云计算环境下低成本存储科学数据的演化 CTT-SP 算法

郭 梅[1] 袁 栋[2] 杨 耘[3]
(广东工业大学计算机学院 广州 510006)[1] (悉尼大学电力信息工程学院 悉尼 2006)[2]
(斯威本科技大学软件与电子工程学院 墨尔本 3122)[3]

摘 要 云计算系统强大的计算能力和存储容量,使得科学家可以在其上部署计算型和数据密集型的应用,并把大量的应用数据存储在云计算环境下。基于云服务即用即付模型,针对原有数据存储状态,考虑云服务价格变化所产生的状态调整成本,同时为降低存储大量生成的科学数据的成本,在传统最小成本基准的 CTT-SP 算法的基础上,提出了一种演化 CTT-SP 算法。在云计算环境下针对云服务的新价格,该算法可自动决定所生成的科学数据是否需要存储,从而使计算和存储达到更佳的平衡。以亚马逊的成本模型为例,对大量随机数据集进行实验,结果表明,当云服务价格变化后,所提演化 CTT-SP 算法有效地降低了存储科学数据的总成本。

关键词 数据存储,计算存储平衡,云计算,科学应用,成本
中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.12.031

Evolutionary CTT-SP Algorithm for Cost-effectively Storing Scientific Datasets in Cloud

GUO Mei[1] YUAN Dong[2] YANG Yun[3]
(School of Computers, Guangdong University of Technology, Guangzhou 510006, China)[1]
(School of Electrical and Information Engineering, The University of Sydney, Sydney 2006, Australia)[2]
(School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne 3122, Australia)[3]

Abstract Massive computation power and storage capacity of cloud computing systems allow scientists to deploy computation and data intensive applications in the cloud, where large application datasets can be stored. Based on the cloud service's pay-as-you-go model, taking the status adjustment cost caused by cloud service's price changes into consideration for the original datasets storage status, we proposed an evolutionary CTT-SP algorithm based on the traditional minimum cost benchmarking CTT-SP algorithm for cost-effectively storing large volume of generated scientific datasets in the cloud. The algorithm can automatically decide whether a generated dataset should be stored or not in the cloud, and also achieve better trade-off between computation and storage at the new price. Random simulations conducted with Amazon's cost model show that the proposed evolutionary CTT-SP algorithm can save the overall cost of storing scientific datasets significantly when the cloud service's price changes.

Keywords Datasets storage, Computation-storage trade-off, Cloud computing, Scientific application, Cost

1 引言

近年来,云计算作为新兴的并行分布式计算模式,根据用户需求,提供冗余、廉价和可扩展的资源[1]。云计算的出现,提供了一种新的部署科学应用程序的方式。基础设施即服务(Infrastructure as a Service, IaaS),是云计算环境下一种非常流行的用来提供云服务的方式[2],其通过虚拟化技术隐藏云服务提供商的计算机系统的异构性[3]。因此,不需要购买昂贵的软硬件等基础设施,科学家可以把他们的应用程序部署在统一的云资源中,并根据云资源即用即付(pay as you go)的费用模式来支付相关成本。

相比于传统的计算机系统(如集群、网格和超级计算机系统),按需的云计算服务给科学研究部署带来了便利[4],但科学研究者为使用的云资源所支付诉成本仍然很高。特别地,科学应用越来越趋向于数据密集型的应用[5],其产生的数据大小通常是 GB 级或者 TB 级的,甚至是 PB 级规模的。这些数据包含着计算中的重要中间数据结果和最终结果,可能会被重用或共享[6],进而需要对数据进行存储。因此,在云计算环境下基于即用即付的费用模式部署科学应用程序,对于决定是否存储产生的科学数据,从而保持存储成本和重新生成所耗的计算成本的相对平衡,最终降低存储科学数据的总成本,是一个重要的问题。

目前,国内几乎没有关于云计算环境下计算和存储之间的平衡问题的研究,而国外已有大量的相关研究。Nectar 系

到稿日期:2016-09-29 返修日期:2017-01-15 本文受广东省产学研重点项目(2014XYD-007),广东省科技计划项目(2012B091000173)资助。
郭 梅(1992-),女,硕士生,主要研究方向为大数据、云计算,E-mail:gmei@sina.cn;袁 栋(1983-),男,博士,讲师,主要研究方向为云计算、数据存储、软件工程;杨 耘(1963-),男,博士,教授,博士生导师,主要研究方向为软件技术、云计算、服务计算。

