面向负载均衡的自主式虚拟机动态迁移框架

孙冬冬 柳 青 武旖旎
(解放军61579部队 北京 102400)

摘 要 借鉴蚁群算法的思想,提出了一种面向负载均衡的自主式虚拟机动态迁移框架,该框架不需要中央管理模块,能够实现服务器的自主迁移,避免了单点失效。利用智能蚂蚁的搜索,实现了自主式框架的迁移机制,而且使用模糊逻辑推理,根据系统的负载状况自动地调整智能蚂蚁的搜索半径来提高搜索性能。最后扩展了云计算平台CloudSim,实现了提出的虚拟机自主式迁移框架。在扩展后的CloudSim平台上进行的仿真实验验证了该框架的可行性。确定了合适的框架参数,并且通过仿真实验与比较分析,验证了所提出的自主式虚拟机迁移框架具有良好的负载均衡效果。

关键词 云计算,负载均衡,虚拟机迁移,蚁群算法
中图法分类号 TP301 **文献标识码** A

Load Balancing-oriented Autonomous Live Migration Framework for Virtual Machine

SUN Dong-dong LIU Qing WU Yi-ni
(Unit 61579 of PLA, Beijing 102400, China)

Abstract This paper proposed an autonomous dynamic virtual machine migration framework for load balancing based on the idea of ant colony algorithm. The framework does not need central management module so that servers can achieve autonomous virtual machines migration, avoiding the single point of failure. The migration mechanism of the framework is achieved making use of intelligent ants and the ants' searching radius can be adjusted automatically according to the load of the system using fuzzy logic reasoning in order to improve the searching performance. At last, this paper extended the cloud computing simulative platform CloudSim to achieve the autonomous virtual machine migration framework proposed in this paper. Experiments were carried out in the extended platform to verify the feasibility of this framework. The parameter of framework was setted, and the excellent load banlancing ability of this framework was demonstrated by analyzing and comparing the results of simulation.

Keywords Cloud computing, Load balancing, Migration of virtual machines, Ant colony algorithm

近年来,随着计算机硬件和软件的飞速发展,计算模型不断发生演化,继分布式计算、并行计算和网格计算等模型后,学术界提出了云计算模型[1]。云计算模型是一种商业的计算模型,它将各种任务分布在由大量计算机构成的动态资源池上,使用户能够按照需要获取信息服务、计算力和存储空间。

为了更好地组织维护云计算数据中心庞大的IT基础设施资源,云计算系统引入了虚拟化[2]技术,通过将一台物理服务器分割为若干个相互隔离的虚拟服务器,实现对物理资源的动态分割,提高了系统的资源利用率,降低了管理难度,同时屏蔽了底层硬件资源的异构性。它使各虚拟机间相互独立,某台虚拟机遭受攻击或出现故障不会影响同一平台中其它虚拟机的正常运行。但是,在数据中心中,负载的变化是不确定的,有的物理服务器的负载较高,而有的物理服务器的负载较低,为了实现整个系统的负载均衡[3],可以借助虚拟机的动态迁移技术[4]进行虚拟机和物理服务器的重映射,将包括操作系统在内的整个运行环境从一台物理服务器迁移到另一台物理服务器上。通过虚拟机动态迁移机制,能够有效管理数据中心的资源[5],适时地调整过载的服务器,提高服务等级协议(Service Level Agreement, SLA)[6,7]。

1 虚拟机动态迁移

虚拟化技术的一个重要应用就是虚拟机的动态迁移,虚拟机动态迁移能够保证虚拟机在运行的情况下,实现虚拟机从一个物理服务器到另一个物理服务器的迁移,虚拟机的动态迁移有助于实现数据中心的负载均衡、服务器的维护和升级等。国内外对虚拟机的动态迁移进行了大量的研究。

