ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2011/05(00)-0000-00
DOI: 10.3778/j.issn.1673-9418.2011.00.000

E-mail: fcst@vip.163.com
http://www.ceaj.org
Tel: +86-10-51616056

支持成本效益分析的 Web 服务容错策略规划方法*

朱建锋, 彭 鑫+, 赵文耘
复旦大学软件学院, 上海 201203

A Cost-Effective Planning Method for Fault Tolerant Web Services *

ZHU Jianfeng , PENG Xin +, ZHAO Wenyun
Software School, Fudan University, Shanghai 201203, China
+ Corresponding author: Phn: 021-51355343, Fax: 021-51355358,E-mail: pengxin@fudan.edu.cn

ZHU Jianfeng, PENG Xin, ZHAO Wenyun. A Cost-Effective Planning Method for Fault Tolerant Web Services. Journal of Frontiers of Computer Science and Technology, 2011, 5(0): 1-000.

Abstract: Replication strategy-based fault tolerance approaches for Web services have been widely used to guarantee the reliability and performance of service-oriented critical systems. Current approaches pay the major attention on performance-related qualities such as response time and failure rate; however, they ignore cost-related qualities such as the cost of service invocation and compensation, i.e., lack the consideration on cost-benefit analysis. In this paper, we propose a strategy-planning approach for Web service fault tolerance supporting cost-benefit analysis. The approach first defines the cost prediction formula for the 9 common replication strategies based on the cost of service invocation and compensation, and then designs a strategy selection algorithm supporting cost-benefit analysis. The experimental study has demonstrated that our approach can significantly improve the effectiveness of the fault tolerance strategy with acceptable overhead.
Key words: Web Service; Fault Tolerance; Replication Strategy; Cost; Service Performance

摘 要: 基于各种冗余策略的 Web 服务容错方法被广泛应用于面向服务的关键性系统的可靠性和性能保障。已有的 Web 服务容错策略选择方法只考虑服务响应时间和失效率等与服务性能相关的因素，而忽略了服务调用和补偿费用等成本问题，导致所选择的策略可能在成本效益方面存在不足。针对这一问题，文章提出

*The National Natural Science Foundation of China under Grant No. 90818009 (自然科学基金).
Received 2000-00, Accepted 2000-00.

了一种支持成本效益分析的 Web 服务容错策略规划方法。该方法针对九种常用的 Web 服务冗余策略给出了综合考虑调用和补偿费用的成本预测公式，并在此基础上给出了支持成本效益分析的 Web 服务容错策略选取算法。针对该方法的验证实验表明，文章所提出的方法能够显著提高 Web 服务容错策略的有效性，并且所产生的额外时间开销在可接受范围内。

关键词: Web 服务；容错；冗余策略；成本；服务性能

文献标识码：A 中图分类号：TP311.5

1 引言

Web 服务是面向服务架构的一种实现技术，通过标准的 Web 协议提供服务，目的是保证不同平台的应用服务可以互操作。Web 服务通过标准的接口和交互协议实现了运行在各种异构计算机软件硬件系统之上的服务能够实现方便的集成和互操作。由于这些特点，Web 服务在面向服务的体系结构（Service-Oriented Architecture, SOA）中得到了广泛的应用。Web 服务已经成为在互联网环境下构建分布式系统的基石，随着越来越多的应用程序建立在 Web 服务的基础上，提供可靠的 Web 服务已经成为了一个重要的问题[1,2,3]。

随着 Web 服务和面向服务的体系结构在电子商务、公共服务等领域的广泛应用，越来越多的关键性（critical）系统都通过服务化的方式实现系统构建和集成。这些系统对于可靠性和性能有着很高的要求，由自身缺陷或外部环境导致的服务失效或延迟问题可能会带来严重的经济损失、社会影响甚至威胁人的生命安全。然而，另一方面，由于大量的第三方 Web 服务由外部提供，且运行在复杂、多变的网络环境之中，这些都使得服务的可靠性和性能无法得到保障。

容错是保证系统可靠性和可用性的关键机制[4]，能使系统在有错误或者失效发生时依然能够正确运行。Web 服务容错主要使用模块冗余来完成。模块冗余有模块多版本和模块复制两种类型[5]，其中模块复制使用的较为广泛。主动复制[6]和被动复制[7]是比较经典的两个复制策略，主动复制策略中，所有冗余副本同时响应客户请求，能提供较好的性能，但是需要耗费较多的资源；被动复制策略中，主副本首先响应客户请求，当主副本失效时备份副本继续响应客户请求。

已有的 Web 服务容错策略的基本思想一般都是使用各种组合方式（如并行、串行等）的冗余服务来保障系统性能并降低失效率。然而，许多由第三方提供的 Web 服务都是收费服务或者存在补偿费用（如订票服务的退票费），因此在现实的面向服务应用中容错策略规划还需要考虑由此带来的成本问题。例如，并行冗余策略中由于多余服务的调用或补偿费用而带来的额外开销很多时候是一个较大的问题。