统[7]面向数据中心内数据进行自动管理和计算，并删除过时的数据集，当需要再次使用这些数据时再重新生成，从而提高了资源利用率。根据文献[8]，相比于每次总是从最初的输入数据再生，适当地存储一些重要的中间数据可以节省成本。文献[9]提出了一个表示计算成本和存储成本平衡的模型，但没有给出任何策略来找到这种平衡。文献[10-11]提出了基于成本率的数据存储策略，通过比较数据自身的存储成本率与重新生成的计算成本率的大小，来决定是否存储该数据。文献[12]提出了最小成本基准的 CTT-SP 算法，采用该算法的数据存储策略可找到计算和存储之间的最佳平衡。基于该 CTT-SP 算法，文献[13-14]提出了局部最优的数据存储策略，其复杂度更低，成本效益相对较高。面向应用数据可能分布在固定位置，需要以一种分布式的方式运行，文献[15]提出了 GT-CSB 算法以寻求成本更小的数据存储策略。文献[16]提出了基于解空间划分方法的高效算法，该算法能动态实时地找到最小成本的数据存储策略。

上述研究均是针对初始数据存储状态(所有数据均存储)来寻求最小成本的存储策略，而不是考虑云服务价格变化，针对原有价格下的数据存储状态来寻求最小成本的存储策略。当云服务价格变化时，针对原有的数据存储状态，上述的传统基准策略可能不是最小成本的存储策略。对于如图 1 所示的初始数据集$\{d_1,d_2,\cdots,d_8\}$，当价格由 *Price*1 变为 *Price*2 时，由传统基准策略可以得到新的数据存储状态 *Price*2_*Set*，从旧的数据存储状态 *Price*1_*Set* 调整到新的数据存储状态 *Price*2_*Set*，可能存在某些具体的数据需要被重新生成，从而产生一次性的状态调整成本 Δ。这种状态调整成本实际上是一次性的计算成本，应计入在新价格下的总成本中，而不应该被忽略。因此，考虑云计算环境下的价格变化，结合状态调整成本，基于传统基准策略的 CTT-SP 算法[12]，设计一种演化的 CTT-SP 算法，从而提出新的存储策略，得到新价格下总成本更小的数据存储状态 *Price*2_*Set*′，节省了新价格下存储科学数据的总成本。

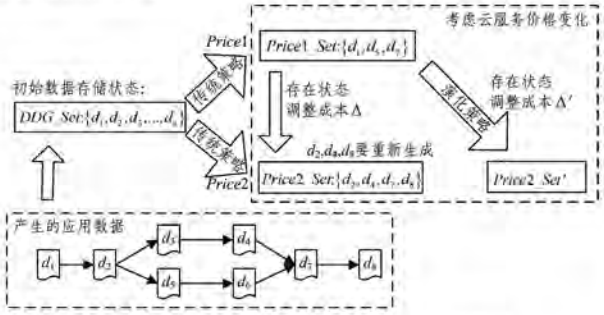

图 1 云服务价格变化所产生的状态调整成本

2 相关前提及定义

2.1 应用数据的分类

通常，云计算环境下存储的应用数据类型主要有两种：原始数据和生成的数据。

1)原始数据，是指用户上传的数据。在科学应用程序中，它们通常是指从实验设备收集到的最原始的数据，并由用户决定这些数据是否应该存储或删除。这些数据一旦被删除，就无法从系统重新生成。存储原始数据的成本是固定的，故原始数据不在考虑范围之内。

2)生成的数据，是指云计算系统中应用程序执行时新产生的数据，是可备用的中间或最终计算结果，可以由系统决定这些数据是否存储。若知道每个数据的起源数据，就可以重新生成该数据。所提的数据存储策略只适用于生成的数据，自动决定应用程序中所生成的数据集的存储状态。因此，将生成的数据作为数据集。

2.2 数据依赖图

采用数据依赖图(Data Dependency Graph，DDG)表示生成的数据集，以及数据之间的关系。DDG[12]是科学应用程序中基于数据起源的有向无环图，描述了数据集的生成关系，对于已删除的数据集，可以由其最近的现有的前驱数据集重新生成。图 2 描述了一个简单的 DDG，其中每个节点代表一个数据。假定数据 d_i 在 DDG 中表示为 $d_i \in \mathrm{DDG}$。d_1 指向 d_2，说明数据 d_1 可以生成数据 d_2；d_2 指向 d_3 和 d_5，说明数据 d_2 通过不同的运算可以生成数据 d_3 和数据 d_5；d_4 和 d_6 均指向 d_7，说明数据 d_4 和数据 d_6 可以共同生成数据 d_7。