Khanna等[8]提出监控服务器和虚拟机的资源(CPU和内存)使用情况的策略,一旦发现某台服务器资源的使用超过了提前设定的阈值或是某些用户服务等级协议SLA受到威胁,系统就进行虚拟机迁移,将服务器中的某台虚拟机迁移到另一台主机中。Sandpiper系统[9]为虚拟机迁移提出了黑盒和灰盒策略,能够自动地进行资源监控和热点侦测,一旦发现

到稿日期:2013-06-05 返修日期:2013-09-25
孙冬冬(1975—),女,讲师,主要研究方向为计算机软件应用,E-mail:691759571@qq.com;柳 青(1981—),女,讲师,主要研究方向为计算机信息安全防护;武旖旎(1982—),女,助理讲师,主要研究方向为保密通信。

热点，就进行虚拟机迁移，实现服务器和虚拟机的重映射。Bobroff 等[10]提出了一种虚拟机迁移算法，该算法主要用来保证随机的服务等级协议。复旦大学的刘鹏程提出了一种灵活的、能够很好满足服务等级协议的动态迁移框架，该框架的整体架构如图 1 所示。该框架由中央控制引擎和位于各个服务器上的本地迁移引擎组成，中央控制引擎从整体上掌握数据中心中各平台资源的使用情况，根据策略来发起虚拟机的迁移，实现数据中心整体范围内的负载均衡，提高资源利用率。

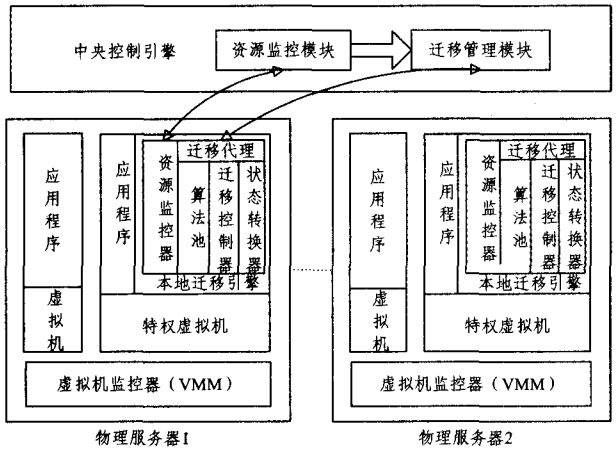

图 1　虚拟机动态迁移框架

以上的虚拟机迁移方案采用的都是集中式的迁移框架，由中央管理模块负责整个数据中心的迁移决策，因此它的故障会造成整个数据中心虚拟机迁移的失效，而且中央管理模块需要监控所有服务器的资源使用状态来进行迁移决策，这容易造成系统瓶颈，降低系统的可靠性。为了解决这一问题，可以考虑设计自主式的虚拟机迁移框架，由服务器自行监控资源使用，若发现超载则自己制定迁移决策，实施虚拟机迁移。这种自主式的虚拟机迁移框架不需要中央管理模块的参与，可以克服集中式框架中单点失效的缺陷。

2　自主式迁移框架设计

本节借鉴蚁群算法的思想提出了一种面向负载均衡的自主式虚拟机动态迁移框架，如图 2 所示，该框架由网络互连的自治服务器组成，每台服务器上都有一个迁移模块，能够实施虚拟机迁移的相关决策，如何时发起虚拟机迁移、如何寻找合适的目标物理服务器等。从图 2 中可以发现，该框架的主要特点是没有中央管理模块，框架的服务器结点通过迁移模块自主迁移，组成分布式的迁移框架。

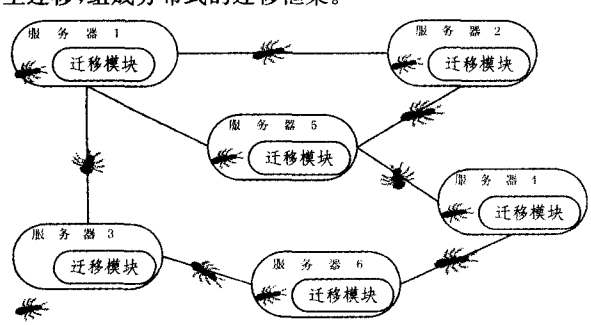

图 2　自治服务器网络视图

图 3 表示了每个服务器中迁移模块的体系架构，其主要包括 6 个组成部分：蚂蚁服务模块(AntService)、决策模块、策略库、负载模块、资源管理模块和通信层。

