# 分布式系统可伸缩性研究综述

陈 斌[1] 白晓颖[1] 马 博[1,2] 黄俊飞[2]
(清华大学计算机科学与技术系 北京 100084)[1] (北京邮电大学网络技术研究院 北京 100876)[2]

**摘 要** 可伸缩性(Scalability)反映了系统可随系统需求和资源变化,持续满足性能需求的能力。在不同的场景下,可伸缩性的基本定义和度量方法能够通过不同的角度进行理解和表达。根据系统需求和运行状态,改变可用资源数量以及任务调度方式,动态调整系统性能,是系统可伸缩性实现的主要途径。分布式资源管理系统可伸缩性设计的关键技术可以从并行任务调度和分布式系统框架两个方面进行分析。可伸缩性测试是检测和评价系统性能的主要依据,并行代码测试以及可伸缩性测试系统设计的主要方法是测试技术的两个重要组成部分。随着软件范型的发展变化,软件的部署和提供逐步向基于开放、共享虚拟化资源管理平台的在线服务方式的转变,可伸缩性已成为云计算背景下软件服务的重要性能指标,进一步探讨可伸缩性在新的软件范型下所面临的挑战性问题是可伸缩性研究的新方向。

**关键词** 可伸缩性,分布式资源管理,可伸缩性度量,设计与测试
**中图法分类号** TP311.56 **文献标识码** A

## Survey on Software Scalability of Distributed Systems

CHEN Bin[1] BAI Xiao-ying[1] MA Bo[1,2] HUANG Jun-fei[2]
(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)[1]
(Research Institute of Networking Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China)[2]

**Abstract** Scalability represents the ability that a system continually satisfies the performance requirements with the changing system requirements and resources. Based on different scenarios, the basic definition and metrics of scalability can be understood and introduced from different perspectives. The main approach that systems implement scalability is to dynamically change the amount of available resources and the manner of task scheduling according to the system requirements and running status, resulting dynamic adjustment of system performance. The key technique of distributed resource management system can be analyzed from parallel task scheduling and distributed system framework. Scalability is the key issue for evaluating and measuring system performance, this paper stressed on parallel code testing and the main approaches in scalability testing system design. With the development in software diagram, the deployment and supply of software are gradually transferring to the approach of open and shared virtual resource management online service platform, scalability becomes an important guideline for software under Cloud Computing, this paper further discussed the challenging problems under new software diagram for further research purpose.

**Keywords** Scalability, Distributed resource management, Scalability measurement, Design and testing

## 1 概述

可伸缩性是分布式计算和并行计算中的重要指标,它描述了系统通过改变可用计算资源和调度方式来动态调整自身计算性能的能力。可伸缩性分为硬件和软件两个方面。硬件方面的可伸缩性指的是通过改变硬件资源来满足变化的工作负载,比如改变处理器的数目、内存和硬盘的容量。软件方面的可伸缩性是通过改变调度方式和并行化程度,来满足变化的工作负载。度量、设计与测试是系统可伸缩性研究的3个主要方面。

·可伸缩性的度量方法是设计和测试可伸缩系统的基础。但是由于可伸缩系统存在的环境复杂多样,因此准确地度量一个系统的可伸缩性是非常具有挑战性的。一般的度量方法是通过加载不同的系统资源和系统负载,来评价系统在这个过程中的性能变化。在最理想的情况下,如果同时对工作负载和计算资源进行 $K$ 倍的增长或减少,而系统或应用的平均响应时间不变,则系统拥有最优的可伸缩性。

·资源的按需动态分配和调度是系统可伸缩性的重要基础。对于传统的分布式和并行系统,在工作负载变化时,底层的资源管理机制和调度效率决定了系统应用的可伸缩性。资源调度效率高的管理机制,在工作负载增大的情况下,能够为系统和应用及时分配更多的资源,使得系统和应用的计算能

到稿日期:2010-09-16 返修日期:2010-12-15 本文受国家自然基金研究项目(61073003)资助。
陈 斌 男,硕士生,主要研究方向为软件测试、SOA,E-mail:chenb.thu@gmail.com;白晓颖 女,副教授,主要研究方向为软件测试、SOA、云计算;马 博 女,硕士生,主要研究方向为软件测试;黄俊飞 男,讲师,主要研究方向为软件测试、服务计算。

力在短时间内适应大工作负载，避免造成计算能力的大幅滑坡；在工作负载降低的情况下，及时回收限制资源，提高系统对资源的利用率，为其余的应用预留资源储备。反之，资源调度效率低的管理机制对改变资源的请求不敏感，会造成分配或者回收资源不够及时和充分。

·测试是检验和评价可伸缩性的依据。目前对可伸缩性存在着两个层次的测试：1）代码级别测试；2）系统级别的测试。代码级别的是针对并行程序中的每一个代码块，检测它对系统可伸缩性的影响。一般利用统计学的方法，对并行算法进行统计建模，根据多次实验的结果统计评价代码块对于系统性能的贡献，定义可伸缩性的权重参数，从而分析整个并行程序的可伸缩性。系统级别的测试方法是通过工作负载的预分析和实时运行状态的监控，将预分析的结果和实时的监控结果结合，分析整个系统的可伸缩性。

随着云计算等分布式计算平台技术的发展，SOA（Service-Oriented Architecture）和 Saas（Software as a Service）等新型软件开发范型的发展对可伸缩性提出了更高的要求。例如，云计算和 Saas 提出了“无限资源”的概念，为大规模应用提供海量资源共享、高可伸缩的能力。可伸缩性已成为新型服务计算模式的基础和关键，也对传统的可伸缩性度量、设计和测试技术提出了新的挑战。本文在对上述已有研究和实践工作调研和分析的基础上，进一步探讨云计算平台下服务软件的可伸缩性研究的挑战性问题。

本文在第 2 节介绍了可伸缩性的定义；第 3 节介绍了典型的可伸缩性度量；第 4 节分析了典型的资源调度方法和管理系统；第 5 节分别讨论了代码级别和系统级别的可伸缩性测试方法；第 6 节探讨了在新的软件开发泛型下可伸缩性的挑战性问题。