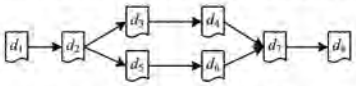

图 2 一个简单的数据依赖图

为了更好地描述数据集在 DDG 中的关系，定义符号“→”，表示两个数据之间有生成的关系，其中 $d_i \rightarrow d_j$ 表示 d_i 是 DDG 中数据 d_j 的前驱数据。例如，在图 1 的 DDG 中，有 $d_1 \rightarrow d_2$，$d_1 \rightarrow d_4$，$d_5 \rightarrow d_7$，$d_1 \rightarrow d_7$ 等。此外，→是可传递的，即：$d_i \rightarrow d_j \rightarrow d_k \Leftrightarrow d_i \rightarrow d_j \wedge d_j \rightarrow d_k \Rightarrow d_i \rightarrow d_k$。

2.3 数据存储成本模型

在商业云计算环境下，云服务提供商提供资源，并按自身的价格模型向用户收取相关成本。通常，云计算环境下基本类型的资源主要有两种：存储和计算[1)]。流行的云服务提供商的成本模型是基于这些类型的资源[2)]。例如，亚马逊云服务的原先价格为[2)]：存储资源按每月每 GB 数据收取 0.15 美元，计算资源按单个 CPU 每小时收取 0.1 美元。

云计算环境下，考虑到云服务价格的变化情况，将数据存储成本模型定义为：

总成本＝计算成本＋存储成本＋状态调整成本

其中，计算成本是指用于重新生成已删数据集所耗费的计算资源成本；存储成本是指用于存储数据所耗费的存储资源成本；状态调整成本是指从原先的数据存储状态调整到新的数据存储状态所耗费的一次性计算成本。总成本是这 3 种成本之和。由于状态调整成本是一次性短期的计算成本，不随使

1) 带宽是云计算环境下另一种常见的资源，用于数据的传入和传出。本文假定科学家把所有原始数据都上传到云计算系统进行科学实验。在单个云服务提供商的设施内，传输数据通常是免费的，因此不考虑管理应用程序数据集所涉及的数据传输成本

2) 由于市场因素，价格可能会不时波动

用频率变化，不同于随时间及使用频率等变化的长期的计算成本，因此将其与计算成本单独列出表示。为后文叙述方便，将计算成本与存储成本均称为常规成本。

为应用数据存储成本模型，在文献[12]的 DDG 数据属性的基础上增加了一个属性 r_i，用于表示数据的状态调整成本。故数据 d_i 的属性表示为 $\langle x_i, y_i, f_i, v_i, provSet_i, r_i, CostR_i\rangle$，其中各属性定义如下：

1）x_i 表示数据 d_i 从其最近的前驱数据生成所耗费的计算成本。

2）y_i 表示数据 d_i 存储在系统里，在单位时间内所耗费的存储成本。

3）f_i 是标记位，表示数据 d_i 在系统中是存储状态还是删除状态。

4）v_i 是使用频率，表示数据 d_i 多久使用一次。

5）$provSet_i$ 表示离数据 d_i 最近的且处于存储状态的前驱数据集，此时生成数据 d_i 的计算成本为：

$$genCost(d_i)=x_i+\sum_{\{d_k\mid d_j\in provSet_i\wedge d_j\to d_k\to d_i\}}x_k$$

6）r_i 表示数据 d_i 的状态调整成本，即数据 d_i 从原有的数据存储状态调整到新的数据存储状态所耗费的一次性计算成本。

假定在原有价格的数据存储状态下，DDG 中所存储的数据集表示为 $origSet$。如果数据 d_i 在原有数据集 $origSet$ 中，则不需要重新生成；否则，需要在 $origSet$ 中寻找离 d_i 最近的前驱数据集，并从该前驱数据集重新生成数据 d_i。故数据 d_i 的状态调整成本为：

$$r_i=\begin{cases}0, & d_i\in origSet\\ x_i+\sum_{\{d_k\mid provSet_i=origSet\wedge d_j\in provSet_i\wedge d_j\to d_k\to d_i\}}x_k, & d_i\notin origSet\end{cases}$$

云服务价格将在一段时间内保持不变，调整后的数据集存储状态将在这段时间内保持不变，这段时间也意味着距离云服务下次调价的时间段。将该时间段定义为 $t(t\neq 0)$，则数据 d_i 的状态调整成本率为 r_i/t。

