ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2000/00(00)-0000-00
doi: 10.3778/j.issn.1673-9418.

E-mail: fcst@vip.163.com
http://www.ceaj.org
Tel: +86-10-89056056

基于模糊控制的应用服务器自适应策略

叶炜煜，童燕翔，马晓星

计算机软件新技术国家重点实验室 南京大学计算机科学与技术系，南京 210023

Application Server Self-Adaptation Strategy based on Fuzzy Control

Weiyu YE, Yanxiang Tong, Xiaoxing MA

State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing, 210023

Weiyu Ye, Yanxiang Tong, Xiaoxing MA. Application Server Self-Adaptation Strategy based on Fuzzy Control. Journal of Frontiers of Computer Science and Technology, 2000, 0(0): 1-000.

Abstract: Application server needs to adjust the parameters of server steadily to maintain the performance, especially in the complicated and volatile environment. However, the artificial adjustment is often difficult, time-consuming and error- prone. So there is a requirement of application server parameter self-adaption system to adjust parameters at the run-time. A method of using hill-climbing algorithm, which was optimized with fuzzy control, was designed to expand the application server. At the same time, an experiment, which deployed an online stock trading system in the server, was conducted to certificate the effectiveness of strategy.

Key words: Self-Adaptive; Application Server; Hill climbing; Fuzzy Control

摘 要: 在复杂而需求多变的实际环境中，应用服务器为了维护系统的性能需要对服务器的配置参数进行不断调整，而当前采用的人为调整总是困难低效而且易错的，因而需要一种参数自适应策略在应用服务器运行时刻动态地调整参数从而提高服务器的性能。论文采用一种基于模糊控制的爬山算法作为自适应策略对应用服务器架构进行拓展，实现了应用服务器的参数自适应，并在应用服务器中部署了在线股票交易系统进行了仿真测试，验证了自适应策略的有效性。

关键词: 应用服务器；自适应；爬山算法；模糊控制

文献标志码：A **中图分类号：**TP311

1 引言

应用服务器是一种生成应用程序且提供其运行环境的软件结构，实现这一功能的主要方式为封装包含服务所有层次的模型并生成可供开发人员调用的 API，使得软件开发者只需要处理应用的商业逻辑即可。当前较为主流的应用服务器为 Java 应用服务器，本文的研究也主要是基于此。当前 Java 应用服务器主要是为基于 Web 的电子商务应用提供服务器端的服务，提供可供设计和部署的服务以及各种软件成分来支持基于网络的复杂应用程序。然而开放动态的网络环境以及复杂的应用程序使得静态的服务器参数配置难以持续满足服务器的应用目标[1]。因此，需要在运行时对应用服务器的相关参数进行不断的调整。已有的人工调整的方式代价高昂易错，且无法保障对系统的连续监控。尤其是在网络负载出现短时间较大波动等高度动态变化时，人工方式常因为无法实时调整而失效[2]。因此，我们希望应用服务器具有一定的自适应能力，可以感知网络环境的变化(如网络负载),实时的调整配置参数从而持续满足应用目标。

本文对现有的应用服务器架构进行拓展，利用一种参数自适应策略实现应用服务器运行时参数的动态调整。基于当前较为主流的自适应框架：Monitor-Analyze-Plan-Execute-Knowledge(MAPE-K) 反馈环[3]设计出自适应模块，通过对系统的实时监控发现环境的变化，分析变化对软件系统造成的影响，并根据分析的结果制定相适应的行为，相适应的行为指的是改变系统的参数与结构，最终执行已制定好的行为从而解决环境变化对软件系统造成的影响，上述四个阶段均被额外的知识库所支持，而知识库是与系统相关，在系统启动前进行静态配置并且在系统运行期间随着系统运行状态的调整而进行即时调整。在已有工作中 ,Raghavachari M 等人[2] 和 Diaconescu A 等人[4]以经验式的思路尝试找到较优的应用服务器配置策略，但都是以人为经验进行指导 ,受到人为经验的局限。Zhang Y 等人[5]使用了模糊控制对自适应算法进行了指导，但却忽略了对自适应算法的科学设计，无法处理搜索过程中可能出现的局部最小点。本文便是基于之前工作的研究，在应用服务器的框架中拓展了一个自适应模块，该模块使用爬山算法作为自适应调整算法，并对自适应策略进行了改进，对局部最小点这一问题进行了处理，在爬山算法搜索的逐步收敛过程中使用模糊控制进行指导，并在仿真实验中验证了模块的有效性。