## 2 可伸缩性定义

André B. Bondi[2] 给出了可伸缩性的定义：可伸缩性是指网络、系统或者进程能够通过扩充自身资源来支撑更大工作负载的能力。他将可伸缩性分为 4 个类别：负载可伸缩性、空间可伸缩性、时空可伸缩性和结构可伸缩性。负载可伸缩性是指系统能够平滑地在变化的工作负载下过渡，通过调节资源来满足变化的负载，在调节的过程中不会产生超过规定限度的时间延迟和资源占用。空间可伸缩性是对系统调节资源的一个限制，即在负载增大的情况下，所占用的资源（内存、磁盘空间等）最多和负载呈线性增长，禁止了通过无穷尽地增加资源来支撑负载的做法。时空可伸缩性是指在给定的运行时间要求下（时），通过扩展资源（空）来满足增大的负载，在扩展的同时应用的运行时间依旧满足要求。结构可伸缩性是指实现的结构能够很容易地进行资源上的扩充和调节。

现有研究进一步探讨了在具体应用领域的可伸缩性定义，提出了多维度的概念，及以维度代表会影响系统行为的应用程序组成部分[4]；并且在系统需求的范围内讨论可伸缩性[3]。在文献[3]中，可伸缩性的定义是：对于特定的需求集合，在不同的负载下，系统对资源的利用率保持不变。对于可伸缩性的度量，在特定的领域下才有意义，这样也就产生了可伸缩性度量的多个维度。文献[3]中给出了 3 个维度的实例：处理效率、存储空间和接入点数目。在处理效率的维度中，系统对信息的处理效率是影响系统行为的最重要指标，它指导了系统调节资源适应负载的行为。在这个指标下，可以对可伸缩性进行更明确的度量。同样地，在存储空间和接入点数目的维度中，系统的伸缩行为受到存储空间和接入点数目的影响和指导。一些研究进一步扩展了可伸缩性的维度[5]，提出了性能、经济、物理大小、寻址、软件独立性、通信能力、技术独立性和可选择性等多个维度。

在实际应用中，可伸缩性被定义为通过增加资源使服务性能产生线性（理想情况下）增长的能力[7]。如图 1 所示，有两种增加资源的方法：一种是上扩，一种是外扩。上扩是指给单个计算节点使用性能更好、速度更快和成本更高的硬件来增加资源，包括添加更多内存、添加更多或更快的处理器，或者只是将应用程序迁移到功能更强大的单个计算机。外扩是增加计算节点的数目来增加资源。外扩增加了计算节点的数目，对管理提出了更大的挑战。由图 1 可以看出，增加资源时，外扩比上扩更容易达到最佳可伸缩性；而且外扩这种方式可以更加灵活地增加或者减少计算节点。

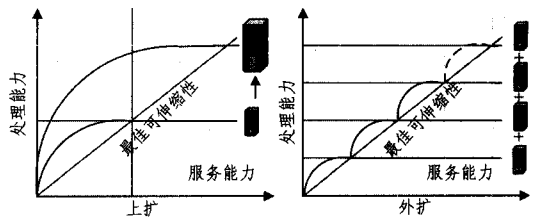


图 1 上扩和外扩

## 3 可伸缩性的度量方法

可伸缩性的度量方法是设计和测试可伸缩性系统的基础，在并行计算和分布式系统中得到广泛的讨论。早在 1990 年，Mark D. Hill[1] 首先提出使用加速度来定义可伸缩性。假设 $time(n,x)$ 是一个有 $n$ 个处理器的系统处理规模为 $x$ 的问题所需要的执行时间，加速度定义为 $speedup(n,x)=time(1,x)/time(n,x)$，即对于同样规模的问题，在采用 1 个和 $n$ 个处理器时所需的执行时间的比值。基于加速度的概念，效率定义为 $efficiency(n,x)=speedup(n,x)/n$，即速度与处理器个数的比值。如果一个系统的所有算法的效率都是 1，那么这个系统是可伸缩的。也就是说，对同样规模的问题，采用 $n$ 个处理器所需执行时间是采用 1 个处理器的 $1/n$。上述定义主要是基于在问题规模不变的条件下，研究处理器数量的改变对系统性能的改变。但当问题中存在大量顺序执行代码，且处理器足够多（例如大于问题规模）时，系统的效率很低。另外，该定义无法度量在问题大小改变时性能变化的幅度。

多数的可伸缩性度量是在并行系统中定义的，主要有基于速度的、基于加速比的和基于效率的方法。

·基于速度的方法

速度是指一个计算节点完成的工作与所需时间的比值。基于速度的方法主要有 isospeed[9] 和 isospeed-efficiency[10,23]。

Sun Xian-He 提出了 isospeed 方法来度量算法的可伸缩性。假设当有 $N$ 个处理器在工作，$W$ 是完成一个算法所需要的工作量，为了保证平均速度不变，假设当 $N'>N$ 个处理器在工作，$W'$ 是完成这个算法所需要的工作量，从系统规模 $N$ 到 $N'$ 的可伸缩性是 $\psi(N,N')=N'W/NW'$。假若处理器之

间不需要通信，每个处理器都有一份工作的拷贝，那么 $W'=N'W/N$，$\psi(N,N')=1$。一般情况下，$W'>N'W/N$，$\psi(N,N')<1$。

Isospeed 方法可以有效度量同构系统的可伸缩性。为了度量异构系统的可伸缩性，Sun Xian-He 又提出了 isospeed-efficiency 方法。在介绍 isospeed-efficiency 方法之前，需要了解几个定义：

· 一个计算节点的标记速度是指这个节点的持续速度。

· 一个计算系统的标记速度是这个系统包含的所有节点的标记速度之和。

· 假设 $S$ 是实际达到的速度，$W$ 是工作量，$T$ 是执行时间，那么 $S=W/T$。

· 假设 $C$ 是系统的标记速度，那么速度－效率 $Es=S/C=W/TC$。

假设 $C$，$W$ 和 $T$ 分别是系统最初的标记速度、工作量和执行时间；$C'$，$W'$ 和 $T'$ 分别是系统增长之后的标记速度、工作量和执行时间，并同时保证 $Es=Es'$，即 $W/TC=W'/T'C'$，那么系统的可伸缩性可以表示为 $\psi(C,C')=C'W/CW'$。

· 基于加速比的方法

加速比 $S$ 是系统使用 $k$ 个计算节点所能达到的速度与只使用 1 个计算节点时的速度的比值，理想状态下 $S(k)=k$。

著名的 Amdahl's Law[12] 给出了最基础的加速比的定义。$speedup=\dfrac{1}{r_s+\dfrac{r_p}{n}}$，其中 $r_s+r_p=1$，$r_s$ 是一个程序中顺序执行部分的比率，$r_p$ 是并行执行部分的比率。