针对以上问题，本文提出了一种支持成本效益分析的 Web 服务容错策略规划方法，以期在合理的冗余成本下实现较高的容错性能。该方法针对常用的 Web 服务冗余策略给出了容错成本计算公式，并在此基础上给出了一个容错策略规划算法。针对该方法的实验验证结果表明，本文所提出的方法能在可控的成本下选择出性能较好的冗余策略，并且算法自身效率在较大规模的服务集合下依然较高。

本文余下部分的组织结构如下。第 2 部分介绍研究背景及相关工作。第 3 部分针对九种常用的 Web 服务冗余策略分别给出了相应的容错成本计算公式，第 4 部分介绍所提出的冗余策略规划算法。第 5 部分基于本文所提出的策略规划方法进行了对比实验和分析。最后，第 6 部分总结全文。

2 背景及相关工作

2.1 背景

按照状态来区分，Web 服务可以分为有状态和无状态两种类型。有状态服务是指服务被调用后，该服务的状态发生了改变或者该服务使现实世界的状态发生了改变。例如，订票服务是有状态服务的一种，调用一次订票服务后，现实世界就产生了一个新的订单并随之生成了一个债务关系；支付服务也是有状态服务的一种，调用支付服务后，现实世界中就产生了一次资金转移。对于有状态服务，

如果成功调用后选择取消服务，那么调用者可能需要为此付出一定的补偿费用。例如，调用了订票服务后如果选择取消，那么退票费就是需要付出的补偿费用。

无状态服务是指被调用后不会引起本身或者现实世界状态变化的服务。典型的无状态服务是信息查询类服务，比如天气查询服务和亚马逊商品信息查询服务等。这类服务调用后，无论是否使用调用的结果，都不需要支付补偿费用。

按照是否收费来区分，可以将 Web 服务分为收费服务和免费服务两种类型。需要支付费用才能调用的服务称为收费服务。收费服务的计费模式多种多样，主要的模式有按次计费、按时间段计费和一次性买断等。按次计费是指以调用次数为单位对调用者收费，即调用者每成功调用一次都需要付给服务提供者一定的费用。如果调用失败，那么调用者可以不为本次调用付费。按时间段计费是指调用者付出一定的费用买断一个时间段，在这个时间段内无论调用多少次都不需要付出额外的费用。常见的按时间段收费有包月收费和包年收费等。而一次性买断则是指调用者付出一定的费用之后，即可终身使用该服务。本文为了简化方法和实验设计，假设所有的服务都采用按次计费的方式。

2.2 相关工作

为了解决在使用 Web 服务时的容错问题 N. Salatge 和 J.C. Fabre[8]使用主动和被动两种冗余策略来提高调用 Web 服务的性能。主动冗余是指同时调用多个功能相同的备份服务，这样就可以降低 Web 服务失效率并提高性能。主动冗余策略在 FTWeb[9]、Thema[10]以及 WS-Replication[11]等工作中都有较好的应用。被动冗余是指当主服务调用失败后，转而调用该服务的备份服务，以期降低失效率。应用了被动冗余策略的工作有 FT-SOAP[12]和 FT-CORBA[13]。

郑子彬等人[14]在他们的研究中将冗余策略细化为九种，分别是：主动冗余、时序冗余、被动冗余、主动时序冗余、时序主动冗余、主动被动冗余、被动主动冗余、时序被动冗余和被动时序冗余。其中前三种是基本冗余策略，后六种是前三种的组合策略。在此基础上，他们还将这九种策略分成并行、串行和混合三种类型。上述第 1 种策略是并行策略，第 2、3、8、9 种策略是串行策略，第 4、5、6、7 种策略是混合策略。他们的工作关注于通过预测和分析为每个服务选择最优的服务冗余策略。主要过程是先使用用户的响应时间阈值选择出合适的服务并行数，然后在串行、并行和混合三种策略类型中选择出对应类型，再根据用户的性能要求最终确定选择何种策略。在他们的研究中，选择合适策略的依据的是冗余策略对于失效率和响应时间的改进效果，忽略了调用服务可能需要的成本。

S. Stein 等人[15]曾提出一种与成本效益相关的 Web 服务流程规划方法。在他们的方法中，Web 服务被统一处理，不按状态进行区分，从而忽略了有状态服务被调用之后可能产生的补偿问题。另外，由于该方法所针对模型的特点，在计算 Web 服务成本时，作者仅仅考虑了采用主动被动冗余策略的情况。

3 冗余策略及成本计算

不同冗余策略所需要的容错成本各不相同。在计算某一策略的容错成本时，需要预先取得该策略使用的备份服务集合。在本节以下的所有公式中，每一策略都使用了一个有序服务集合$\{\mathrm{ws}\}_{\mathrm{i}=1}^{\mathrm{n}}$，该集合包含了 n 份备份服务，服务按照某一属性从高到低有序排列。用户排序的属性可以是服务的性能，也可以是服务的性价比。下面分别说明使用不同冗余策略需要的成本：