7）$CostR_i$ 表示数据 d_i 在云计算环境下单位时间内所耗费的总成本，即数据 d_i 的总成本率，该值的大小取决于数据的状态。在传统的基准策略中，不考虑云服务价格变化，传统数据 d_i 的总成本率 $CostR_{i(old)}$ 为：

$$CostR_{i(old)}=\begin{cases}y_i, & f_i=\text{stored}\\ genCost(d_i)*v_i, & f_i=\text{deleted}\end{cases}$$

结合云服务价格变化，需在传统的数据 d_i 的总成本率上增加数据 d_i 单位时间内状态调整的成本。新采用的数据 d_i 的总成本率为：

$$CostR_i=CostR_{i(old)}+r_i/t$$

存储一个 DDG 所耗费的总成本率 $CostR$，是 DDG 中所有数据的总成本率之和，即 $\sum_{d_i\in DDG}CostR_i$。给定一段时间 t，存储一个 DDG 所耗费的总成本，是指所有数据的总成本率之和在该时间段的积分，即

$$Total\_Cost=\int_t\left(\sum_{d_i\in DDG}CostR_i\right)\cdot \mathrm{d}t$$

进一步将 DDG 的存储策略定义为 S，其中 S 属于 DDG 中的数据集，表示为 $S\subseteq DDG$，这说明云计算环境下存储数据集 S 中的数据，其余的则删除。将该存储策略 S 下存储 DDG 的总成本率定义为 SCR(Sum of Cost Rate)，则该总成本率为：

$$SCR=\left(\sum_{d_i\in DDG}CostR_i\right)_S$$

基于上述定义，对于云计算环境下的科学应用数据，不同的存储策略将导致不同的总成本率。新提出的总成本率反映了云服务价格变化。所提的存储策略旨在降低总成本率。

3　演化策略

3.1　演化 CTT-SP 算法在线性 DDG 的应用

线性 DDG 是指 DDG 中没有分支，除第一个和最后一个数据，其余每个数据只有一个直接前驱数据和一个直接后驱数据。文献[12]提出的 CTT-SP(Cost Transitive Tournament Shortest Path)算法，可以在固定每个数据使用频率的线性 DDG 下找到最小成本的存储策略。CTT-SP 算法的核心是基于 DDG 在第一个数据的前端增加虚拟数据 d_s，在最后一个数据的后端增加虚拟数据 d_e，构建成本传递竞赛图(Cost Transitive Tournament，CTT)。在该 CTT 中，从起始节点到结尾节点所经过的路径是一一对应于该 DDG 的存储策略；该路径的长度等于总成本率。采用 Dijkstra 最短路径算法，可得到该 CTT 的最短路径 $P_{min}\langle d_s,d_e\rangle$，即最小成本存储策略。

假定通过传统 CTT-SP 算法，在原有云服务价格下已得到原有数据存储策略的数据集 $origSet$；在新的云服务价格下已得到线性 DDG 下传统的 CTT。在该传统 CTT 中，对于每条边 $e\langle d_i,d_j\rangle$，其权重值定义为 $\omega\langle d_i,d_j\rangle$，表示 d_j 以及 d_i 与 d_j 之间的总成本率之和，其中 d_i 和 d_j 呈存储状态，而它们之间的数据呈删除状态。传统 CTT 的边权重值 $\omega_{(old)}\langle d_i,d_j\rangle$ 为：

$$\begin{aligned}&\omega_{(old)}\langle d_i,d_j\rangle\\&=CostR_{j(old)}+\sum_{\{d_k\mid d_k\in DDG\wedge d_i\to d_k\to d_j\}}CostR_k\\&=y_j+\sum_{\{d_k\mid d_k\in DDG\wedge d_i\to d_k\to d_j\}}(genCost(d_k)*v_k)\end{aligned}$$