在 Amdahl's Law 的基础上，Sun Xian-He 提出了 4 种加速比[12-14]，分别是大小固定加速比、执行时间固定加速比、内存有限加速比和普遍加速比[14]。先给出几个定义：

· 一个程序的并行度是指在给予无限的可用处理器时，在一个特定时刻可以参加计算的处理器的最大数目。

· 假设 $T_i(W)$ 是 $i$ 个处理器完成 $W$ 的工作所需要的时间。

· $W_i$ 是一个并行度是 $i$ 的程序的工作量，$m$ 是并行度的最大值，那么 $W=\sum_{i=1}^{m}W_i$。

· $\Delta$ 是一个处理器的计算能力。

(1)大小固定加速比

当有 $i$ 个处理器时，计算 $W_i$ 的时间是 $t_i(W_i)=\dfrac{W_i}{i\Delta}$。当处理器的数目无限上升时，执行时间不会无限地减少。$t_\infty(W_i)=\dfrac{W_i}{i\Delta}$ for $1\leqslant i\leqslant m$。在工作 $W$ 和无限的处理器的情况下，加速比的最大值为 $c$。在有 $N$ 个处理器的情况下，加速比为 $S_N(W)=\dfrac{T_1(W)}{T_N(W)}=\dfrac{\sum_{i=1}^{m}W_i}{\sum_{i=1}^{m}\dfrac{W_i}{i}\left\lceil\dfrac{i}{N}\right\rceil}$。假设 $Q_N(W)$ 是 $N$ 个处理器未完成 $W$ 的工作时相互通信所造成的时间延迟，那么加速比为 $S_N(W)=\dfrac{T_1(W)}{T_N(W)}=\dfrac{\sum_{i=1}^{m}W_i}{(\sum_{i=1}^{m}\dfrac{W_i}{i}\left\lceil\dfrac{i}{N}\right\rceil)+Q_N(W)}$。

这个大小固定的加速比描述了在处理器无限的情况下，完成一定工作所需的最短时间与其他因素之间的关系，适合描述工作量不会变化的情况。

(2)执行时间固定加速比

执行时间固定加速比的公式为

$$S_N(W')=\frac{T_1(W')}{T_N(W')}=\frac{\sum_{i=1}^{m'}W_i'}{\sum_{i=1}^{m'}\dfrac{W_i'}{i}\left\lceil\dfrac{i}{N}\right\rceil+Q_N(W')}=\frac{\sum_{i=1}^{m'}W_i'}{\sum_{i=1}^{m}W_i}$$

式中，$x'$ 是经过变化的系统的各项指标。执行时间固定即 $T_1(W)=T_N(W')$，即 $\sum_{i=1}^{m}W_i=\sum_{i=1}^{m'}\dfrac{W_i'}{i}\left\lceil\dfrac{i}{N}\right\rceil+Q_N(W')$。

(3)内存有界加速比

内存有界加速比的公式为

$$S_N(W^*)=\frac{\sum_{i=1}^{m^*}W_i^*}{\sum_{i=1}^{m^*}\dfrac{W_i^*}{i}\left\lceil\dfrac{i}{N}\right\rceil+Q_N(W^*)}$$

式中，$x^*$ 是经过变化的系统的各项指标，$M$ 是每个处理器的内存上界，且 $W=g(M)$，那么 $W^*=g(NM)=g(Ng^{-1}(W))$。

(4)普遍加速比

普遍加速比适合于共享虚拟内存的环境，其公式为

$$Generalized\ Speedup=\frac{W}{\dfrac{\sum_{i=1}^{p}\dfrac{W_i}{i}C_p(i,W)}{C(s,W_0)}}=\frac{W\cdot c(s,W_0)}{\sum_{i=1}^{p}\dfrac{W_i}{i}C_p(i,W)}$$

式中，$c_p(i,W)$ 是在并行度为 $i$ 的系统中 $p$ 个可用处理器在完成 $W$ 的工作时所产生的花费。

· 基于效率的方法

效率 $E$ 是指每个单独的处理器产生的加速度，即 $E(k)=S(k)/k$。

Ananth Y. Grama 等提出 isoefficiency[15,16] 方法来度量系统的可伸缩性。平行系统由一个并行架构和一个运行在上面的并行算法组成。顺序执行时间 $T_1$ 是一个算法在一个处理器上的执行时间。并行执行时间 $T_p$ 是相应的并行算法在 $p$ 个处理器上的执行时间。所有的处理器在做那些不是顺序算法的工作时花费的时间称作总开销 $T_o$，则 $pT_p=T_1+T_o$。那么加速比就是 $S=\dfrac{T_1}{T_p}=\dfrac{pT_1}{T_1+T_o}$。效率 $E=\dfrac{S}{p}=\dfrac{T_1}{T_1+T_o}=\dfrac{1}{1+\dfrac{T_o}{T_1}}$。假设 $T_1=t_cW$，$t_c$ 是执行一个操作的花费，$W$ 是问题规模，则 $E=\dfrac{1}{1+\dfrac{T_o}{t_cW}}$，经过变换得到 $W=\dfrac{1}{t_c}\left(\dfrac{E}{1-E}\right)T_o$。设 $K=\dfrac{1}{t_c}\left(\dfrac{E}{1-E}\right)$，$K$ 是基于效率的常数，则 $W=KT_o$ 这就是著名的 isoefficiency 函数。这种方法适合于组合的并行算法/结构。

并行系统中执行时间是表示效率的重要指标。并行系统中每个节点是匀称的，所以节点数目是适合的缩放属性。但是这些属性在分布式系统中不适合缩放，所以以上度量方法不能用于分布式系统中。Prasad Jogalekar 和 Murray Woodside 在 P-scalability[17] 的基础上为分布式系统提出基于成本-效率的可伸缩性度量方法[8]，其中效率是系统吞吐量和服务质量的函数。系统在伸缩因子 $k$ 的控制下基于伸缩策略来对系统进行缩放。$\lambda(k)$ 表示每秒响应的吞吐量；$f(k)$ 是每个响应的平均值，由服务质量计算得出；$C(k)$ 是每秒运行成本，则