本文的组织结构如下：第二章对自适应算法的设计进行了说明，并且阐述了自适应算法的设计原理；第三章介绍了自适应模块的设计思路，该模块以 MAPE-K 反馈环的自适应框架为指导进行设计，对组成部分进行了详细说明，并且对各部分之间的联系进行了一定的介绍；第四章主要说明了基于自适应模块进行的仿真实验，并对实验结果进行了分析；第五章是相关工作的介绍；第六章为结束语，总结了本文的结果和意义，对未来工作的方向进行了阐述。

2 自适应算法设计

本文采用的自适应算法是在爬山算法的基础上进行一定拓展，使用模糊控制进行指导的的自适应算法。

2.1 爬山算法

爬山算法最早出现在数学分析之中，经过不断发展逐渐在计算机算法中被使用，其核心思路是通过某种策略进行迭代搜索，根据搜索策略的实际效果，不断改变搜索策略中的某个元素从而使得搜索结果向着好的趋势发展，即对某个元素的改变使得

搜索策略的结果向好的方向发展则继续在相同方向调整该元素，反之则在其他方向进行参数调整，不断迭代直到策略收敛，从而搜索到最优的调整策略。

应用服务器在架构上提供了多个可在运行时刻修改的参数，应用服务器维护人员可以根据应用服务器的当前状态即时调整参数从而保障服务器性能，我们的自适应算法就是自动的实现这一过程。在设计自适应算法时，首先需要确定的是应用服务器参数对性能的影响规律。目前的应用服务器中主要可实时调整的参数为：最大线程池、最大数据库连接池和虚拟机的最大堆等。在实际服务器中测试发现，对最大线程池这个参数进行调整对应用服务器的整体性能提升是最显著的，同时应用服务器的性能主要体现在响应时间上，所以将最大线程池作为调节的参数，将响应时间作为评价系统性能的标准。动态改变应用服务器的最大线程池参数，并对响应时间进行记录，得到两者的关系图，如图 1 所示，最大线程池对响应时间呈现下凸的影响，改变工作负担重复进行实验发现在不同工作负担下依旧呈现图 1 所示的曲线，这一变化规律符合爬山算法的应用范畴，所以本文将爬山算法作为自适应算法的基础。

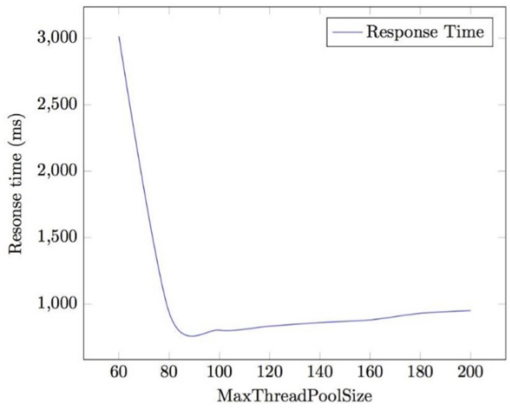

Fig.1 Relationship between RT and MTPS

图 1　最大线程池和响应时间关系图

使用爬山算法进行求解最优值的过程是逐步收敛的，我们在 Diao Y 等人[6]所提出的使用模糊控制优化收敛步骤的基础上，设计出基于模糊控制的爬山算法作为应用服务器自适应算法。