1. 主动冗余。主动冗余是指集合中 n 份备份服务同时被调用。这种冗余策略一般能够获得最好的性能，也能够获得最小的失效率和最低的响应时间，但是却需要最多的成本。当服务集合中包含的服务是有状态服务时，需要支付的补偿费用将相当可观，因为所有调用成功的服务中，只有一个结果会被使用，其他未被使用的结果都需要支付补偿费用。主动冗余策略的成本可以使用公式（1）表示：

$$\mathrm{D}=\sum_{i=1}^{n} d_i\left(1-f_{\mathrm{i}}\right)+\frac{n-1}{n}\sum_{i=1}^{n}\mathrm{c}_i\left(1-f_{\mathrm{i}}\right) \quad (1)$$

公式（1）中，第 i 个服务ws_i每次调用的成本为d_i，f_i是服务ws_i的失效率，c_i为服务ws_i的补偿费用。

2. 时序冗余。时序冗余是指调用集合中某个特定服务，如果调用失败，将重新尝试调用该服务。使用这种冗余策略需要付出的成本较低，但是在服务失效率较高的情况下可能需要等待较长的时间，

总体性能较差。时序冗余策略需要支付的成本可以使用公式（2）表示：

$$D = d_{select} \quad (2)$$

公式（2）中，d_{select}是特定服务每次调用需要支付的成本。

3. 被动冗余。被动冗余是指在 m 份备份服务中，按顺序依次调用，当第 i 份备份服务调用失败，就调用第 i+1 份备份服务。这种策略和时序冗余一样，具有较低的成本和较差的性能。被动冗余策略需要支付的成本可以使用公式（3）表示：

$$\mathbf{D} = \sum_{i=1}^{m}\left(\prod_{k=0}^{i-1} f_k\right)(1 - f_i)d_i \quad (3)$$

$$\text{其中 } f_0 = 1$$

公式（3）中，第 i 个服务ws_i每次调用需要支付的成本为d_i，f_i是服务ws_i的失效率。

4. 主动时序冗余。主动时序冗余是指在 n 份备份服务中，前 v 份备份服务被同时调用，当这 v 份备份服务都失败之后，再重新同时调用这 v 份服务。这种策略将主动冗余和时序冗余两种策略结合在一起，在宏观上使用时序冗余策略，在微观上使用主动冗余策略。主动时序冗余策略需要支付的成本可以使用公式（4）表示：

$$\mathbf{D} = \sum_{i=1}^{v} d_i(1 - f_i) + \frac{v-1}{v}\sum_{i=1}^{v} c_i(1 - f_i) \quad (4)$$

公式（4）中，第 i 个服务ws_i每次调用需要支付的成本为d_i，f_i是服务ws_i的失效率，c_i为服务ws_i的补偿费用。以下公式（5）到公式（9）中所使用的符号与公式（4）中使用的符号意义相同，不再赘述。

5. 时序主动冗余。时序主动冗余是指在候选服务集合中，前 v 份服务同时被调用，当这 v 份中的任何一个服务调用失败后，都将立刻重新调用此服务，重试次数为 m。时序主动冗余策略与主动时序冗余策略刚好相反，在宏观上使用主动冗余策略，在微观上使用时序冗余策略。时序主动冗余策略需要支付的成本可以使用公式（5）表示：

$$\mathbf{D} = \sum_{i=1}^{v} d_i \sum_{k=1}^{m} f_i^{k-1}(1 - f_i) + \frac{v-1}{v}\sum_{i=1}^{v} c_i \sum_{k=1}^{m} f_i^{k-1}(1 - f_i) \quad (5)$$

6. 主动被动冗余。主动被动冗余指的是将备份服务集合按顺序分成 m 个子集，每个子集包含 v 份备份服务，先并行调用第一个子集中的所有备份服务，当所有服务都失败之后，再并行调用第二个子集中的所有备份服务，以此类推。该策略将主动冗余和被动冗余两个策略结合到一起，在宏观上使用被动冗余策略，在微观上使用主动冗余策略。使用主动被动冗余策略需要支付的成本可以使用公式（6）表示：

$$\mathbf{D} = \sum_{i=1}^{m}\left(\left(\prod_{k=0}^{i-1}\prod_{j=1}^{v} f_{kj}\right)\left(1 - \prod_{j=1}^{v} f_{ij}\right)\left(\sum_{j=1}^{v} d_{ij}(1 - f_{ij}) + \frac{v-1}{v}\sum_{j=1}^{v} c_{ij}(1 - f_{ij})\right)\right) \quad (6)$$

$$\text{其中 } f_{kj} = \begin{cases} 1, k = 0 \\ f_{(k-1)v+j}, k \geq 1 \end{cases}, d_{ij} = d_{(i-1)v+j}$$