生产率 $F(k)=\lambda(k)*f(k)/C(k)$，可伸缩性 $\psi(k_1,k_2)=F(k_2)/F(k_1)$。

## 4 分布式资源调度与管理

分布式资源管理是分布式系统和应用的基础，资源分配和调度的效率直接决定了上层系统和应用的可伸缩性。本节分析了分布式资源调度的两种模式和它们所对应的典型管理系统。

分布式资源调度围绕资源调度器，构建资源管理系统，监控各个资源的状态，将状态反馈给资源调度器作为决策的基础。根据调度器做出的分配方案，操作底层的资源为任务在不同的节点上分配资源。资源分配方案和资源的操作都可以影响系统的可伸缩性，方案决定了任务需要的资源可以得到调度的时刻，而资源的操作效率则决定了为任务调度资源所耗费的时间，二者联合决定了系统对于变化的任务需求和资源变化的反应时间，从而影响了可伸缩性。

根据是否具有资源分配的核心控制程序，分布式资源管理系统可以分为中心式和分布式两种模式。中心调度的分布式资源管理系统把所有对于资源的请求都交给核心调度程序处理，由这个核心程序来分配各个节点的资源。而分布式调度由一个计算节点接收请求，这个节点同其他的节点进行交互和协商来完成资源的分配。

### 4.1 中心调度模式

中心调度的分布式资源管理系统一般由 4 个组件构成，它们是调度器、信息存储中心、资源分配器和本地资源管理器。中心调度的系统框架一般如图 2 所示。

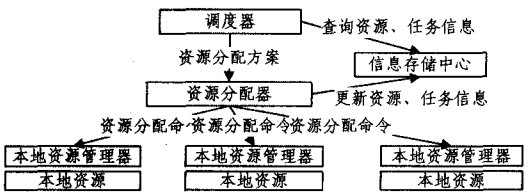


图 2 中心调度资源管理框架

在这个框架中，4 个组件的功能如下。

调度器：是资源调度请求的接收者和调度算法的执行者，它根据请求的具体内容，以信息存储中心中所存储的资源和任务的状态信息为基础，用调度算法计算得出一个资源分配方案，并且把这个方案传递给资源分配器。

信息存储中心：存储着资源和任务的实时状态信息，这些信息是调度器做出资源分配方案的依据，调度器可以通过主动查询的方式获得这些信息。

资源分配器：是资源分配方案的实际执行者，它将调度器给予的资源分配方案拆分为更小单元，即资源分配命令。每个资源分配命令是一个二元组，包括了一个资源节点的标识符和在对应资源节点上需要分配的资源数量。同时，资源分配器还负责更新信息存储中心中的信息。

本地资源管理器：是每一个资源节点上的监控者和资源分配者，它接收资源分配命令，在节点上为请求分配数量的资源，并且监控本地节点的任务运行状态和资源状态，把这些状态信息反馈给资源分配器，由资源分配器对信息存储中心的数据进行更新。

具有代表的中心调度资源管理系统有 Karl Czajkowski 等提出的应用于元计算的资源管理框架[19]、Legion[22] 和 Nimrod/G[21]。下面通过 Legion 来描述一个中心调度系统的工作流程。

Legion 的框架图如图 3 所示。

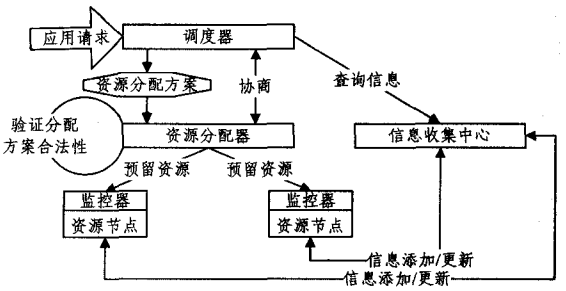


图 3 Legion 框架

在 Legion 中，资源节点利用资源预留的机制来进行本地资源的分配。即对于某一个资源分配命令，监控器把命令解析之后传递给对应的资源节点，节点根据命令为任务预留资源，在任务执行完毕之前，这些资源将不会被占用。同时，资源节点还负责更新任务和资源状态，它可以主动地向信息收集中心添加或者更新信息，信息收集中心也可以询问资源节点来获得自己需要的信息。

Legion 中，典型的资源分配流程如下。

1）决策步骤：调度器接收一个资源请求，通过解析请求的内容和查询信息收集中心中的任务和资源的状态信息，根据预先定义的调度策略来产生一个资源分配方案。

2）协商步骤：资源分配器从调度器处接收到资源分配计划，检验计划的合法性。如果合法，就进入分配步骤；如果不合法（例如在某个资源节点上需要分配超过可用数量的资源），资源分配器就与调度器进行协商，得出一个新的合法方案。协商的根据是信息收集中心中的信息和资源分配方案。

3）分配步骤：资源分配器根据合法的资源分配方案，向监控器发送预留资源的命令，监控器接到命令之后在本地为到来的任务预留所需数目的资源。在任务到达之后，监控器启动任务，并且监控本地任务的运行状态和本地资源的状态。

4）信息更新步骤：在任务开始运行之后，资源节点根据设定好的时间间隔向信息收集中心更新或者添加任务和资源的状态信息，信息收集中心也可以主动查询资源节点上的信息。

以上 4 个步骤是循环的，当信息更新步骤结束之后，新一轮的决策步骤又重新开始，为下一个请求分配任务，直到所有任务的资源调度都结束之后，这个循环才停止。

中心调度的资源管理模式在系统负载较小时能够较好地维护系统的可伸缩性，因为调度器可以从信息存储中心获得系统中所有的任务和资源的状态信息，根据一步到位的信息迅速做出决策，并把资源分配命令交给资源分配器去完成。但是当系统负载较大时，调度器需要频繁处理很多请求，造成调度器的响应变慢，系统需要较长的时间来重新调度，系统的可伸缩性较差。

### 4.2 分布式调度模式

前面提到的 Legion 资源分布框架都是基于中心调度的。中心调度的缺点也很明显，它只有一个组件能够做出资源分配的决策，这个组件很容易成为系统的瓶颈。当它出现问题

的时候，系统的资源分配将不能继续进行。分布式调度框架中有多个成员组件都能够完成资源分配的功能，从而避免了系统中有明显的瓶颈存在。分布式调度模式的资源管理系统中，节点通过相互通讯连接构成一张图，任意一个节点都能够和自己相连的伙伴进行协同，从而满足所接收到的资源请求。其框架如图 4 所示。