2.2 基于模糊控制的爬山算法

模糊控制是一种实时在线的优化算法[7]，模糊指的是某些事物的特征无法用绝对的真与假来刻画，转而使用部分真和部分假来刻画这样的特征，从而可以将人类世界中总结的经验与方法系统地使用在软件的设计和控制之中。模糊控制主要有三个部分：模糊化、规则匹配和清晰化。模糊化指的是将真实系统的数值映射到若干个隶属函数，称作模糊集，规则匹配指的是模糊控制器基于模糊规则推理，判断采取什么动作，清晰化就是模糊化的逆过程。通过模糊控制可以将爬山算法的历史搜索结果作为已有经验指导后续的搜索从而有效提高收敛速度。

在爬山算法应用模糊控制主要是使用 IF-THEN 形式的规则来优化寻找策略最优点这一逐步收敛的过程。IF-THEN 规则的形式主要如下：IF chgMTPS is neglarge and chgRT is neglarge, THEN nextMTPS is neglarge. 其中，chgMTPS 指的是当前最大线程池的调整方式，chgRT 指的是响应时间的变化情况，nextMTPS 指的是下一步最大线程池的调整方式，chgMTPS、chgRT 和 nextMTPS 都有自己对应的模糊集{neglarge，poslarge，zero}，对应的含义是变小，变大和不变。模糊控制使用模糊化和清晰化在真实系统的数值和语言值之间建立起映射关系，从而刻画真实系统的特征。完整的模糊控制规则如表 1 所示。

Table1 Fuzzy Control Rules

表 1　模糊控制规则

Rule	IF			THEN
	Change in MaxThrea-dPoolSize	AND	Change in Response-Time	Change in next Ma-xThread-PoolSize
1	neglarge	AND	neglarge	neglarge
2	neglarge	AND	poslarge	poslarge
3	poslarge	AND	neglarge	poslarge
4	poslarge	AND	poslarge	neglarge
5	neglarge	AND	zero	zero
6	poslarge	AND	zero	zero

表 1 所列即模糊控制规则，之前所列的规则是表中的规则 1，其余五条规则同样是在算法中使用的搜索规则。

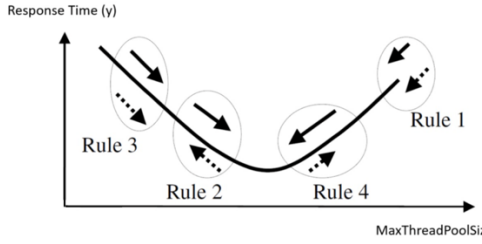

Fig.2 Hill-climbing algorithm based on fuzzy control

图 2　模糊控制指导的爬山算法

图 2 即应用模糊控制指导下的爬山算法示意图，虚线箭头表示的是当前实际采用的调整方式，实线箭头表示的是通过模糊控制生成的调整方式。依旧以规则 1 为例，将对应用服务器监控到的结果作为输入传入模糊控制模块，假设当前环境下的最大线程池大小调节策略为减小，模糊化后 chgMTPS 对应模糊集的真值为 neglarge，再假设此时应用服务器响应时间也为减小，模糊化后的 chgRT 对应的模糊集真值也为 neglarge，在规则库中进行规则匹配发现符合当前的规则 1 ,nextMTPS 对应的模糊集真值为 neglarge，清晰化后的结果是减小当前最大线程池大小，而此时系统实际采用的调整方式是减小最大线程池大小，即通过该种调整方式之后响应时间逐渐减少，说明目前的调整方式正确，之后的调整应当按照相同方向。规则 3 和规则 1 一样，当前调整方向一致，而规则 2 和规则 4 则是采取了错误的调整策略使得响应时间增大，应当采取相反的调整策略，规则 5 和 6 指的是在当前状态下略微增大或减小线程池均不会改变响应时间，即算法进入收敛，这里根据实际实验设定了阈值避免出现因震荡而导致的算法无法收敛。决策改变的幅度决定了最终收敛时间的大小以及在稳定状态下的振幅大小，所以在进行策略调整的时候在曲线陡峭的地方进行略微的调整，而在曲线为平滑的状态下进行较大幅度的调整。基于模糊控制的爬山算法的伪代码如图 3 所示。