7. 被动主动冗余。被动主动冗余是指备份服务集合中所有的备份服务按照顺序分成 m 个子集，每个子集中含有 v 份备份服务，首先并行调用第一个子集中的所有备份服务，当任何一个被调用的服务失败时，立刻调用该服务在第二个子集中对应的备份服务，以此类推。该策略与主动被动冗余策略刚好相反，在宏观上使用主动冗余策略，在微观上使用被动冗余策略。使用被动主动冗余策略所需要支付的成本可以使用公式（7）表示：

$$\mathbf{D} = \sum_{i=1}^{v}\left(\sum_{j=1}^{m}\left(\prod_{k=0}^{j-1} f_{ik}\right)(1 - f_{ij})d_{ij}\right)\left(1 - \prod_{j=1}^{m} f_{ij}\right) + \frac{v-1}{v}\sum_{i=1}^{v}\left(\sum_{j=1}^{m}\left(\prod_{k=0}^{j-1} f_{ik}\right)(1 - f_{ij})c_{ij}\right)\left(1 - \prod_{j=1}^{m} f_{ij}\right) \quad (7)$$

$$\text{其中 } f_{ik} = \begin{cases} 1, k = 0 \\ f_{(k-1)v+i}, k \geq 1 \end{cases}, d_{ij} = d_{(k-1)v+i}$$

8. 时序被动冗余。时序被动冗余是指在备份服务集合中取出前 u 份备份服务组成被调用的服务子集，首先调用子集中的第一份备份服务，如果调用失败，将继续重新调用该服务，重试 m 次仍失败之后，按照相同方式调用子集中第二份备份服务，以此类推。时序被动冗余策略将时序冗余策略和被动冗余策略结合到一起，在宏观上使用被动冗余策略，在微观上使用时序冗余策略。使用时序被动冗余策略所需要支付的成本可以使用公式（8）表示：

$$\mathbf{D} = \sum_{i=1}^{u}\left(\prod_{k=0}^{i-1} f_k^m\right)(1 - f_i^m)d_i \quad (8)$$

$$\text{其中 } f_0 = 1$$

9. 被动时序冗余。被动时序策略是指在备份服务集合中取出前 u 份备份服务组成被调用的服务子集，首先调用子集中的第一份备份服务，如果调用失败，那么继续调用子集中的第二份备份服务，如果子集中的 u 份备份服务全部调用之后仍然没有正确结果返回，那么就在子集的 u 份备份上再次执行一次上述调用过程，重试上述过程的次数为 m。该策略与时序被动策略是相反的过程，该策略在宏观上使用时序冗余策略，在微观上使用被动冗余策略。使用被动时序策略所需要支付的成本可以使用公式（9）表示：

$$\mathbf{D}=\sum_{i=1}^{u}\left(\prod_{k=0}^{i-1}\boldsymbol{f_k}\right)(\mathbf{1}-\boldsymbol{f_i})\boldsymbol{d_i} \quad (9)$$

其中 $f_0=1$

4 冗余策略选择算法

4.1 算法输入及输出

算法输入包括：

1）$\{ws\}_{i=1}^{n}$：所有备份服务组成一个按性价比从高到低排列的有序备份服务集合。按照性价比排列备份服务优于单纯按照性能或者单纯按照成本排列。因为现实世界中的任何商品总是价格高者质量相对更好，消费者选购商品时也是在同一价位上选择质量更优者，收费 Web 服务当然也不会违背基本经济规律，这种排列方式即考虑到了服务性能也考虑到了服务的成本。

2）r_i：备份服务集合中第 i 个服务的性能值。服务性能的计算有较多方法，一般都考虑了响应时间、可靠性等多种因素，本文不做探讨，只定义 r 为服务的性能，且值越大表示服务性能越好。

3）d_i：备份服务集合中第 i 个服务单次调用的成本。

4）f_i：备份服务集合中第 i 个服务的失效率。

5）$d_{threshold}$、$r_{threshold}$、$f_{threshold}$：分别表示用户在执行一次调用策略时能够接受的成本、性能和失效率的阈值。

6）偏好：性能优先或者成本优先。

除了以上输入之外，该算法还有两个常量系数，即性能阈值系数 a 和失效率阈值系数 b。这两个常量系数并不是算法的输入，但是需要根据实际情况进行调整。

算法输出即执行完该算法所选择出的冗余策略。

4.2 算法过程

首先确定一次并行调用的最小备份数 v，欲得到 v 的值，须求得一个最小的 x 值，使得如下不等式组成立：

$$\begin{cases} D_x \le d_{threshold} \\ r_x \ge r_{threshold} \\ f_x \le f_{threshold} \end{cases}$$

其中，D_x、r_x、f_x分别表示前 x 份备份服务并行调用时的成本、性能和失效率。

解上述不等式组得到 x 的解集 X。当X = ∅时，说明三个阈值设置不合理，需要重新设置；当X ≠ ∅时，可得$v = \mathrm{MIN}(x), x \in \mathrm{X}$。

以下，根据 v 取值不同，可以分为三种情况：

1、当 v=1 时，采取串行策略，即策略 2、3、8 和 9。

偏好为性能优先时，为了确定哪种串行策略更好，需要先取得一个高性能服务集合 G。

$$\mathrm{G}=\{ws_i | r_i > ar_{threshold} \&\& 1 \le i \le n\}$$

如果|G|=0，证明系数 a 需要重新调整。如果|G|=1，只有一个服务高性能服务要求，那么使用策略 2，即时序冗余策略。如果|G|=n，所有服务都符合高性能服务的要求，那么使用策略 3，即被动冗余策略。否则，就使用策略 8 或者策略 9。当 G 中的服务ws_1的性能$r_1 > avg(G)$，第一个服务的性能优于集合 G 的平均性能时，使用策略 8 更合适，即重试ws_1比调用其他备份服务更可能获得高性能。反之，使用策略 9。

