性质, 并对弹性算法未来的研究方向进行展望, 提出一系列的研究挑战.

关键词: 云计算; 现用现付; 弹性算法
中图分类号: TP 301 **文献标志码:** A **文章编号:** 1007-2861(2013)01-0001-04

Elastic Algorithm in Cloud Computing: Overview and Prospect

GUO Yi-ke[1,2], HAN Rui[1]
(1. School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China;
2. Department of Computing, Imperial College London, London SW7 2AZ, UK)

Abstract: In recent years, cloud computing has emerged as a cost-effective way to deliver on-demand and metered computing resources. In a cloud, elasticity of resource usage is typically realized through the “on-demand” provision principle supported by the “pay-as-you-go” business model. However, little has been investigated into elasticity of algorithm for cloud computing. This paper introduces a novel elastic algorithm (EA) in which the computation itself is organized in a “pay-as-you-go” fashion. In contrast to conventional algorithms, where computation is a deterministic process that only produces an “all-or-nothing” result, an EA generates a set of approximate results corresponding to its resource consumption. As more resources are consumed, better results can be derived. In this sense, quality of the algorithm is elastic to its resource consumption. The desirable properties for EA are formalized, and ambitious agenda for future research is provided in this area and propose several challenges.

Key words: cloud computing; pay-as-you-go; elastic algorithm (EA)

当前, 云计算已经成为一种支持按需提供计算资源的低成本传输模式. 云计算的重要性及其对社会的影响力是能够预见的. 在云平台上, 计算资源(硬件资源和软件应用)是作为一种资源提供给多个用户共享的. 在“现用现付”的商业模式下, 每个用户根据需要被提供一份能够单独占用和使用的资源. 弹性的资源则意味着依据“按需”原则, 分配给一个用户的资源可以根据其需求动态变更. 在这个意义上, 弹性算法对于云计算模型是不可或缺的, 因为它不仅提供了一个机制, 将有限的资源提供给无限制数目的用户分享使用, 更为实现云计算“现用现付”的商业模式打下了基础.

当前云平台上的弹性, 一直被视为支持“现用现付”模式中资源管理的一个特性. 在传统的观点中, “现用现付”是由一个算法所执行的次数来衡量的, 并没有考虑算法本身具有的弹性. 因此, 使用传统算法可以解决诸如“在特定的时间, 我们需要支付多少费用才可以得出结果?”这样的问题, 但并没有回答诸如“给予一定的预算, 从我的算法中我能得到什么样的结果?”这类问题. 在传统算法中, 这类问题甚至会被认为是错误的, 因为传统算法只能给出一个确定的结果或者没有结果, 算法的

收稿日期: 2012-11-28
通信作者: 郭毅可(1962—), 男, 教授, 博士, 研究方向为云计算. E-mail: y.guo@imperial.ac.uk

结果本身并没有弹性可言. 因此, 当前的云计算主要是商业创新, 而不是一个新的计算模式, 这一限制已经严重影响了对云计算全部潜力的探索. 例如, 云计算平台上的现货价格模型首先是一种强大的动态定价机制; 其次, 现货价格模型的实际应用需要算法提供所谓的“检查点”, 使得算法的执行可以暂停, 然后安全地恢复; 进一步地, 算法需要在“检查点”提供某种有意义的“近似结果”, 当算法从“检查点”恢复, 则可以以当前结果为起点进行进一步的优化.

本研究介绍了一类新颖的算法——弹性算法. 该算法非常适合于云平台上的“现用现付”商业模型. 理想情况下, 与那些只能通过固定过程产生确定完全结果的传统算法不同, 弹性算法能够产生多个中间的近似结果, 其结果质量的好坏与消耗的资源多少成正比. 从这个角度可以认为, 算法本身就是弹性的. 使用弹性算法, 即便预算有限, 仍然可以保证用户得到可用的结果. 如果有更多的预算, 弹性算法能够为用户提供更好质量的结果. 本研究的目的就是探讨弹性算法所需满足的性质, 并试图从算法层面而不是资源层面对弹性算法进行正式定义.