Algorithm 1: Adaptive Tuning of Application Server

input : currentMaxPoolSize, tuningStep, tuningDir
output: currentMaxPoolSize

```
preRT ← getRt();
currentRT ← preRT;
while True do
    < tmpRT, tmpMaxPoolSize > ←
    getNextAction(tuningDir, tuningStep, tuningDir);
    if currentRT < tmpRT and currentRT < preRT then
        return currentMaxPoolSize;
    end
    if tmpRT < currentRT then
        preRT ← currentRT;
        currentRT ← tmpRT;
        currentMaxPoolSize = tmpMaxPoolSize;
    end
    tuningDir ← fuzzyCtrl(tuningDir, currentRT, tmpRT);
end
```

Fig.3 Pseudocode of Hill-climbing algorithm

图 3　模糊控制指导的爬山算法伪代码

3　自适应模块设计

通过自适应算法的原理可以看出自适应调整是一种反馈控制的过程，本文在第三章叙述的自适应算法的基础上基于实验环境中采用的中创应用服务器的实际结构，设计出了应用服务器的自适应模块(ASAT)。如图 4 所示，自适应模块主要由监测、知识库、决策制定和参数调整四部分构成。自适应模块通过对比当前应用服务器的运行状态和目标状态的差异，对下一步的调整进行相适应的指导，不断迭代该过程从而使得系统达到预期的目标状态。

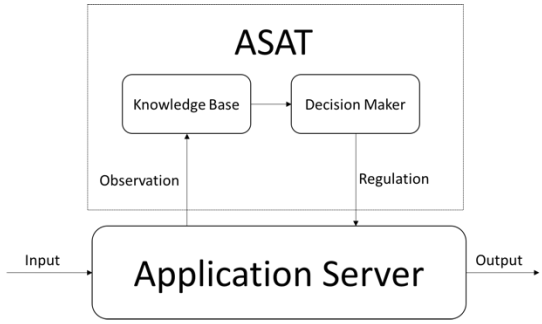

Fig.4 Application Server Adaptive Tuning

图 4　应用服务器自适应模块

3.1 监测模块

本文采用 Jmeter 监测服务器性能。Jmeter 是 Apache 组织开发的基于 Java 的开源软件压力测试工具，被多用于 Web 应用测试、服务器测试和 Java 代码测试等。但是当前版本的 Jmeter 工具将测试的结果保存在文件中，缺少直接获得服务器响应时间的 API 接口，所以在实际使用中修改了 Jmeter 的 Apache core library 文件，以 Jmeter 的 NON-GUI 模式测试应用服务器，在其将测试结果输出到控制台的同时发送给 InfluxDB 数据库 ,InfluxDB 是一个使用类似 spl 查询语言的开源数据库，监测模块对数据库进行直接访问即可获取到当前应用服务器的响应时间。通过该方法获取到的应用服务器的响应时间抖动较小，并且可持久化存储应用服务器的响应时间，从而监测模块可以有效获取到多个时间片的响应时间。

3.2 知识库和决策制定模块

知识库部分是模糊控制的规则部分，在实际设计中将模糊控制的指导过程在知识库模块中完整实现，并将其封装为函数供决策制定模块直接进行调用。实现的主要方式是采用了一个基于 Java 架构的模糊逻辑控制库 JFuzzyLite ,JFuzzyLite 是一个基于 Java 的开源模糊逻辑控制库。在模糊化过程中，隶属函数采用三角形隶属函数而不是高斯隶属函数从而提高收敛的效率。在逻辑规则库中添加如表 2 所列规则。

Table2 Rules in Logical Base

表 2　逻辑库中的规则

If chgMS is neglarge and chgRT is neglarge then nextMS is neglarg
If chgMS is neglarge and chgRT is poslarge then nextMS is poslarge
If chgMS is poslarge and chgRT is neglarge then nextMS is poslarge
If chgMS is poslarge and chgRT is poslarge then nextMS is neglarge
If chgMS is poslarge and chgRT is zero then nextMS is zero
If chgMS is neglarge and chgRT is zero then nextMS is zero