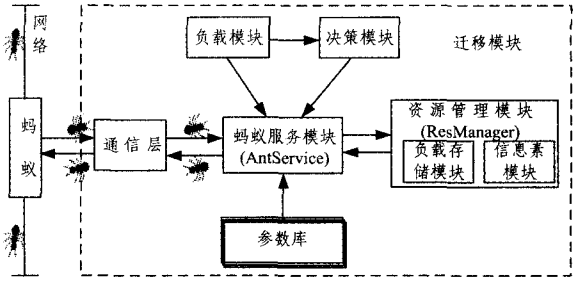

图 3　迁移模块的体系架构图

通信层负责中转各节点之间的通信，根据策略进行新节点的探索，进行网络拓扑的维护。负载模块实施负载策略，负责对服务器的资源使用进行监测，并根据当前情况和历史负载信息预测服务器未来的负载信息，防止误判而产生不必要的迁移，浪费服务器资源，并影响服务质量。负载模块收集资源使用数据和预测数据进行综合计算，并将所得的负载数据发送给决策模块进行迁移决策，以确定服务器的负载状态。参数库包含了框架中的相关参数，包括负载状态域阈值、搜索蚂蚁的步长及蚂蚁的内存空间。

2.1　资源管理模块

资源管理模块主要包括两部分：负载存储模块和信息素模块，负载存储模块主要维护服务器 neighbors 的负载信息(neighbors 和其对应的负载值)，而信息素模块主要维护服务器 neighbors 的信息素(neighbors 和其对应的信息素值)。

负载存储模块中数据的更新主要靠蚂蚁来完成，当蚂蚁到达服务器时，蚂蚁服务模块实施蚂蚁程序，将蚂蚁中携带的其他服务器及其负载信息更新到本地服务器的负载存储模块中，负载信息是信息素更新的依据。负载存储模块的负载信息存放在一个组织良好的 XML 文件中，如下例所示。

```
〈?xml version="1.0" encoding="GB2312"?〉
〈neighbors-load〉
〈neighbors〉
〈id〉 0〈/id〉
〈load〉 0.25〈/load〉
〈/neighbors〉
〈neighbors〉
〈id〉 1〈/id〉
〈load〉 0.5〈/load〉
〈/neighbors〉
〈/neighbors-load〉
```

在上述例子中，定义了服务器的两个 neighbors，其中一个 neighbors 的 id 号为 0，负载均衡值为 0.25，而另一个 neighbors 的 id 号为 1，负载均衡值为 0.25。

信息素模块则运行着信息素更新程序，根据策略确定信息素的挥发系数，进行信息素的更新。同样，信息素模块的信息素信息也存放在一个组织良好的 XML 文件中，如下例所示。

```
〈?xml version="1.0" encoding="GB2312"?〉
〈pheromone〉
  〈coefficient〉 0.1 〈/coefficient〉
〈/pheromone 〉
〈neighbors〉
  〈id〉 0 〈/id〉
  〈pheromone_value〉 6 〈/pheromone_value〉
```

```
</neighbors>
<neighbors>
  <id> 1 </id>
  <pheromone_value > 5</pheromone_value >
</neighbors>
```

在上述例子中，定义的信息素的挥发系数为 0.1，也定义了两个 neighbors 的信息素值，id 号为 0 的 neighbors 的信息素值为 6，而 id 号为 1 的 neighbors 的信息素值为 5。

2.2 蚂蚁服务模块

蚂蚁服务模块 AntService 是该架构的主体部分，负责与其它部分进行交互，实现虚拟机的自主式迁移。蚂蚁到达服务器后，由蚂蚁服务模块下载算法并实施，在算法实施过程中与其它各个模块进行交互，获得算法需要的数据。蚂蚁服务模块的具体实施算法如表 1 所列。