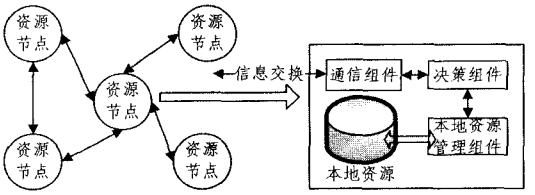


图 4　分布式调度资源管理模式框架

对于分布式调度模式中的每一个节点，都有 3 个组件。

1)通信组件：用于和其它的伙伴节点进行通讯，是资源节点之间进行协同和决策的基础。

2)决策组件：是节点的核心控制单位，决定了节点之间的协同方式和最终资源分配方案指定的算法。

3)本地资源管理组件：用于监控本地资源的状态，为决策组件提供决策的依据。

ARMS(Agent-based Resource Management System)[20]是典型的采用分布式调度模式的资源管理系统。Agent 是自主的智能软件实体，它具有自主性、社会性、反应性和适应性的特点，Agent 之间能够在目标驱动的情况下通过相互之间的交流，对外界环境的变化和事件做出响应，完成某些特定的任务。在 ARMS 中，每一个 Agent 体现了系统中的一个资源。Agent 按照层次结构组织起来，结构如图 5 所示，图中每一个节点都是一个 Agent。

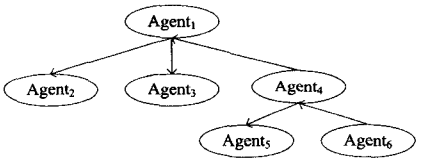


图 5　ARMS 的 Agent 层次结构

ARMS 中 Agent 分为 3 层，分别是本地资源管理层，协同层和通讯层。Agent 的结构如图 6 所示。

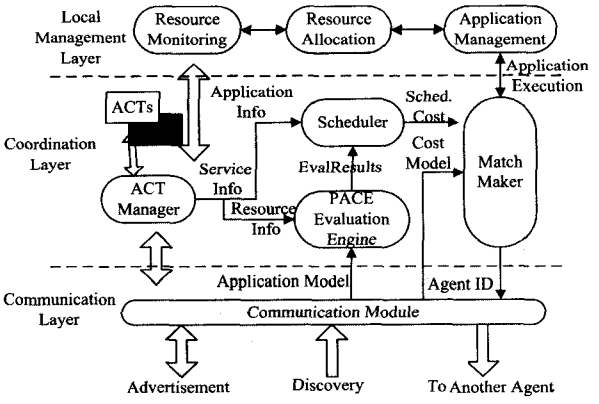


图 6　ARMS 的 Agent 结构

ARMS 中 Agent 3 层的功能如下。

通讯层：负责 Agent 同其他 Agent 的通信，信息分为 3 个类型：1) 广告(Advertisement)，一个 Agent 新加入 ARMS 的层次结构时，利用广告信息对其余 Agent 宣布自己的存在；2)探索(Discovery)，探索信息是分布式应用程序的资源的请求；3)转发(To another agent)，当 Agent 本身和它已知的父节点、子节点都不能满足一个请求时，它会将这个请求转发给自己的邻近节点，由邻近节点继续寻找合适的资源。或者当 Agent 发现自己的一个邻近节点满足请求时，这个请求也会被转发到该节点，由这个邻近节点来完成资源的分配工作。

协同层：是 Agent 的核心结构，Agent 利用 ACT(Agent Capability Table)来维护自己本身、父节点 Agent 和子节点 Agent 所代表资源的性能信息。ACT 可以分为 4 类：1)T_ACT，用来维护自身的信息；2)L_ACT，用来维护子节点 Agent 的信息；3)G_ACT，用来维护父节点 Agent 的信息；4)C_ACT，用来维护缓存的 Agent 信息。

Scheduler 通过估计请求的结束时间来进行任务的调度，调度的目的是找到一个最小的请求结束时间。请求结束时间 $T_{she}=exet+ts$，$exet$ 是估计的请求执行时间，$ts$ 是请求开始执行的时刻。PACE(Performance Analysis and Characterize Environment)引擎可以根据本地资源的情况和应用的请求来估计 $exet$。ARMS 的应用程序请求都包含一个应用模型(application model，am)，PACE 通过内部定义的 $eval$ 函数来估计 $exet$：$exet=eval(am,act)$。$ts$ 则与上一次请求的结束时间相关。Agent 估计 $exet$ 使用 ACT 的顺序是：C_ACT，T_ACT，L_ACT，G_ACT。根据估计结果，找出最小 $T_{she}$ 所对应的 ACT。如果是 T_ACT，则 Match Maker 把调度结果发送到本地管理层，进行本地的资源分配。否则，把请求转发到 ACT 对应的 Agent。如果找不到满足条件的 ACT，则将请求转发到自己的父节点和子节点，由父节点和子节点继续上面的寻找过程，直至找到满足条件的 ACT。

本地管理层：负责本地资源的监控、分配和应用程序的管理，是 Agent 和本地底层系统的接口层。

分布式调度的资源管理模式适用于系统负载较大的情况，把决策的指定分散到各个不同的节点并行进行，使得系统在大负载下仍然能够较快地对资源进行重新调度。在系统负载小的情况下，一个节点要做出调度需要同多个节点进行交流，这反而降低了决策的效率，降低了系统的可伸缩性。

## 5　可伸缩性测试现状

测试是验证和评价可伸缩性的基础。可伸缩性测试主要有两种方式：1)代码级测试。通过执行多组不同条件的测试，在每组测试中评价代码块与程序总响应时间的相关系数。对比各组测试中代码块相关系数的变化，来衡量代码块的并行度。通过各代码块的并行度评估程序的可伸缩性。2)系统级别测试。系统级别的测试通过对工作负载的估计，测试运行的实时监控和测试运行结果来综合评估程序的可伸缩性。

在多数研究中，可伸缩性只是作为一个实验的性能指标，目前针对可伸缩性的测试方法和系统本身的研究比较有限。下面分别介绍一种代码级别和一种系统级别的可伸缩性测试方法。

### 5.1　并行代码的可伸缩性测试

并行代码的可伸缩性可以通过分析每个代码块的性能得到：代码块的并行性越好，说明增加少量的处理器就可以保持

并行程序的效率；反之，需要增加更多的处理器来保持程序的效率。测试人员经常用建模的办法来分析系统或者程序的性能，根据系统和应用内部模块的组成和相互关系来构造系统的结构模型，进行性能分析。