1 弹性算法的动机

在传统的云计算领域, 弹性被认为是一个按需扩展或缩减资源的管理机制. 具体来说, 它通过提供实时获取/释放计算资源来控制应用程序使用的规模, 满足用户的服务质量(quality of service, QoS)需求, 而用户只需根据资源使用量付费[1-3]. 本工作将改变这个资源导向的视角, 从算法层面来研究弹性的概念. 这要求我们能够形式化地定义算法弹性以及讨论该弹性如何影响算法的运行方式.

多年来, 人们对算法一直有一个简单的理解: 它是一系列的计算步骤, 在消耗一些时间和资源后必定产生一个确定的结果. 因为输出结果是确定的, 一个算法的计算性能通常都是由其计算(时间和空间)的复杂性来衡量的, 而计算的应用性能指标则是由服务质量, 如响应时间和吞吐量来衡量的. 因此, 在当前的服务架构中, 用户通过购买资源来满足所需要的服务质量. 这种类型的应用有一个典型的例子[2]: 在基于多层服务架构的网站应用程序中, 用户的服务质量可以表达为响应时间和/或吞吐量, 应用提供商通常购买更多的资源, 部署更多的服务器, 从而获得较短的响应时间(更多的处理资源, 如CPU和内存)和/或更高的吞吐量(更多的带宽资源). 因此, 资源的使用是随着成本的变化而富有弹性的.

弹性在经济术语中有明确的定义, 它可以衡量一个经济变量的改变是如何影响其他变量的. 弹性y与对应的x被定义为$E_x^y = \frac{\%\Delta y}{\%\Delta x} = \frac{\partial y}{\partial x} \cdot \frac{x}{y}$. 在云计算中, 一个经济模型具有价格弹性, 而一个应用程序(服务)则具有资源弹性. 因此, 在云平台上, 传统算法所解决的一个典型的商业问题是“在特定的时间内, 我需要花费多少钱来得到我想要的结果?”和“我需要花费多少钱来实现每分钟响应1 000个并发请求?”

本研究讨论的一个关键问题是传统的资源弹性是否足够? 钱能否买别的东西而不仅仅是性能或服务质量? 是否可以添加另外一种形式的弹性? 算法的结果本身对资源(或者用户的成本)是否具有弹性? 基于这些问题, 本研究提出了一类新的算法, 能够产生一系列质量正比于资源消耗的近似结果. 随着越来越多的资源被消耗, 更好的结果将随之产生. 以一个图像渲染应用程序为例, 一个常规算法着重于生成的结果具有最高分辨率, 而一个云模型可能更关注于采用增量渲染的方法, 即随着用户购买和使用越来越多的资源, 不断产生更高分辨率的图像. 在这种情况下, 结果的质量(如图像的分辨率)随着资源的使用量而弹性变化. 可以将具有上述行为的算法称为弹性算法. 同样地, 弹性算法可以被设计和使用在更广泛的领域, 包括金融、科学计算和工程模拟等.

2 相关工作

本研究提出的弹性算法继承和发展了当前已有的自适应的、灵活的算法. 根据已有的研究, 本算法主要参考了以下两个方面的工作.

(1) 资源感知算法(cost-aware algorithms)[4-5]. 在很多设备中, 资源是有限的, 比如移动设备、传感器. 一直以来, 资源感知算法[5]常用于这些资源有限的设备, 以控制算法的行为和执行. 这类算法基于可用的资源来灵活调整其输入、输出和/或处理函数. 资源感知算法有3个关键的策略: ①控制算法的输入粒度, 例如改变分辨率或输入数据的结构; ②控制算法的处理粒度, 例如增加或减少执行的次数; ③控制算法的输出粒度, 例如控制分辨率或输出数据的结构.

资源感知算法需要一个资源监控器和决策系统(如一套规则)在3种策略中作出选择, 从而允许用户在算法输出结果的质量和可用资源间作出取舍. 虽然这类算法并没有明确定义结果的质量, 但可以从中学习如何控制算法的运行, 在计算结果的质量和使用的资源量之间进行权衡, 利用有限的资源产生尽可能好的结果.