表 1 蚂蚁服务模块的实施算法

```
public class AntService {
  ResManager getResManager(){}; //获得资源管理对象
  Load getLoad(){}; //获得负载对象
  Algorithm getAntAlgorithm(){}; //下载蚂蚁算法
  Policy getPolicy(){}; //获得策略对象
  Ant produceAnt(){}; //产生搜索蚂蚁
  initAnt(){}; } //蚂蚁的初始化
public class ResManager {
  float getLoadStorage(){}; //得到各个 Neighbors 的负载
  void updateLoadStorage(){}; //更新资源管理层中 Neighbors 的负载
  float getPheromone (){}; //得到各个 Neighbors 的信息素
  void updatePheromone(){}; //更新 Neighbors 的信息素
  int[] getNeighborsId(){}; //得到存储的其它服务器的 ID
  void addNeighbors(){}; //将搜索蚂蚁中记录的服务器 ID 存入资源管理层中
public class ResManager {
  float getLoadStorage(){}; //得到各个 Neighbors 的负载
  void updateLoadStorage(){}; //更新资源管理层中 Neighbors 的负载
  float getPheromone (){}; //得到各个 Neighbors 的信息素
  void updatePheromone(){}; //更新 Neighbors 的信息素
  int[] getNeighborsId(){}; //得到存储的其它服务器的 ID
  void addNeighbors(){}; } //将搜索蚂蚁中记录的服务器 ID 存入资源管理层中
public class Policy {
  Interger getCounter(){}; //得到搜索蚂蚁的步数
  float getThreshold(){}; } //得到阈值
  float getMem(){}; //得到蚂蚁的内存空间
public class Load() {
  float getSeverLoad(){}; //得到服务器的负载;
  float getPredictedLoad(){}; } //得到服务器的预测负载
```

蚂蚁在框架中扮演着非常重要的角色，它负责服务器节点之间的相互通信，蚂蚁一旦产生，便会在服务器节点之间“游荡”，每经过一个服务器节点，便会检查该服务器的资源使用情况，若服务器满足要求，则蚂蚁将节点的资源状况进行记录，然后接着进行搜索，直到蚂蚁的步数(counter)为 0。搜索完成后，蚂蚁便返回到源服务器节点进行匹配，寻找合适的服务器进行迁移。

在网络中，每个服务器节点都有一个唯一标识符，为了与其它节点进行通信，必须知道它们的唯一标识符。服务器节点所记录的所有其它节点称为该服务器节点的 neighbors，它是动态的，因为服务器可能会发现新的其它服务器节点而增加 neighbors，或发现不可到达的服务器节点而删除 neighbors。本文中，服务器节点 neighbors 的维护通过搜索蚂蚁来实施，蚂蚁每到一个服务器节点，会将已经搜索记录的物理节点信息存储到该节点中，为后续蚂蚁提供更高效的搜索。

2.3 迁移机制

蚂蚁是该框架的通信媒介，负责系统中各个节点间的通信交流，它能发现过载的服务器和低负载的服务器，并通过迁移虚拟机达到服务器间的负载均衡。算法是蚂蚁的灵魂，其具体的算法实现如表 2 所列。

表 2 蚂蚁的实施算法

```
public class Ant {
  Interger counters; //蚂蚁的剩余步数
  List<Server> serverList; //服务器列表
  Vm vm; //需要迁移的虚拟机
  float M; //判断服务器是否满足要求的阈值
  AntService antservice; //蚂蚁服务对象
  float probability;
  Server nextServer;
method doAntImplement() {
  ResManager resManager=antservice. getResManager();
  probability=antservice. getPro();
  serverlist=antservice. getServerList()
  void updateResManager(resManager); // 更新资源管理器的相关信息
  if (checkServer()) { // 根据 M 来判断服务器是否满足要求
    serverlist. add();
    goNextServer();}
  else
    goNextServer();}
method goNextServer() {
  if ( random(probability)) {
    nextserver=doSearchNewServer(); } / / 搜寻新节点
  else {
    list=getOrderedList(serverlist); //对列表中的服务器进行排序
    nextserver=serverlist. get[0];
    move(nextserver); }}
```