其中 chgMS 为当前的参数调整策略 ,而 chgRT 为监控的服务器响应时间，poslarge 和 neglarge 是模糊集的真值分别表示变大与变小 ,而 zero 是指当前不需要对策略进行调整，即可能已经到达了收敛的状态。通过添加如上规则，使得知识库模块可以根据当前 Input：当前最大线程池大小、响应时间、上一步最大线程池调整策略和响应时间变化结果，通过模糊逻辑库生成下一步最大线程池的调整策略并作为 Output 传递给决策制定模块 ,为爬山算法提供需要的指导，从而决策出下一步的调整策略。

决策制定模块是基于模糊控制指导的爬山算法，用以决策出当前工作负载下参数的最优值。该模块通过调用监测模块中的监控函数获取到当前的响应时间以及之前时间片响应时间的纪录，对响应时间分析，设置触发器的触发规则为：若在连续的两个时间片中，平均响应时间提高百分之二十则触发自适应策略，根据知识库模块中的模糊逻辑控制函数获得当前调整策略的方向，然后搜索出最优的调整策略，根据之前设置的寻优的调整粒度、寻优的邻域半径和当前调整方向进行最大线程池的参数设置，经过爬山算法指导的搜索找到当前最优的调整策略。本文采取的调整粒度和邻域半径是为了避

免搜索陷入局部最小点而生成次优的调整策略，邻域半径的设置是由实验情况而进行调整，邻域半径越大收敛时间越大而调整的效果越好，邻域半径越小收敛时间越小而调整的效果越差，通过实际实验调节合适的邻域半径，在相当程度上减少了寻优结果为局部最优点的概率。

3.3 参数调整模块

参数调整模块主要是调用应用服务器提供的 API 接口将参数实时地配置到服务器中，采用 HttpPost 的方式将需要调整的最大线程池的大小传递给应用服务器，应用服务器接收到相关数据对服务器的参数进行调整，整个调整过程封装成函数供决策制定模块直接进行调用。这一模块的设计较为简单，只是对应用服务器的相关 API 进行调用并封装。

4 实验

为了验证自适应模块的有效性，我们在应用服务器上运行了 Stack-Online[8]这一模拟在线股票交易系统，在模拟多个用户访问服务器的场景下对应用服务器的性能和自适应模块的调节结果进行了完整记录并对记录进行分析，从而获得了自适应模块的调节效果。本实验的基本环境详细配置如表 3 所示。

Table3 Environment configuration

表 3 环境配置

Application Server	InforSuite
Stress Test Tool	Jmeter
Benchmark	Stack-online
Monitor Tool	Jmeter and InfluxDB

实验采用的应用服务器为山东中创软件商用中间件公司自主研发的应用服务器 InforSuite Application Server，在应用服务器框架中拓展了自适应模块，使用的自适应算法和自适应模块设计如本文之前所述。实验的 Benchmark 是在应用服务器上部署了一个基于 Servlet / Jsp 的简单 Web 服务，该 Web 服务是对 Stock-Online 进行的一系列查询操作，这里的 Stack-Online 是一个模拟股票在线交易系统，提供用户股票交易、股价查询和用户所持有股票信息查询等若干项服务。实验中采用 Jmeter 进行压力测试，模拟多用户同时使用服务器提供的 Web 服务，同时采用 Jmeter 和 InfluxDB 相结合的方式进行对服务器的性能进行实时监测，自适应调整的应用服务器参数即前文提到的最大线程池大小。

本实验采用 Jmeter 模拟若干用户同时对应用服务器提供的 Web 服务进行并发请求，从而模拟出应用服务器正常运行的状态，在服务器达到稳定的时刻通过不断提高访问用户数目的方式模拟出应用服务器遭受较大运行压力的情况，同时启动自适应模块，根据存储在数据库中的响应时间和最大线程池大小绘制出随时间变化的曲线，从而可以客观的对自适应模块进行分析，实验结果如图 5 以及图 6 所示。