(2) 任意时间算法(anytime algorithms)[6-8]. Dean等[6-7]首先提出了任意时间算法, 并将其应用于预期驱动的迭代优化算法中. 这类算法能够在任何时间返回近似的结果, 并尝试通过更多的运算返回更好质量的结果; 能够定义一个增量式的计算过程, 并随着计算量的增加, 尝试改进当前的结果. Zilberstein[8]扩展了Dean等的工作, 定义了任意时间算法的几个关键特性: ①可被打断和恢复, 任意时间算法可以在任何时候被打断, 并返回一个大

致的近似结果, 此外, 该算法可以稍后重新启动, 并继续其计算; ② 明确和可测量的质量, 任意时间算法提供一个精确的和可量化的质量函数, 在算法运行时对产生结果的质量进行度量; ③ 质量的单调性和改进幅度递减, 随着计算时间的增加, 任意时间算法每次都返回当前的最好结果, 从而保证了算法质量的单调递增, 同时, 随着算法结果越来越接近可能的最优结果, 算法改进的幅度会逐步减少.

任意时间算法可以在产生最终结果前, 通过增量式的方法产生一系列的近似结果, 并保证结果的质量随着计算量的增加而不断改进.

结合上述两类算法, 本工作对弹性算法进行了研究. 在云计算平台中, 本算法可以很好地和“现用现付”的商业模式相结合, 产生一个按质量付费的新的付费模式. 在“现用现付”模式中, 弹性算法的关键问题是研究算法计算结果质量的好坏相对于用户在云平台中购买资源的弹性.

3　弹性算法的定义

弹性范式[9]的提出给用户提供了一个行之有效的方法, 它将算法本身的运行方式组织成一个“现用现付”的方式. 一个弹性算法能够提供给用户多个有意义的近似结果, 并保证结果的质量随着用户投入的增加而不断提高. 具体来说, 在弹性算法中, 每个计算结果的质量都可以用一个量化的函数来度量, 同时保证计算结果的质量和消耗的资源成正比. 进一步地, 弹性算法允许用户从当前结果出发, 通过新的运算来优化算法的质量. 同时, 在运行过程中, 算法可以随时被中止, 且恢复成本很小. 使用弹性算法, 用户即使预算有限, 依然能够保证得到一个有用的结果. 如果用户能够投入更多的运算, 弹性算法又能保证进一步优化现有算法结果的质量. 因此可以认为, 资源感知算法可对计算结果的质量和所需的资源进行权衡, 并产生尽可能好的结果. 同时, 任意时间算法可将确定的计算过程组织成一系列的迭代过程, 并在不断地迭代中, 通过投入更多的资源, 提高计算结果的质量. 需要说明的是, 在任意时间算法中, 计算的过程被严格定义为一个顺序迭代的过程, 而不允许用户灵活地根据自己的预算选择相应质量的算法和运算. 因此, 可以认为任意时间算法是弹性算法的一种特殊情况.

下面正式定义弹性算法的必要性质.

定义 1　(弹性算法的必要性质)一个弹性算法能够产生一系列的近似结果 $\{S_1, S_2, \cdots, S_n\}(n > 1)$, 其质量为 $\{Q(S_1), Q(S_2), \cdots, Q(S_n)\}$. 假定 S_{a} 为算法的初始状况, 弹性算法能够允许用户投入 I_i, 进而产生一个近似结果: S_i = 弹性算法$(I_i, S_0)(i = 1, 2, \cdots, n)$. 算法能够接受用户进一步地投入 ΔI, 并基于当前结果 S_i 产生一个新的结果: S_j=弹性算法$(\Delta I, S_i)$ $(j = i+1, i+2, \cdots, n)$.

弹性算法需要满足下面 4 个性质.

(1) 可度量的质量. 对于任意的计算结果 S, 都有一个可度量的质量函数 $Q(S)$.

(2) 有意义的计算结果. 对于一个任意的计算结果 S, 都有一个有意义的质量度量 $\epsilon \geqslant 0$, 能够保证 $Q(S) \geqslant \epsilon$.