结合云服务价格的变化情况，针对线性 DDG，在所提的演化的 CTT-SP 算法中，新 CTT 的边权重值为：

$$\omega\langle d_i,d_j\rangle=\omega_{(old)}\langle d_i,d_j\rangle+r_j/t$$

图 3 给出了一个带有 3 个数据的 DDG 通过演化的 CTT-SP 算法构建新 CTT 的简单例子。

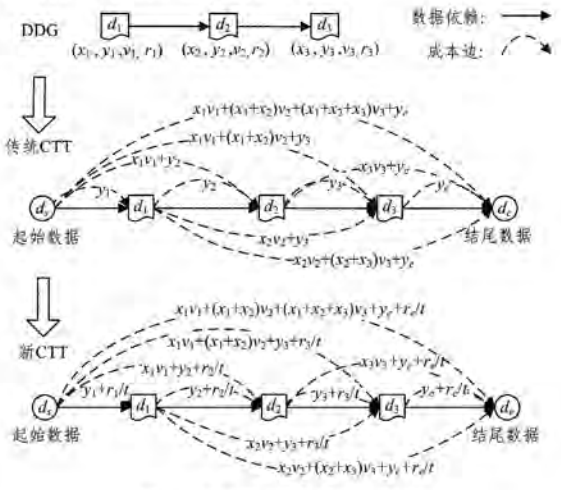

图 3　构建新 CTT 的简单例子

其中，对于两个虚拟数据 d_s（该起始数据只有出度边）和 d_e（该结尾数据只有入度边），其相关数据属性均设置为 0，即 $x_s=y_s=v_s=r_s=0$，且 $x_e=y_e=v_e=r_e=0$。

定理 1 给定数据集 $\{d_1,d_2,\cdots,d_n\}$ 的线性 DDG，通过传统 CTT-SP 算法得到 CTT 中每条边的权重值 $\omega\langle d_i,d_j\rangle$，并在该权重值上增加 d_j 的状态调整成本率 (r_j/t)，以构建新的 CTT。对于新的 CTT，求得最短路径 $P_{\min}\langle d_s,d_e\rangle$ 的长度，即为系统存储该 DDG 数据的最小成本率；最短路径所经过的数据即为相应的存储策略。

证明：1)CTT 原权重值所增加的 d_j 的状态调整成本率为 (r_j/t)，其取值是最小值。

①对于 d_j 在数据集 *origSet* 的情况下，d_j 的状态调整成本 r_j 的取值为 0，显然已为最小值。

②对于 d_j 不在数据集 *origSet* 的情况下，d_j 的状态调整成本 r_j 是取 *origSet* 中离 d_j 最近的前驱数据 d_p 重新生成所耗费的计算成本，该状态调整成本是最小值。

反证法：假设在数据集 *origSet* 中，存在 d_j 的一个不同于 d_p 的前驱数据 $d_{p'}$（$d_{p'}\neq d_p$，且 $d_{p'}\rightarrow d_p\rightarrow d_j$）。$d_j$ 的状态调整成本选用从前驱数据 $d_{p'}$ 生成，并且此时该状态调整成本 $r_{j'}$ 值是最小值，则有：

$$\begin{aligned}r_{j'}&=x_j+\sum_{\{d_k\mid d_{p'}\rightarrow d_k\rightarrow d_j\}}x_k\\&=x_j+\sum_{\{d_{k'}\mid d_{p'}\rightarrow d_{k'}\rightarrow d_p\}}x_{k'}+\sum_{\{d_k\mid d_p\rightarrow d_k\rightarrow d_j\}}x_k\\&>x_j+\sum_{\{d_k\mid d_p\rightarrow d_k\rightarrow d_j\}}x_k=r_j\end{aligned}$$

这与“$r_{j'}$ 是最小值”矛盾。因此，选用 origSet 中离 d_j 最近的前驱数据 d_p 重新生成 d_j，可使 d_j 的状态调整成本 r_j 是最小值。对于给定时间 t 值，该状态调整成本率 (r_j/t) 是最小值。

2)对于新的 CTT，求得最短路径 $P_{\min}\langle d_s,d_e\rangle$ 的长度，即为系统存储该 DDG 数据的最小成本率；最短路径所经过的数据即为相应的存储策略。文献[12]已证，在此不再赘述。

因此，定理 1 成立。

另外，普通 DDG 可切分成线性 DDG，因此针对线性 DDG 提出的演化策略也可应用在普通 DDG 上，以达到局部最优的存储策略[13]。

3.2 演化 CTT-SP 算法的复杂度分析

对于传统 CTT-SP 算法中 CTT 的每条边 $e\langle d_i,d_j\rangle$，演化 CTT-SP 算法首先得到每个数据 d_j 的状态调整成本，再将该状态调整成本率增加到原有边权重值里，最后用 Dijkstra 算法找到最短路径，并返回最小总成本的存储策略。演化 CTT-SP 算法的具体过程如算法 1 所示。

算法 1 演化 CTT-SP 算法

输入：起始数据 d_s，结尾数据 d_e；线性 DDG；传统 CTT；数据集 origSet（原有价格下的数据存储策略）；时间段 t（即离云服务下次调价的时间段）