偏好为成本优先时，使用公式（2）、（3）、（8）、（9）计算每一个策略的成本，选择成本最低的使用。

2、当 $1<v<n$ 时，采取混合策略，即策略 4、5、6 和 7。

偏好为性能优先时，p_j表示第 j 组由 v 份服务组成的服务集合并行调用时的性能，其中 $1<j<m$，其中 m 是组数。那么当$p_1 > avg(p_1, p_2 \dots p_m)$，第一组服务的性能大于平均服务性能时，使用策略 4、5 比使用策略 6、7 要好。即重试第一组服务能获得更好的性能。当第一组服务的平均失效率 $\mathrm{avg}(f_1, f_2 \dots f_v) > bf_{threshold}$时，使用策略 5、7 能获得更短的等待时间。在网络环境较差、服务失效率较高的情况下，使用策略 4、6 需要等待较长的时间才能进行下一次的重试。上述选择过程如表 1 所

示。

偏好为成本优先时，使用公式（4）、（5）、（6）、（7）分别计算出策略 4、5、6、7 的预期成本，然后选择成本最少的策略即可。

3、当 $v=n$ 时，采取并行策略，即策略 1。

表 1 混合策略选择

Table 1 Hybrid Strategies Selection

	$p_1 > avg(p_1, p_2 \dots p_m)$	$p_1 \le avg(p_1, p_2 \dots p_m)$
$avg(f_1, f_2 \dots f_v) > bf_{threshold}$	策略 5	策略 7
$avg(f_1, f_2 \dots f_v) \le bf_{threshold}$	策略 4	策略 6

以上过程使用伪码可表述为算法 1-1 到算法 1-3。

算法 1-1.

```
1. StrategySelection(S,d,r,f){
2.     n=|S|;
3.     v=MIN(S,d,r,f);
4.     if (v == 1) {
5.         return UseSequentialStrategy(S,d,r,f);
6.     } else if (1 < v && v < n) {
7.         return UseHybridStrategy(S,d,r,f);
8.     } else if (v == n) {
9.         return "Active Strategy";
10.    } else {
11.        Error("Input Error");
12.    }
13.}
```

算法 1-1 的输入 S 表示备份服务集合，d、r、f 分别表示成本、性能和失效率的阈值。其中，第 3 行通过 MIN 函数获得最小并行数 v，从第 4 行开始根据 v 的取值进行策略选择。当 v 是 1 时，进行串行策略选择过程，如算法 1-2 所示。当 v 在 1 到 n 之间时，进行混合策略选择过程，如算法 1-3 所示。当 v 等于 n 时，使用主动策略。

算法 1-2.

```
1. UseSequentialStrategy(S,d,r,f){
2.         if (prefer=="performance") {
3.             SH=GetHighPerformance(S);
4.             if (|SH| == 0) {
5.                 Adjust(a);
6.             } else if (|SH| == 1) {
7.                 return "Time Strategy";
8.             } else if (|SH| == n) {
9.                 return "Passive Strategy";
10.            } else {
11.                R1=GetFirstPerformance(SH);
12.                AVG=GetAvgPerformance(SH);
13.                if (R1 > AVG) {
14.                    return "Time+Passive";
15.                } else {
16.                    return "Passive+Time";
17.                }
18.            }
19.        } else {
20.            return UseLowestCost("Time",
21.                "Passive","Time+Passive",
22.                                "Passive+Time");
23.        }
24.}
```

算法 1-2 中第 5 行表示系数 a 不合理需要调整。第 11、12 行分别取得集合 SH 中第一份服务的性能 R1 和 SH 中所有服务的平均性能 AVG。第 20 行 UseLowestCost 函数计算传入策略的成本并返回成本最低的策略。

算法 1-3.

```
1. UseHybridStrategy(S,d,r,f){
2.         if (prefer=="performance") {
3.         S1,S2...Sm=Split(S,v);
4.         P1,P2...Pm=Performance(S1,S2...Sm);
5.         AVGP=AvgPerformance(S1,S2...Sm);
6.         AVGF=AvgFailure(S1);
7.         if (P1 > AVGP) {
8.             if (AVGF > b * f) {
9.                 return "Time+Active";
10.            } else {
11.                return "Active+Time"
12.            }
13.        } else {
14.            if (AVGF > b * f) {
15.                return "Passive+Active";
16.            } else {
17.                return "Active+Passive";
18.            }
19.        }
20.    } else {
21.        return UseLowestCost("Time",
22.          "Passive","Time+Passive",
23.                        "Passive+Time");
24.    }
25.}
```