(3) 算法结果的单独递增性. 对于任意的 $i, j \geqslant 1$, 以及计算结果 S_i=弹性算法 (I_i, S_0) 和 S_j=弹性算法(I_j, S_0), 如果 $I_i > I_j$, 那么 $Q(S_i) \geqslant Q(S_j)$.

(4) 计算的累加性. 对于任意的 $i, j, k \geqslant 1$, 以及已有的运算结果 S_i 和 S_j, 设 ΔI_{ik} 为把结果从 S_i 优化成 S_k 所需的投入 S_k=弹性算法$(\Delta I_{ik}, S_i)$, ΔI_{jk} 为把结果从 S_j 优化成 S_k 所需的投入 S_k=弹性算法$(\Delta I_{jk}, S_j)$, 如果 $Q(S_j) > Q(S_i)$, 那么 $\Delta I_{jk} < \Delta I_{ik}$.

本研究用图像渲染算法[10]作为一个典型例子, 讨论弹性算法的 4 个必要性质.

(1) 可度量的质量. 在图像渲染中, 计算结果的质量可以通过图像的分辨率(图像像素)来定量度量.

(2) 有意义的计算结果. 图像渲染算法可以产生一系列的近似结果, 即不同分辨率(如 1×1 像素、10×10 像素或 100×100 像素)的完整渲染图像. 对于任意图像, 最少有 1×1 像素.

(3) 算法结果的单独递增性. 在图像渲染算法中, 当用户加大投入时, 就能期望得到更高分辨率/更好质量的图像. 例如, 花费 1 美元能得到 100×100 像素的图像, 而花费 5 美元就能得到 1 000×1 000 像素的图像.

(4) 计算的累加性. 在图像渲染中, 可以用一个可恢复文件来保存当前的计算结果, 并作为下一次渲染的起点, 从而避免重复渲染已经被渲染的像素点. 假定 S_i 和 S_j 为用户已有的两个图像, 并且这两个图像的分辨率分别为 10×10 和 100×100 像素. 图像渲染算法可以从其中任何一个图像出发, 去渲染得到一个新的 1 000×1 000 像素的图像 S_k. 假定从图像 S_i 和 S_j 开始, 用户分别需要投入 ΔI_{ik} 和 ΔI_{jk} 来获取图像 S_k, 则有 $\Delta I_{ik} > \Delta I_{jk}$, 这是因为图像 S_j 的可恢复文件保存了更多的已经被渲染的像素点, 从而允许渲染算法进行更少的渲染(更少的计算)来得到 1 000×1 000 像素的图像.

开发一个弹性算法并不容易, 因为在实际软件程序中, 许多算法本身并不如图像渲染算法一样能够满足弹性算法的性质. 因此, 将非弹性算法转换成弹性算法有着诸多的挑战: ① 需要一个质量度量函数, 对近似结果的好坏进行评估; ② 传统的非弹性算法有一个确定性的计算过程来产生“完全”的结果, 而将这个确定性的计算过程转换成弹性算法的“现用现付”过程, 并根据用户的投入产生一系列不同质量的近似结果有一定的难度; ③ 最

重要的是, 当用户投入更多的计算资源时, 算法产生的结果的质量必须也要更好, 因为弹性算法的关键性质是保证算法质量的单独递增性.

4 总结和展望

云计算改变了访问和使用计算资源的模式, 实现了按需增加或减少计算的能力, 从而使得利用计算周期计费和消耗资源量计费变成可能. 现阶段所面临的关键挑战是如何编写软件程序, 以便充分利用云模型所具备的资源弹性和价格弹性, 使算法本身也具有弹性.

本研究讨论了弹性的概念, 对弹性算法的属性进行了正式定义, 并通过一个增量式图像渲染的例子进行了说明. 对弹性算法的研究, 仍然是一个新的研究领域, 有许多关键的挑战仍需要解决.