由于并行计算系统内部的结构和组件之间的逻辑关系都比较复杂，建模十分困难且代价昂贵，且模型的正确性难以保证。为此，Gordon Lyon[24]提出了 DEX(statistically designed experiment)方法，DEX 将被测的并行程序和系统视为一个完整的实验整体，把代码片段和系统的各个参数视为实验的各个因子。基于程序的运行结果，通过监控并行程序中各个代码模块的性能指标，采用统计的方法来分析各代码块对程序运行时间的影响，得出每一个代码块对算法效率的重要程度。DEX 引入了 SP(synthetic perturbation)，用来对实验中各个因子施加扰动，这些扰动用于在系统运行时制造延时，用于构建实验因子到实验结果的映射。当某个因子 $f$ 被施加扰动时，可以通过对扰动造成延迟的分析来衡量 $f$ 对系统或者程序运行的影响程度。

DEX 的目的是对于一个被测的并行程序 $P$，找出 $P$ 中可能成为程序运行瓶颈的代码块。DEX 中，实验以组为单位进行。在同一组实验中，系统的各个因子保持不变，而对于每个代码块的扰动在持续改变，通过分析这组实验中扰动和程序运行时间的关系，得出运行时间对每个代码的敏感度 $E$，$E$ 越大的代码块，当被施加同等长度的时间延迟扰动时，造成程序运行时间的延迟越大。接着进行其他组的实验，改变系统中的因子(一般是处理器的数目)，在增加或者减少处理器数目的情况下，重新分析每个代码块的 $E$ 值。最终，对比两组实验中同一代码块的 $E$ 值，观察在处理器的增加的情况下 $E$ 的增长幅度，$E$ 增长越大的模块，说明该代码块并行度越差，成为瓶颈的可能性越大。若 $E$ 随着处理器增加而减小，则说明这个代码块的并行度较好，成为瓶颈的可能性越小。

### 5.2 可伸缩性测试系统设计

可伸缩性反映了系统可随系统需求和资源变化，持续满足性能需求的能力。可伸缩性测试系统需要关注变化的系统需求和资源与系统性能的关系，所以可伸缩性测试系统可以分为下面 4 个模块。

1)工作负载分析模块：用于实时捕获和分析动态变化的工作负载。

2)资源性能分析模块：用于实时分析底层不断变化的资源的性能。

3)系统性能监视模块：用于在运行时捕获系统性能数据，以便进一步进行分析。

4)可伸缩性分析模块：根据前 3 个模块实时捕获的数据来分析系统的可伸缩性。

STAS(Scalability Testing and Analysis System)[23]是一个典型的系统级别的可伸缩性测试系统，它由 4 个部分组成：系统特征组件、算法预分析组件、可伸缩性测试组件和可伸缩性分析组件。STAS 的特点在于它结合了算法的工作负载和最后的测试响应时间，结合负载和资源的情况对可伸缩性做出评估。同时 STAS 还支持测试人员自定义可伸缩性的测试方法。STAS 的结构如图 7 所示。

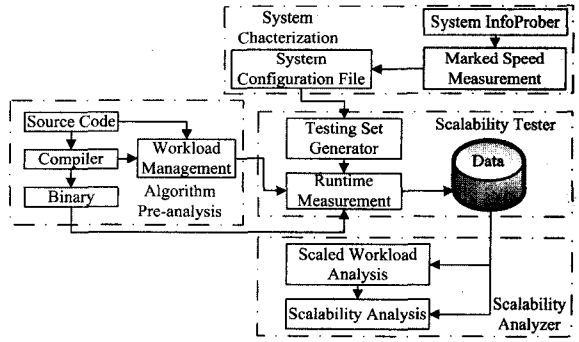


图 7 STAS 结构图

系统特征组件：其任务是获得底层系统的信息，并衡量每个节点的计算性能。它分为两个模块：系统信息探测模块和节点性能衡量模块。系统信息探测模块探测系统中的每一个节点，从中去除重复或不可用的节点，只保留活动的节点。对于活动的节点，节点性能衡量模块通过用户自定义的 benchmark 来衡量该节点的性能。最终这个模块的输出是二元组的集合：〈节点名，性能〉。

算法预分析模块：它将分布式算法的源代码作为输入，输出可执行的文件和算法的工作负载分析。算法的工作负载分析可以在编译器编译时同时完成，也可以采用用户自定义的分析方式。用户通过指定工作负载和输入数据之间的公式来人为地衡量算法的负载。

可伸缩性测试组件：它负责在运行时根据测试人员定义的方式测试系统和算法的可伸缩性。它由 3 个模块组成：被测试集生成模块、实时测量模块和数据库模块。被测试集生成模块从两个节点开始，每次将节点的规模扩大为上一次的两倍。被测试集和算法预分析模块得到的可执行文件与算法负载一并作为实时测量模块的输入，最终将测试执行的时间和这些参数一起存储到数据库模块，供分析组件进行分析。

可伸缩性分析组件：它通过数据库模块中的数据，利用 isospeed-e[9]的定义来计算算法和系统的可伸缩性。

## 6 云计算背景下系统可伸缩性问题探讨

Saas(Software as a Service)是基于互联网提供软件服务的新软件应用模式，Saas 为用户提供了完备的软件实现，用户通过互联网就可以访问服务，只需要按照自己的需求向提供商租赁服务，省去了购买硬件、开发软件和后期维护的费用。云计算是并行计算和分布式计算概念的发展，它能够为 Saas 提供可靠的基础设施。云是能够进行自我管理和维护的虚拟计算资源的集合。云计算拥有以下特点：1)可伸缩性，云的规模可以动态地进行扩展和收缩，以满足用户变化的需求；2)按需付费，用户可按自己实际的消耗来对购买的云资源进行付费；3)超大规模，云计算的规模一般达到了几十万台服务器以上；4)持久性，云计算系统能够为用户提供持久的计算资源和能力。这些不同于传统分布式和并行计算的特点，为云计算在可伸缩性方面的度量、设计和测试方面带来了新的挑战性问题。