算法 1-3 中第 3 行表示将 S 分成 m 个子集，表

示为 S1,S2...Sm，每个子集中含有 v 个备份服务，分别计算 m 个子集各自并行调用时的性能，表示为 P1,P2...Pm，计算 m 个子集的平均并行调用性能 AVGP，计算第一个子集中服务的平均失效率 AVGF。

5 实验

5.1 实验设计及过程

实验过程中所使用的备份服务集合中所有服务的功能都是相同的，均是接收一个问候字符串并返回一个问候字符串，每个服务有不同的响应时间和失效率。在实验中，我们使用响应时间的倒数乘以服务的成功率 $r=(1/t)*(1-f)$来表示服务的性能。表 2 展示了该服务集合中 5 个典型的服务。

实验中，服务集合中的所有服务均使用 Java 编写，运行于 tomcat6 中，需要 Axis1.1 和 JDK1.6 支持。运行服务的计算机硬件配置和操作系统为：Intel Core 2.33GHz 4 核 CPU、4G 内存、64 位 Windows Server 2008 操作系统。冗余策略选择算法以及实验客户端使用 Java 实现，运行的计算机配置为：Intel core 1.66GHz 双核 CPU、2G 内存以及 Windows7 操作系统。

表 2 实验中所使用 Web 服务参数

Table 2 Parameters of Web Services used in Experiments

成本（元）	响应时间（秒）	失效率	性能	性价比
0.5	1	10%	0.9	1.8
0.4	1	20%	0.8	2
0.3	1	30%	0.7	2.33
0.2	2	20%	0.4	2
0.1	3	40%	0.4	2

表 2 中的五个服务，基本上遵循了实际生活中质优价高的规律，性能较高的服务价格也较高。如

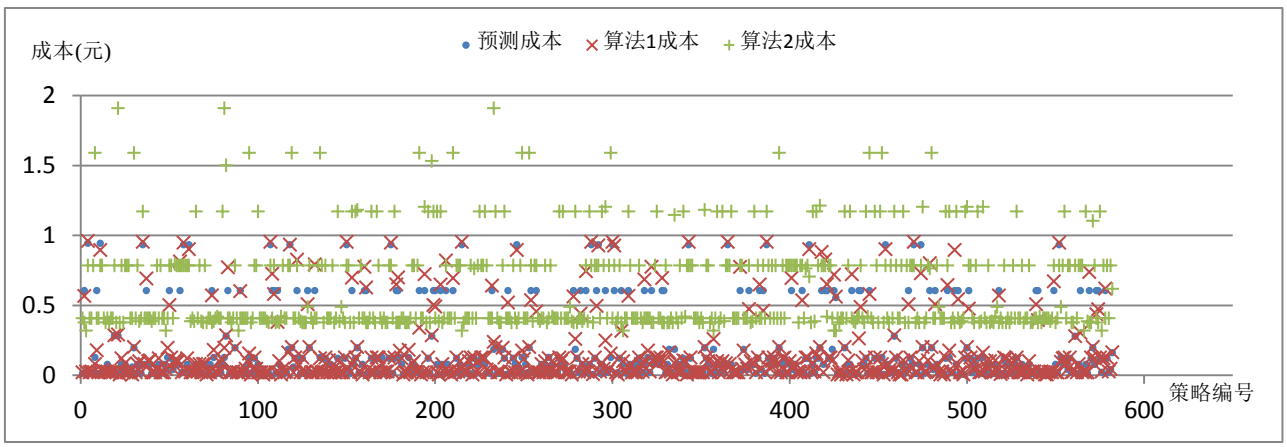


图 1 成本对比

Fig.1 Cost comparison

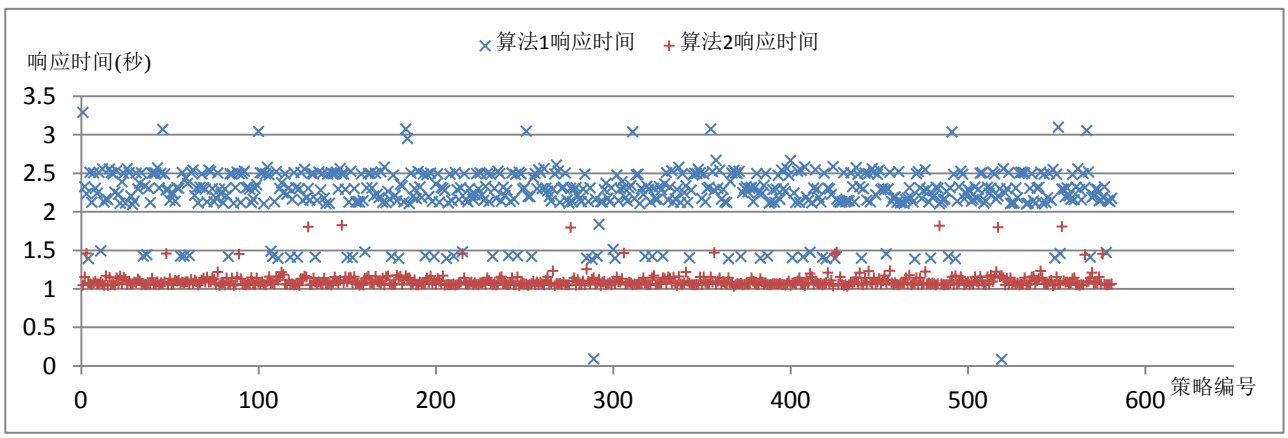


图 2 响应时间

Fig.2 Response time

果只考虑性能，那么第一个服务是最好的服务，但是如果考虑到成本因素，第三个服务虽然性能不是最好，但是性价比却最高。所以按照性价比对服务进行排序是一个比较经济又不失性能的方法。