为了降低服务器间的通信负载，本文实施了缓存机制，用版本号对不同的算法进行唯一标识，当蚂蚁到达服务器时，服务器首先检查状态数据中的版本号，若服务器已经缓存了该版本号的蚂蚁算法，则无需下载，当新版本的蚂蚁算法出现时，服务器便会更新本地缓存的算法。

服务器为蚂蚁分配了一定的内存空间 Mem，用来记录搜索过的环境(服务器和虚拟机)信息。蚂蚁的内存空间分为 3 个列表：过载列表(OverLoaded List)、低负载列表(UnderLoaded List)和负载列表(Load List)，其中过载列表中记录着负载过高的服务器信息及该服务器中待迁移的虚拟机信息，低负载列表中则只记录着负载较低的服务器信息，而负载列表中则记录着蚂蚁搜索过的所有服务器的负载信息，负载列表中的信息被用来对服务器的 neighbors 进行维护。

该框架的效率依赖于蚂蚁的数量 n 和步长 m。如果蚂蚁的步长过短，它会频繁地进行搜索，如果步长过大，蚂蚁的搜索时间变长，会降低进行搜索的频率。因此蚂蚁的步长应该根据系统的负载动态确定，使之能够适应不断变化的系统环境。

2.3.1 蚂蚁的产生

如果服务器超载，成为资源热点，就会产生搜索蚂蚁。本文采用负载状态域模型，根据服务器的负载均衡值，将其分为 4 个状态域：低负载域、平衡域、过载域和超载域，如图 4 所示。

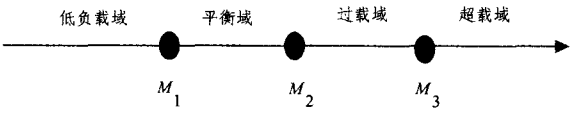

图 4 服务器的负载状态域

其中，M_1 为平衡域阈值，M_2 为过载域阈值，M_3 为超载域阈值，当服务器处于超载域时，即被确定为资源热点，产生搜索蚂蚁进行负载均衡操作。

蚂蚁产生后首先进行初始化，包括步长 m、过载列表、低负载列表、负载列表、过载列表的个数 OverLoadedCount(OLC)、低负载列表的个数 UnderLoadedCount(ULC)、负载列表的个数 LoadedCount(LC)和算法。初始化后便会在服务器节点之间进行搜索，每经过一个服务器节点，便会检查该服务器的资源使用情况，若服务器过载，蚂蚁便将服务器和其上待迁移虚拟机的信息记录到过载列表中，且过载列表个数 OLC 加 1；若服务器负载较低，便将服务器的信息记录到低负载列表中，且低负载列表个数 ULC 加 1；而负载列表则总是记录蚂蚁搜索过的每个服务器节点，相应的负载节点个数 LC 总是加 1。然后蚂蚁接着进行搜索，直到搜索结束条件满足(步长 m 为 0)。搜索完成后，蚂蚁便进行待迁移虚拟机与候选服务器的匹配，寻找合适的服务器进行迁移。

2.3.2 服务器搜索

蚂蚁搜索服务器时，并不是完全随机地进行寻找，它根据服务器节点的信息素浓度、负载均衡值和网络特性决定下一个游荡的节点。本文借鉴了蚁群算法中信息素[11]浓度的思想，但是其应用的方法不同，蚁群算法中，信息素浓度越大，该路径被选择的概率就越大。但是，在本文的应用场景中，蚂蚁经过某一路径后，表示该路径上的节点已经进行了负载均衡操作，为了使蚂蚁能尽量搜寻所有的节点，达到整体的负载均衡，本文对信息素的应用为：信息素浓度越大被选择的概率越小，初始情况为每个节点拥有相同的信息素浓度。为了进行高效的负载均衡，蚂蚁进行服务器搜索时，优先选择处于低负载域和过载域的服务器。因此，蚂蚁在节点 i 选择节点 j 的概率如下式所示。