### 6.1 云计算可伸缩性度量问题

云的可伸缩性不仅仅注重传统的扩展性，也强调云的收缩性。它的重要特点之一即按需付费，让多个云的租户在共

享云的计算资源时能够尽可能地节约费用。这就要求用户的应用在工作负载小的时候，能够把应用的资源消耗减少，在减少资源消耗的同时可以维持应用的性能，满足用户的需求。传统的可伸缩性更多着眼于代表了"伸"的可扩展性，在计算资源增加的情况下，衡量某一性能指标的增长率。与之相比，云计算同样也关注"缩"这一方面，即工作负载降低、应用性能依旧可以得到满足的情况下云能够回收的资源，提高资源的利用率，降低用户使用费用的能力。云计算的收缩性的度量不仅包括对资源增长带来的性能增长的度量，还包括对在负载减少时资源利用率提升程度的度量和用户费用减少的度量。

### 6.2 云计算可伸缩性设计问题

云计算超大规模的特点使得在同一云计算平台中的资源数目变得极为庞大，其对资源管理系统的性能要求也越高。在中心调度和分布式调度两种模式中，当云的规模扩大时，中心调度的调度器会成为系统的瓶颈。因此，分布式调度更适合云计算的范型。而分布式调度把决策工作分散到各个节点，降低了瓶颈出现的概率。与此同时产生了新的问题：资源分配是通过不同节点之间的沟通与协作进行的，多方协同的决策模式比中心式的决策更加复杂。

在分布式调度中，设计多方协同的决策模式需要注意下面的问题。

节点之间的异构问题：在超大规模的云计算系统中，会存在结构不同的资源节点。在对这些异构节点进行调度的时候，并不能直接操作这些异构节点，需要设计一个抽象的资源节点来屏蔽不同节点之间的差别，上层的操作只作用于抽象的资源节点。

节点之间通信协议的设计问题：节点之间的协议规定了交换信息的格式和方式，而这些在节点之间交换的信息构成了调度算法做出决策的依据。在设计协议时，需要考虑到协议的准确性和完备性问题。准确性能够确保节点之间交换的信息是无差错的，而完备性能够确保协议能够囊括节点之间所有可能出现的信息交换场景，避免了因为缺少对某种场景的处理而出现错误。

### 6.3 云计算可伸缩性测试问题

测试作为衡量系统伸缩性的基础，测试系统应该针对被测试系统的特点进行设计。云计算可伸缩性测试的系统应该拥有解决下面问题的能力：1)实时监控的能力，云计算具有持久性的特点，而且无法对下一刻的工作负载、资源性能进行精确的估计，所以需要测试系统能实时地捕获这些数据，综合应用性能指标来分析云计算平台的可伸缩性；2)双向测试的能力，相对于传统分布式计算只注重于扩展的能力，测试应该同时关注"伸"和"缩"两个方面，在不同的场景之下分析扩展性或者收缩性。当应用负载增加时，通过新分配资源的数目和应用性的提升来衡量扩展性能；在负载降低时，通过减少的资源消耗数目来计算衡量收缩性能。

**结束语** 可伸缩性作为云的一项重要属性，随着云计算和 Saas 的发展，它已经成为了当前研究的热点问题。可伸缩性的定义和度量方式也将随着不同的应用场景而改变。在云计算的背景下，可伸缩性的研究重点将集中在资源收缩时可伸缩性的定义和度量、基于异构资源节点的云资源分配和实时双向的可伸缩性测量。

## 参 考 文 献


[1] Hill M D. What is scalability? [J]. ACM SIGARCH Computer Architecture News, 1990, 18(4): 18-21

[2] Bondi A B. Characteristics of scalability and their impact on performance[C] // Proc. Second Int'l Workshop on Software and Performance. ACM Press, 2000: 195-203

[3] Brataas G, Hughes P. Exploring architectural scalability [C]// Proc. Fourth Int'l Workshop on Software and Performance. ACM Press, 2004: 125-129

[4] Duboc L, Rosenblum D S, Wicks T. A Framework for Modelling and Analysis of Software Systems Scalability[C] // ICSE'06. Shanghai, China, 2006

[5] Gustavson D B. The many dimensions of scalability [C]// COMPCON. 1994: 60-63

[6] van Steen M, van der Zijden S, Sips H J. Software engineering for scalable distributed applications[C]//Proc. 22nd Int'l Computer Software and Applications Conference. 1998: 285-293

[7] http://msdn. microsoft. com/zh-cn/library/aa292172(v=VS.71)

[8] Jogalekar P, Woodside M. Evaluating the scalability of distributed systems[J]. IEEE Trans. Parallel and Distributed Systems, 2000, 11(6): 589-603

[9] Sun X-H, Rover D T. Scalability of Parallel Algorithm-Machine Combinations[R]. IS- 5057. Ames Lab. , Iowa State Univ, 1991

[10] Sun X H, Chen Y, Wu M. Scalability of Heterogeneous Computing[C] // Proceedings of 34th International Conference on Parallel Processing. 2005: 557-564

[11] Amdahl G. Validity of the single-processor approach to achieving large scale computing capabilities[C] // Proc. AFIPS Conf. 1967: 483-485

[12] Sun X H, Ni L M. Scalable Problems and Memory-Bounded Speedup[J]. J. Parallel and Distributed Computing, 1993, 19: 27-37

[13] Sun X H, Ni L M. Another View of Parallel Speedup[C]// Proc. Super computing '90. Los Alamitos, Calif: IEEE Computer Soc. Press, 1990: 324-333

[14] Sun X H, Zhu J. Performance Considerations of Shared Virtual Memory Machines[J]. IEEE Trans. Parallel and Distributed Systems, 1995, 6(11): 1185-1194

[15] Grama A Y, Gupta A, Kumar V. Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures[J]. IEEE Parallel and Distributed Technology, 1993, 1(3): 12-21

[16] Grama A, Gupta A, Kumar V. Isoefficiency function: a scalability metric for parallel algorithms and architectures[R]. IEEE Parallel Distributed Technol. Systems Appl, 1993: 12-21

[17] Jogalekar P P, Woodside C M. A Scalability Metric for Distributed Computing Applicationsin Telecommunications[R]. SCE-96-07. Ottawa, Canada: Department of Systems and Computer Enginering, 1997

[18] Kumar V, Gupta A. Analyzing the scalability of parallel algorithms and architectures: A survey[C] // Proceedings of the 1991 International Conference on Supercomputing. 1991