(1) 挑战一: 哪些应用程序可以利用弹性行为? 弹性行为如何能得到支持? 利用本研究提出的弹性范式, 可以在任何近似计算结果具有价值的领域, 包括数值科学计算和工程模拟, 开发和应用弹性算法. 然而, 很多应用程序并不会简单地适用于弹性范式. 例如, 从表面上看, 一个传统的用来计算和发放员工工资的管理程序可能就不适用, 因为支付一半(或部分)工资给员工可能会影响一个企业的正常运营. 对于这个应用程序, 第一个结果必须执行一个强制性的任务, 即支付给员工基本工资. 进一步地, 可以开发更为复杂的工资发放系统, 在不同的粒度上进行工资管理. 例如, 根据员工的工龄和绩效计算和发放工资.

(2) 挑战二: 弹性算法的开发需要编程工具和编程库的支持. 使用具有不同粒度的分层数据结构, 能够有效地支持一个弹性算法的开发. 然而, 弹性算法要成为主流, 需要通过开发数据结构、编程库和相关的工具来简化这一过程.

(3) 挑战三: 如何选择一个合适的弹性质量度量函数? 本研究讨论了支持弹性算法所需的质量度量函数. 在图像渲染的例子中, 使用了图像分辨率作为质量度量函数, 但在更为广泛的应用程序中, 质量的定义可能很难. 设想一个国际象棋游戏, 一系列走法的尝试(探索)未必能改变结果(单位移动), 然而却可以增加获得更好结果的信心, 该信心可能是一个好的质量度量函数.

(4) 挑战四: 什么类型的调度算法最适合弹性算法? 传统的调度算法是用于在可用的资源下在最短的执行时间内完成任务, 并且假设闲置的资源可以免费使用. 在云计算“现用现付”的模式下, 这两个假设可能无法成立. 一个用户宁可节约预算, 也不愿意浪费更多的钱却无法获得足够的收益, 而且资源的价格也会随着时间的改变而波动, 且并非免费的. 当把不同的弹性算法组合在一起, 调度问题会变得更为复杂.

(5) 挑战五: 什么标准适合于比较不同的弹性算法? 当应用多个弹性算法解决同样的问题时, 需要制定一个统一的标准来比较不同的弹性算法. 不同的算法有不同的弹性行为, 因此, 制定一个全面准确的标准变得至关重要.

进一步的工作将在云平台上研发弹性算法, 以期开发出一个全面统一的框架来应对上述挑战.

参考文献:

[1] Han R, Guo L, Guo Y, et al. A deployment platform for dynamically scaling applications in the cloud [C]// The 3rd IEEE International Conference on Cloud Computing Technology and Science. 2011: 506-510.

[2] Han R, Ghanem M M, Guo L, et al. Enabling cost-aware and adaptive elasticity of multi-tier cloud applications [J]. Future Generation Computer Systems, 2012, DOI: 10.1016/j.future.2012.05.018.

[3] Han R, Guo L, Ghanem M M, et al. Lightweight resource scaling for cloud applications [C]// The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. 2012: 644-651.

[4] Poladian V, Sousa J P, Garlan D, et al. Dynamic configuration of resource-aware services [C]// The 26th International Conference on Software Engineering. 2004: 604-613.

[5] Gaber M M, Yu P S. A framework for resource-aware knowledge discovery in data streams: a holistic approach with its application to clustering [C]// The 2006 ACM Symposium on Applied Computing. 2006: 649-656.

[6] Dean T L. Intractability and time-dependent planning [C]// The 7th National Conference on Artificial Intelligence. 1989: 245-266.

[7] Dean T, Boddy M. An analysis of time-dependent planning [C]// The 7th National Conference on Artificial Intelligence. 1988: 49-54.

[8] Zilberstein S. Using anytime algorithms in intelligent systems [J]. AI Magazine, 1996, 17: 73-83.

[9] Guo Y, Ghanem M, Han R. Does the cloud need new algorithms: an introduction to elastic algorithms [C]// The 4th International Conference on Cloud Computing. 2012: 66-73.

[10] Pharr M, Humphreys G. Physically based rendering: from theory to implementation [M]. San Fransisco: Morgan Kaufmann, 2004: 1-43.