输出：DDG 中的一组数据集 S(即最小成本存储策略)

```
1. for 传统 CTT 中的每条边 e<di,dj> do
2.    if dj 在数据集 origSet 中 do
3.       rj=0;
4.    end
5.    else do
6.       在 origSet 中，找到离 dj 最近的前驱数据 dp
7.       rj=xj;
8.       for DDG 中满足 dp→dq→dj 的数据 dq do
9.          rj=rj+xq;
10.      end
11.   end
12.   ω<di,dj>=ω<di,dj>+rj/t;
13. end
14. Pmin<ds,de>=Dijkstra_Algorithm(ds,de,CTT);
15. S=Pmin<ds,de>遍历的数据集;
16. return S;
```

由算法 1 可知，对于带有 n 个数据的线性 DDG，CTT 的构建是 n^2 条边的数量级；对于最长边，计算其权重值的时间复杂度 $O(n)$；此时总的时间复杂度为 $O(n^3)$。而 Dijkstra 算法的时间复杂度是 $O(n^2)$。因此，所提的演化 CTT-SP 算法在最坏情况下的时间复杂度为 $O(n^3)$。

4 实验结果与分析

4.1 实验环境与比较策略

亚马逊是知名的且被广泛认可的云服务提供商，因此选用亚马逊云的按需服务来进行模拟实验。采用 Python 语言编程，实现了所提的演化 CTT-SP 算法，并基于亚马逊 Linux 镜像的虚拟化 EC2 实例来运行该算法，以分析其成本效益。其中选用 EC2 标准的小实例(ml. small)来进行实验，因为它是 EC2 CPU 实例中的基本类型，在单个 ECU 中有稳定的性能。

为了证明在云服务价格变化后，所提的演化策略成本效益更高，将传统 CTT-SP 算法的策略[12]与相之比较。

4.2 数据集

真实的科学应用所产生的科学数据在数据大小、生成时间(从直接前驱数据生成所耗的计算时间)、使用频率与 DDG 结构上有很大不同。因此，在实验中采用随机参数来评估所提算法的普遍有效性。数据集采用随机生成的 DDG 数据，该 DDG 数据具有随机的数据大小、生成时间和使用频率。在本实验中，随机生成了大量带有不同数据个数的 DDG，根据文献[15]对有限元模型(FEM)等科学数据个案研究的数据参数，每个数据的大小是从 1GB 到 100GB 随机选取；生成时间是从 10 小时到 100 小时随机选取；使用频率是从每月一次到每年一次随机选取。云服务价格采用亚马逊云的成本模型，原先的价格是存储资源按每月每 GB 数据收取 0.15 美元，计算资源按单个 CPU 每小时收取 0.1 美元；新的价格是存储资源按每月每 GB 数据收取 0.0330 美元，计算资源按单个 CPU 每小时收取 0.058 美元。

4.3 成本评估

依据不同的数据存储策略，以及不同的参数时间 t 值(即离云服务下次调价的时间段)，对带有不同数据个数的随机 DDG 模拟存储科学数据的过程，并记录各存储策略的成本结

果，进行多次成本比较。实验时该时间 t 从 0.25 年到 10 年，每隔 0.25 年递增。

(1)针对不同数据个数的 DDG，固定时间 t 值为 0.25 年，各存储策略的成本率如图 4 所示。

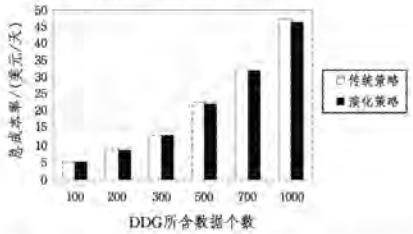

图 4　各存储策略的成本率比较(t=0.25 年)

(2)不同时间 t 值下，固定 DDG 的数据个数为 500(简称 DDG500)，各存储策略的成本如图 5 所示。

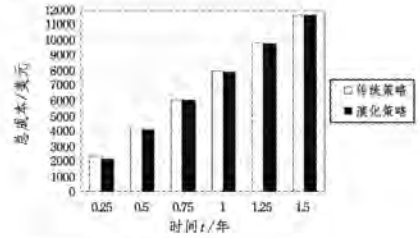

图 5　各存储策略的成本比较(DDG500)

由图 4 和图 5 可知，不管 DDG 和时间 t 值如何变化，演化策略仍不大于传统策略的成本(率)值。变换所固定的时间 t 值或 DDG 数据个数，进行多次实验，仍得到上述类似结果。由此可知，所提的演化策略是有效的，比传统策略更节省成本。

为了更明显地看出演化策略的有效性，以 DDG500 为例来进一步比较两存储策略之间各类成本的差值，并分析时间 t 的大小对各策略的影响。

(1)两个存储策略之间总成本的差值如图 6 所示。

据图 6 可知，随着时间 t 值的增大，相比于传统策略，演化策略所节省的总成本越来越少，甚至慢慢接近于 0，这意味着演化策略的总成本越来越接近传统策略，其效果越来越不明显。而参数时间 t 取值越小，演化策略的效果就越明显。另外，仅针对 DDG500，相比于传统策略，演化策略可节省 11～210 美元，数据量越大节省越多。