实验分为两部分：在第一部分中使用 100 个服务构成备份服务集合，使用不同的阈值作为输入分别执行本文描述的选择算法和不考虑成本的选择算法进行策略选择并运行选择出的策略，记录下不同策略的实际性能以及实际支出成本等信息；在第二部分中，不断增加备份服务集合的规模，记录策略选择过程需要的时间。

5.2 实验结果

实验过程中分别使用本文提出的算法（称为算法 1）和不考虑成本的算法（称为算法 2）分别进行了 581 次策略选择和执行，即使用每一组阈值同时执行算法 1 和算法 2。实验中每种算法的所有策略选择成本如图 1 所示，其中横轴表示策略选择的编号，纵轴表示成本。算法 1 所选择的 581 个策略平均花费为 0.164 元，而算法 2 所选择的 581 个策略的平均花费为 0.618 元，可见算法 2 在成本上远远高于算法 1。

表 3 平均成本对比表

Table 3 Comparison of average cost

	成本（元）
成本预测平均值	0.161
算法 1 成本平均值	0.164
算法 2 成本平均值	0.618

由于在实验中没有出现策略失效的情况，即两种算法所选择的策略失效率都为 0，所以可以只使用平均响应时间来比较两者所选择策略的平均性能。每次选择策略的响应时间如图 2 所示。算法 1 所选择的 581 次策略平均响应时间为 2.23 秒，算法 2 所选择的 581 次策略的平均响应时间为 1.1 秒。虽然在响应时间上，算法 1 所选择的策略不占优势，但是综合成本考虑，算法 1 还是能够选择出性能不太差同时成本又比较低的策略。而算法 2 虽然能选出性能较好的策略，但是成本却是很高的。

算法 1 在执行时会使用公式（1）-（9）预测每种策略的成本，图 1 显示了实验中所有策略成本的预测值和实际值。将每次预测的结果和实际成本取平均值得到表 3，预测平均值为 0.161 元，实际平均值为 0.164 元，两者相差非常小，可见，上述九个公式对成本的计算是比较准确的。

另一方面，为了检测算法 1 本身在执行过程中的性能，我们不断增加备份服务集合的规模，并在该集合上执行算法 1，得到图 3 所示的结果。由图 3 可以看出，随着备份服务集合的规模从 1000 增加到 10000 的过程中，策略选择的时间也在增加，从 0.002 秒增加到 0.196 秒。但是与策略执行时间相比还比较小，与实际的 Web 服务调用时间相比也比较小，不会对应用的性能造成明显影响。

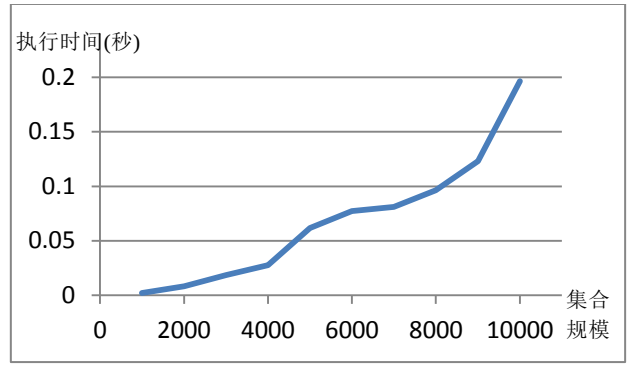


图 3 平均每次执行时间变化图

Fig.3 Average Executing Time

6 总结及展望

本文提出了一种支持成本效益分析的 Web 服务冗余容错策略规划方法。该方法以性价比作为备份服务集合排序的依据，并在容错策略选择过程中综合考虑服务价格和补偿费用等成本因素以及容错性能。该方法允许用户设置可以接受的成本上限，并在成本范围内进行冗余容错策略的优化选择。针对现有方法的对比实验表明，本文的方法能够在合理的性能开销基础上选取成本效益更高的服务冗余容错策略。

References:

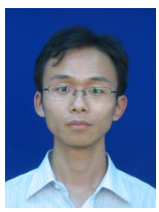
[1] K. Iwasa, C. Evans, D. Chappell, et al., WS-Reliability 1.1, OASIS Web Services Reliable Messaging Technical Committee,2004

[2] N.B. Lakhal, T. Kobayashi and H. Yokota. THROWS: An Architecture for Highly Available Distributed Execution of Web Services Compositions[C]//RIDE. 14th International Workshop on Research Issues in Data Engineering(RIDE-WS-ECEG 2004), Boston, March 28-29, 2004. USA: IEEE Computer Society, 103-110