[19] Czajkowski K, Foster I, Karonis N, et al. A Resource Management Architecture for Metacomputing Systems[J]. Information Sciences, 1-19

[20] Cao J, Jarvis S A. ARMS: An agent-based resource management system for grid computing[J]. Scientific Programming, 2002, 10: 135-148

[21] Buyya R, Abramson D, Giddy J. Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid[J]. Computer, 2000: 1-7

[22] Chapin S J, Katramatos D, Karpovich J, et al. Resource Management in Legion[J]. Future Generation Computer Systems, 1999, 15(5): 583-594

[23] Chen Y, Sun X. STAS: A Scalability Testing and Analysis System[C]// 2006 IEEE International Conference on Cluster Computing. 2006: 1-10

[24] Lyon G, Kacker R, Linz A. A scalability test for parallel code [J]. Software: Practice and Experience, 1995, 25(12): 1299-1314

[25] Liu C L, Layland J W. Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment[J]. J. ACM, 1973, 20 (1): 46-61

[26] Schmid U, Blieberger J. Some investigations on FCFS scheduling in hard real-time applications[J]. Journal of Computer and System Sciences, 1992, 45(3): 493-512

[27] Homayoun N, Ramanathan P. Dynamic priority scheduling of periodic and aperiodic tasks in hard real-time systems[J]. Real-Time Systems, 1994, 6(2): 207-232

[28] Biyabani S R, Stankovic J A, Ramamitham K. The Integration of Deadline and Criticalness in Hard Real-Time Scheduling[C]// Real-Time Systems Symposium. 1988

[29] Jensen E D, Locke C D, Tokuda H. A Time-driven Scheduling Model for Real-time Operating Systems[C]// IEEE Real-Time Syst. Symp. 1985: 112-122

[30] Buttazzo G, Spuri M, Sensini F. Value vs. deadline scheduling in overload conditions[C]// Proceedings of the 16th IEEE Real-Time Systems Symposium. 1995

[31] 金宏，王宏安，王强，等. 一种任务优先级的综合设计方法[J]. 软件学报，2003，14(3)：376-382

[32] 王永炎，王强，王宏安，等. 基于优先级表的实时调度算法及其实现[J]. 软件学报，2004，15(3)：360-370

[33] 黄德才，钱能. 多机相关任务均衡调度问题的复杂性与新算法[J]. 计算机工程与科学，2000，22(2)

(上接第 16 页)

[40] Schmid S, Sifalakis M, Hutchison D. Towards Autonomic Networks[J]. Lecture Notes in Computer Science, 2006, 4195: 1-11

[41] Baumgarten M, Bicocchi N, Kusber R, et al. Self-organizing Knowledge Networks for Pervasive Situation-aware Services[C]// IEEE International Conference on Systems, Man and Cybernetics. Quebec, Canada, October 2007: 1-6

[42] Wang Hui-qiang, Feng Guang-sheng, Zhao Qi-an, et al. Progress of Research on Cognitive Networks[M]. Sciencepaper Online, 2009

[43] Hillston J. Fluid flow approximation of PEPA models[C]// Proceedings of the 2nd International Conference on Quantitative Evaluation of Systems. Torino: IEEE Computer Society Press, Sep. 2005: 33-42

[44] Katsuno Y, Aihara T. Autonomic Network Configuration for Networkable Digital Appliances[J]. IEEE Transactions on Consumer Electronics, 2005, 51(2): 494-500

[45] Devroye N, Mitran P, Tarokh V. Achievable Rates in Cognitive Radio Channels[J]. IEEE Transactions on Information Theory, 2006, 52(5): 1813-1827

[46] Sahai A, Hoven N, Tandra R. Some Fundamental Limits on Cognitive Radio [C] // Forty-second Allerton Conference on Communication, Control, and Computing. Monticello, Israel, October 2004: 1-11

[47] Jovicic A, Viswanath P. Cognitive Radio: An Information-Theoretic Perspective[OL]. http://www.ifp.uiuc.edu/~jovicic/JV06.pdf, June 2006

[48] Koulouriotis D E, Diakoulakis I E, Emiris D M, et al. Development of Dynamic Cognitive Networks as Complex Systems Approximators: Validation in Financial Time Series[J]. Applied Soft Computing, 2005, 5(2): 157-179

[49] Thomas R W, Friend D H, Dasilva L A, et al. Cognitive Networks: Adaptation and Learning to Achieve Eend-to-End Performance Objectives[J]. Communications Magazine, 2006, 44 (12): 51-57

[50] Kephart J O. Research Challenges of Autonomic Computing[C]// Proceedings of the 27th International Conference on Software Engineering. Missouri, USA, May 2005: 15-22

[51] Strassner J. Autonomic Networking: Theory and Practice[C]// Proceedings of 2008 IEEE/IFIP Network Operations and Management Symp. Salvador, Brazil, April 2008: 786- 786

[52] Hinchey M, Sterritt R. Autonomicity-an Antidote for Complexity? [C]// Proceedings of Computational Systems Bioinformatics Conference, Workshops and Poster Abstracts. Stanford University, Aug. 2005: 283-291

[53] Gelenbe E, Lent R. Power-aware Ad-hoc Cognitive Packet Networks[J]. Ad-hoc Networks, 2004, 2(3): 205-216

[54] Gelenbe E, Lent R, Xu Z. Measurement and Performance of a Cognitive Packet Network[J]. Computer Networks, 2001, 37 (6): 691-701

[55] Gelenbe E, Lent R, Xu Z. Design and Performance of Cognitive Packet Networks[J]. Performance Evaluation, 2001, 46(2/3): 155-176

[56] Hey L A. Reduced Complexity Algorithms for Cognitive Packet Network Routers[J]. Computer Communications, 2008, 31(16): 3822-3830

[57] Koulouriotis D E, Diakoulakis I E, Emiris D M, et al. Development of Dynamic Cognitive Networks as Complex Systems Approximators: Validation in Financial Time Series[J]. Applied Soft Computing, 2005, 5(2): 157- 179

[58] Rondeau T W, Bostian C W, Bruce A F. Cognitive Techniques: Physical and Link Layers[M]. Cognitive Radio Technology. Newnes: Burlington, 2006: 219-268

[59] Smith J M, Bruce A F. Cognitive Techniques: Network Awareness[M]. Cognitive Radio Technology. Newnes: Burlington, 2006: 299-311