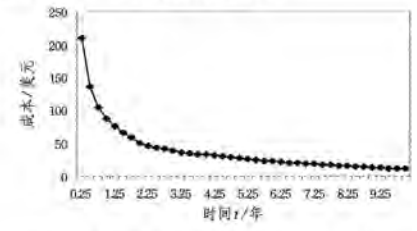

图 6　演化策略节省的总成本(DDG500)

(2)两个策略之间常规成本(即计算成本与存储成本之和)的差值和状态调整成本的差值如图 7 所示。

由图 7 可知，不管时间 t 如何变化，演化策略的常规成本总是不小于传统策略，而传统策略的状态调整成本总是不小于演化策略，这是因为传统策略得到的是最小常规成本的数据存储策略，演化策略得到的是最小总成本(常规成本与状态调整成本之和)的存储策略。

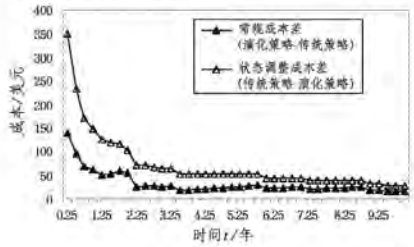

图 7　常规成本差与状态调整成本差(DDG500)

为进一步分析该结论的合理性，图 8 从公式推理的角度，分别分析并比较了两策略之间的常规成本 *Regular_Cost* 的大小和状态调整成本 Δ 的大小。由此可知，相比于传统策略，所提的演化策略是一种相对折中的策略，能以较小的状态调整成本调整到新的存储策略，但是所耗费的常规成本较大。而传统策略所耗的常规成本最小，但是却未考虑调整到新的存储状态所耗费的状态调整成本大小。另外，随时间 t 值的增大，传统策略所节省的常规成本逐渐减小，演化策略所节省的状态调整成本逐渐减小，这意味随着时间 t 的增大，演化策略所耗的常规成本和状态调整成本均越来越接近传统策略，效果逐渐不明显。

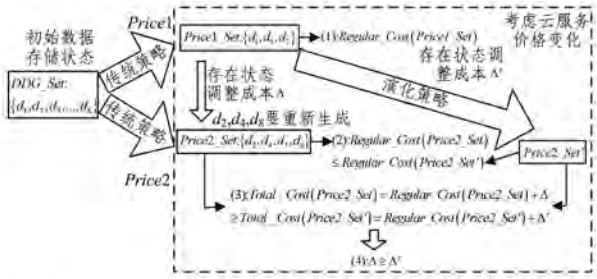

图 8　分析比较常规成本大小和状态调整成本大小

结束语　本文结合云服务价格变化，研究了在云计算环境下存储大量生成的科学数据的成本问题。为了使存储科学数据所耗的成本更小，提出了演化 CTT-SP 算法，在固定云服务下次调价的时间段 t 值时，对于 DDG 的原有数据存储状态，可得到新的成本最小的存储策略；并且 t 值越小，效果越明显。大量的模拟实验表明，对于在云计算环境下降低存储科学数据的成本，所提的演化策略非常有效。

当前采用云服务提供商按需服务的定价模式，并且假定所有大规模的应用数据存储在一个单一的云服务提供商下，不存在数据传输成本。然而，云服务提供商的定价模式越来越灵活丰富，需考虑复杂的定价模式；并且一些应用数据可能分布在固定位置，需要以一种分布式的方式运行，需考虑多个云服务提供商的情况。因此，未来工作将结合数据传输成本，以及更复杂的定价模式、成本模型展开。此外，将进一步研究预测数据集的使用频率、云服务下次调价时间，使得所提的存储策略更容易适应不同类型的应用。

参 考 文 献

[1] SINGH S, CHANA I. Cloud resource provisioning: survey, status and future research directions[J]. Knowledge & Information Systems, 2016, 49(3): 1-65.

[2] Amazon Cloud Services [EB/OL]. http://aws.amazon.com.

[3] LI Q, ZHENG X. Research Survey of Cloud Computing[J]. Computer Science, 2011, 38(4): 32-37. (in Chinese)
李乔, 郑啸. 云计算研究现状综述[J]. 计算机科学, 2011, 38(4): 32-37.