[3] F. Tartanoglu, V. Issarny, A. Romanovsky, and N. Levy, Dependability in the Web Services Architecture[C]//WADS: Architecting Dependable Systems. ICSE 2002 Workshop on Software Architectures for Dependable Systems, 2002. Springer, 2677: 90-109

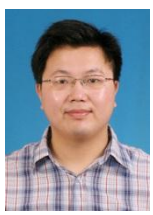
[4] K. P. Dhiraj, Fault-tolerant Computer System Design . New Jersey:Prentice Hall PTR, 1996

[5] F. Cristian, Understanding fault-tolerant distributed systems. Communications of ACM, 1991, 34(2):57-58
[6] F. B. Schneider, Implementing fault-tolerance services using the state machine approach. ACM Computing Surveys, 1990, 22(4):299-320
[7] N. Budhiraja, K. Maraulllo, F. B. Schneider, et al. Optimal Primary-backup protocols[C]//WDAG. Sixth International Workshop on Distributed Algorithm, Israel, November 2-4, 1992. Springer, 1992, 647: 362-378
[8] N. Salatge and J. C. Fabre, Fault Tolerance Connectors for Unreliable Web Services[C]//DSN. 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks(DSN'07), Edinburgh, UK, June 25-28, 2007. IEEE Computer Society, 2007, 51-60
[9] G. T. Santos, L. C. Lung, and C. Montez, FTWeb: A Fault Tolerant Infrastructure for Web Services[C]//EDOC. Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05), Enschede, September 19-23, 2005. IEEE Computer Society, 2005, 95-105
[10] M. G.Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan, Thema: Byzantine-Fault Tolerant Middleware for Web-Service Applications[C]//SRDS. 24th IEEE Symposium on Reliable Distributed Systems (SRDS 2005), Orlando, FL, USA, October 26-28, 2005. IEEE Computer Society, 2005, 131-140
[11] J. Salas, F. Perez-Sorrosal, M. Patino-Martinez, and R. Jimenez-Peris, WS-Replication: a Framework for Highly Available Web Services[C]//WWW. Proceedings of the 15th international conference on World Wide Web(WWW 2006), Edinburgh, Scotland, UK, May 23-26, 2006. ACM, 2006, 357-366
[12] D. Liang, C. Fang, and C. Chen, FT-SOAP: A Fault Tol-erant Web Service[C]//APSEC. 10th Asia-Pacific Software Engineering Conference (APSEC 2003), Chiang Mai, Thailand, December 10-12, 2003. IEEE Computer Society, 2003, 310-319
[13] D. Liang, C. Fang and S. Yuan, A Fault-Tolerant Object Service on CORBA[C]//Journal of Systems and Software. 1999, 48:197-211
[14] Z. Zheng, M. R. Lyu, "A Distributed Replication Strategy Evaluation and Selection Framework for Fault Tolerant Web Services", in Proc. 6th IEEE International Conference on Web Services (ICWS2008), Beijing, China, Sep. 23-26, 2008, pp.145-152.
[15] S. Stein, T.R. Payne, and N.R. Jennings, Flexible provisioning of web service workflows[C]//ACM Transactions on Internet Technology (TOIT). ACM, 2009, Vol 9, No. 1, Article 2



ZHU Jianfeng was born in 1985. He is a M.S. candidate at Fudan university. His research interests include Self-Adaptive Systems , Autonomous Computing, etc.

朱建锋(1985–)，男，安徽阜阳人，复旦大学硕士研究生，主要研究领域为自适应系统，自治计算等。



PENG Xin was born in 1979. He received the Ph.D. degree from Fudan University in 2006. He is a associate professor at Fudan University. His research interests include software maintenance, adaptable software, software reuse, etc.

彭鑫(1979-)，男，湖北黄冈人，2006 年在复旦大学获得博士学位，现为复旦大学计算机学院副教授、硕士生导师，主要研究领域为软件维护，自适应软件，软件复用与产品线。在 RE、ICSM、ICSR、WCRE、IST Journal、JCST、计算机学报、软件学报、电子学报、计算机研究与发展等国内外会议及期刊上发表论文 30 余篇，主持自然科学基金项目 1 项，参加 863 项目多项。CCF 高级会员，会员号:E200010133S。

ZHAO Wenyun was born in 1964. He received the M.S. degree from Fudan University in 1989. He is a professor and doctoral supervisor at Fudan University. His research interests include software engineering, electronic commerce, etc.

赵文耘(1964-)，男，江苏常熟人，1989 年在复旦大学获得硕士学位，现为复旦大学计算机学院教授、博士生导师，主要研究领域为软件工程、电子商务。在 RE、ICSM、ICSR、WCRE、IST Journal、JCST、计算机学报、软件学报、电子学报、计算机研究与发展等国内外会议及期刊上发表论文 50 余篇，主持自然科学基金项目 2 项、863 项目多项。CCF 高级会员。