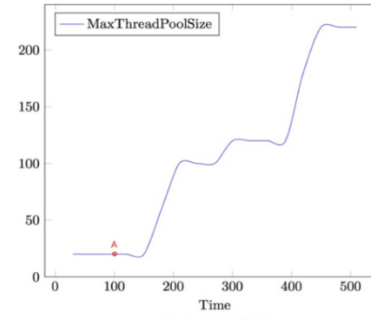

Fig.5 Graph about MTPS change

图 5 最大线程池的变化曲线图

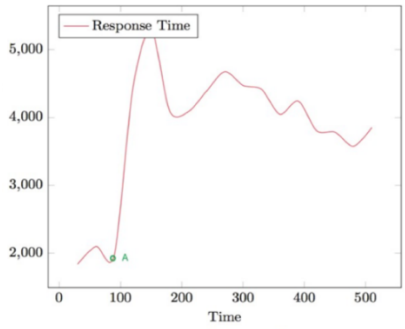

Fig.6 Graph about RT change

图 6　响应时间的变化曲线图

图 5 为应用服务器最大线程池大小随时间变化的曲线，图 6 为 Web 服务的响应时间随时间变化的曲线，在图中标注的 A 点之前，应用服务器的最大线程池大小设置为 20，通过 Jmeter 对 Web 服务模拟 200 个用户的并发请求，此时响应时间稳定在 2000ms 左右。在 A 点时刻，使用 Jmeter 增加 250 个用户的并发请求，应用服务器在面对突然增加的用户并发访问时，响应时间急剧变化增加到 5000ms 左右。自适应模块监测到响应时间的异常变化，开始调用自适应算法搜索当前负载下最大线程池的最优值并把搜索到的结果应用到服务器上，即图 5 的最大线程池大小开始调整，最终调节至 220，此时响应时间由最初的 5000ms 下降到 3500ms，并在这个区间进行小幅度的波动。

根据上述实验的结果我们可以得出这样的结论，自适应模块可以及时发现应用服务器的性能变化，并根据自适应算法对应用服务器的参数进行动态调节，从而消除服务器中的性能异常，满足应用服务器性能的要求，使得应用服务器在面对复杂网络状况和用户访问突然激增的情况下能够进行有效地处理，从而明显改善用户的访问体验和服务器的鲁棒性。

5　相关工作

目前在应用服务器中使用自适应策略进行调整已经有了一定的研究，最初 Diao Y[9]在服务器中使用 Proportional-intergral-derivative(PID)对服务器的响应时间进行优化，后来该领域的研究方法主要划分为两种思路，一种是以分析为核心，另一种是以经验指导为核心。

以分析为核心指的是采用控制理论和排队论等分析性的方法进行研究。Bergmans L 等人[10]最早建立了一个基于控制的框架来保证应用服务器的 QoS。Angelopoulos K 的团队[11]采用模型预测控制的方法，构建了 CobRA 框架在服务器中建立了自适应系统，采用控制理论的方法对需求和可行措施之间的关系进行建模，对环境中的不确定性进行了有效处理。Imre G 等人[12]和 Zhang R 等人[13]都采用排队论的方法，构建了一个以排队网络为基础的评估模型刻画了各个参数之间的关系，从而为自适应算法建立相应的规则。

以经验指导为核心的研究主要是采取各种决策算法将应用服务器看作是一个黑盒模型进行最优化决策搜索。Xi B 等人[14]同样采用爬山算法作为自适应搜索策略，但将搜索过程分为全局和局部两个阶段分别搜索，从而提高搜索的效果。Lesnik M 等人[15]采用差分遗传算法作为搜索策略进行自适应调整，并将遗传算法进行改进，使其适用于优化参数调整，减少自适应需要的对于服务器内部的结构知识，对参数最优值进行自行搜索，并生成一系列调整方案然后根据需求选择最优的自适应方案。Guo Y 等人[16][17]采用快速在线学习、神经网络和模糊控制相结合的方式对服务器参数进行自适应调节。