[4] KONDO D, JAVADI B, MALECOT P, et al. Cost-benefit analysis of Cloud Computing versus desktop grids[C]// Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2009). Washington DC, 2009: 1-12.

[5] CHEN C L P, ZHANG C Y. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data[J]. Information Sciences, 2014, 275(11): 314-347.

[6] YANG X, WALLOM D, WADDINGTON S, et al. Cloud computing in e-Science: research challenges and opportunities[J]. Journal of Supercomputing, 2014, 70(1): 408-464.

[7] GUNDA P K, RAVINDRANATH L, THEKKATH C A, et al. Nectar: automatic management of data and computation in datacenters[C]// Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI 2010). Berkeley, 2010: 1-8.

[8] DEELMAN E, SINGH G, LIVNY M, et al. The cost of doing science on the cloud: The Montage example[C]// International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2008). Austin: IEEE, 2008: 1-12.

[9] ADAMS I F, LONG D D E, MILLER E L, et al. Maximizing efficiency by trading storage for computation[C]// Proceedings of the 2009 Conference on Hot topics in Cloud Computing (HotCloud'2009). Berkeley, 2009: 1-5.

[10] YUAN D, YANG Y, LIU X, et al. A cost-effective strategy for intermediate data storage in scientific cloud workflow systems [C]// 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS). Atlanta, 2010: 1-12.

[11] YUAN D, YANG Y, LIU X, et al. A data dependency based strategy for intermediate data storage in scientific cloud workflow systems[J]. Concurrency and Computation: Practice & Experience, 2012, 24(9): 956-976.

[12] YUAN D, YANG Y, LIU X, et al. On-demand minimum cost benchmarking for intermediate dataset storage in scientific cloud workflow systems[J]. Journal of Parallel and Distributed Computing, 2011, 71(2): 316-332.

[13] YUAN D, YANG Y, LIU X, et al. A Local-Optimisation Based Strategy for Cost-Effective Datasets Storage of Scientific Applications in the Cloud[C]// Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing (Cloud 2011). Washington DC, 2011: 179-186.

[14] YUAN D, YANG Y, LIU X, et al. A Highly Practical Approach toward Achieving Minimum Data Sets Storage Cost in the Cloud [J]. IEEE Transactions on Parallel and Distributed Systems, 2013, 24(6): 1234-1244.

[15] YUAN D, CUI L, LI W, et al. An Algorithm for Finding the Minimum Cost of Storing and Regenerating Datasets in Multiple Clouds[J]. IEEE Transactions on Cloud Computing, 2015(99): 1.

[16] YUAN D, LIU X, YANG Y. Dynamic On-the-Fly Minimum Cost Benchmarking for Storing Generated Scientific Datasets in the Cloud[J]. IEEE Transactions on Computers, 2015, 64(10): 2781-2795.

[17] YUAN D, YANG Y, LIU X, et al. Computation and Storage Trade-Off for Cost-Effectively Storing Scientific Datasets in the Cloud[M]// Handbook of Data Intensive Computing. Springer New York, 2011: 129-153.

(上接第 134 页)

[15] TURHAN B, BENER A. Analysis of naïve bayes' assumptions on software fault data: An empirical study[J]. Data Knowledge Engineering, 2009, 68(2): 278-290.

[16] WANG J, SHEN B J, CHEN Y T. Compressed C4.5 Models for Software Defect Prediction[C]// International Conference on Quality Software. 2012: 13-16.

[17] SUN Z B, SONG Q B, ZHU X Y. Using Coding Based Ensemble Learning to Improve Software Defect Prediction[J]. IEEE Transactions on Systems Man and Cybernetics Part C, 2012, 42(6): 1806-1817.

[18] MA G, LUO C, CHEN H. Kernel Based Asymmetric Learning for Software Defect Prediction[J]. IEICE Transactions on Information and Systems, 2012, E95-D(1): 215-226.

[19] ZHENG J. Cost-sensitive boosting neural networks for software defect prediction [J]. Expert Systems with Applications, 2010, 37(6): 4537-4543.

[20] ROSASCO L, VERRI A, SANTORO M, et al. Iterative projection methods for structured sparsity regularization[OL]. http://dspace.mit.edu/bitstream/handle/1721.1/49428/MIT-CSAIL-TR-2009-050.pdf?sequence=1.

[21] YANG M, ZHANG L, YANG J, et al. Metaface learning for sparse representation based face recognition[C]// International Conference on Image Processing. 2010: 1601-1604.

[22] ZHANG D, YANG M, FENG X. Sparse representation or collabortive representation: which helps face recognition? [C]// IEEE International Conference on Computer Vision. 2011: 471-478.