6　结束语

本文基于软件自适应的相关理论，设计出基于模糊控制的自适应算法对应用服务器的性能进行

了优化，使其能够更好地处理高度动态变化的工作负担，并在应用服务器架构的基础上拓展了完整的自适应模块，进行仿真实验验证了自适应模块的有效性。

在未来的工作研究中，我们将在目前的自适应算法的基础上进行更多的优化。目前的不足之处在于：算法适用于调节参数较少的系统，无法处理高维度的参数；自适应算法只是减少了搜索到局部最小点的概率，不能绝对避免这一情况。后续工作将从这两个角度出发进行更深入的研究。

References:

[1] Daniel Menascé, Riedi R, Fonseca R, et al. In search of invariants for e-business workloads. In Proceeding of the 2nd ACM conference on Electronic commerce(EC), pp.56-65, 2000.

[2] Raghavachari M, Reimer D, Johnson R D. The deployer's problem: configuring application servers for performance and reliability. In Proceedings of the 25th International Conference on Software Engineering(ICSE), pp.484-489, 2003.

[3] Kephart J O, Chess D M. The vision of autonomic computing. In Computer, Vol.36, No.1, pp.41-50, 2003.

[4] Diaconescu A, Mos A, Murphy J. Automatic Performance management in component based software systems. In Proceedings of the First International Conference on Autonomic Computing(ICAC), pp.214-221, 2004.

[5] Zhang Y, Qu W, Liu A. Automatic performance tuning for J2EE application server systems. In Proceedings of the 6th international conference on Web Information Systems Engineering(WISE), pp.520-527, 2005.

[6] Diao Y, Hellerstein J L, Parekh S. Optimizing Quality of Service Using Fuzzy Control. In Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems(DSOM), pp.42-53, 2002.

[7] Driankov D, Hellendoorn H, Reinfrank M. An introduction to fuzzy control. In Springer-Verlag, 1997.

[8] Brebner P, Chen S, Gorton I, et al. Report: Evaluating J2EE Application Servers. In Csiro Middleware Technology Evaluation, 2002.

[9] Diao Y, Hellerstein J L, Parekh S S. A Business-Oriented approach to the design of feedback loops for performance management. In Proceedings of the 12th International Workshop on Distributed Systems(DSOM), pp.11-22, 2001.

[10] Bergmans L, Halteren A V, Sinderen M V, et al. A QoS-Control architecture for object middleware. In Proceedings of the 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services(IDMS), pp.117-131, 2000.

[11] Angelopoulos K, Papadopoulos A V, Souza V E S, et al. Model predictive control for software systems with CobRA. In Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems(SEAMS), pp.35-46, 2016.

[12] Imre G, Levendovszky T, Charaf H. Modeling the effect of application server settings on the performance of J2EE web applications. In Proceedings of the 2nd international conference on Trends in enterprise application architecture(TEAA), pp. 202-216, 2006.

[13] Zhang R, Phillis Y A. Fuzzy control of arrivals to tandem queues with two stations. In Fuzzy Systems IEEE Transactions on, vol.7, No.3, pp.361-367, 1999.

[14] Xi B, Liu Z, Raghavachari M, et al. A smart hill-climbing algorithm for application server configuration.In Proceedings of the 13th international conference on World Wide Web (WWW), pp.287-296, 2004.

[15] Lešnik M, Bošković B, Brest J. Performance tuning of Java EE application servers with multi-objective differential evolution. In Proceedings of the Differential Evolution(SDE), pp.69-76, 2013.

[16] Guo Y, Lama P, Zhou X. Automated and Agile Server Parameter Tuning with Learning and Control. In Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium(IPDPS), pp.656-667, 2012.

[17] Guo Y, Lama P, Jiang C, et al. Automated and Agile Server ParameterTuning by Coordinated Learning and Control. In IEEE Transactions on Parallel & Distributed Systems, Vol.25, No.4, pp.876-886, 2014.