$$p(i,j)=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}[net_{ij}(t)]^{\beta}\left|L_{host(j)}-\dfrac{M_1+M_2}{2}\right|^{\gamma}}{\sum\limits_{k\notin advoids(t)}[\tau_{ik}(t)]^{\alpha}[net_{ik}(t)]^{\beta}\left|L_{host(k)}-\dfrac{M_1+M_2}{2}\right|^{\gamma}} & j\notin advoids(t)\\ 0, & j\notin advoids(t)\end{cases}$$

式中，$\tau_{ij}(t)$ 表示节点 i 观察到的节点 j 的信息素浓度；$advoids(t)$ 表示时刻 t 时蚂蚁的禁忌列表，蚂蚁每经过一个节点就将该节点加入到禁忌列表中以防止进行重复搜寻；$net_{ij}(t)$ 用来考虑节点 i 和节点 j 之间带宽的影响，带宽越大，节点被选择的概率越大；$\left|L_{jhost}-\dfrac{M_1+M_2}{2}\right|$ 代表节点 j 的负载因子，表示负载越高或越低，其被选择的概率越大，以进行高效的服务器搜索。α、β 和 γ 体现了信息素、网络带宽和负载在服务器选择中的相对重要性，$\alpha\geqslant 0$，$\beta\geqslant 0$，$\gamma\geqslant 0$。

为了及时反应信息素的更新变化，本地更新策略会实时地修改节点的信息素浓度。全局信息素更新方式如下式所示：

$$\tau_{ij}(t+1)=(1-\rho)*\tau_{ij}(t)$$

其中，ρ 是全局信息素挥发参数，且 $0<\rho<1$

2.4 蚂蚁级的负载均衡

当蚂蚁进行服务器搜索时，可能会遇到两只或多只蚂蚁到达同一台物理服务器的情况，每只蚂蚁记录了自己搜索到的服务器的相关信息，而且各自收集的信息量未必相同。比如两只蚂蚁在某个服务器相遇，其中一只可能记录了 4 台过载服务器、3 台低负载服务器和 9 台服务器的负载信息，而另一只可能记录了 6 台过载服务器、5 台低负载服务器和 12 台服务器的负载信息。为了进行更高效的搜索，相遇的蚂蚁之间可以进行信息交互。在上述的例子中，蚂蚁交互后，每只蚂蚁都变成记录了 5 台过载服务器、4 台低负载服务器和 21 台服务器的负载信息。实际上，当蚂蚁间进行信息交互时，它们扩大了自己的搜索范围，这有助于进行全局的服务器搜索，从而提高整个框架的迁移性能。

而且当蚂蚁进行服务器搜索时，若蚂蚁搜索结束条件未满足($m>0$)，蚂蚁的内存完全被占用，这时蚂蚁会继续搜索，若蚂蚁搜索的下一节点为过载域节点，这时会产生新的蚂蚁进行服务器搜索，若下一节点为低负载域节点或平衡域节点，蚂蚁会将其负载与低负载列表中的最大负载值进行比较，若该节点的负载大于低负载列表中的最大负载值，则丢弃该服务器节点，不进行任何记录；而若节点负载小于低负载列表中的最大负载值，则进行低负载列表和负载列表的服务器信息替换。这样可以提高负载均衡性能和适应系统负载变化的能力。

2.5 匹配迁移

蚂蚁完成搜索后，内存中的过载列表记载着待迁移的虚拟机列表$\{VM_1, VM_2, \cdots, VM_i\}$，低负载列表中记录着目标服务器列表$\{Ser_1, Ser_2, \cdots, Ser_i\}$，其中，$i+k\leqslant m$，$m$ 为蚂蚁的游荡步数。因为采用了蚂蚁交互，待迁移虚拟机的个数与目标服务器的个数一般是比较接近的。接下来进行虚拟机与服务器的匹配，决定迁移方案。

本文提出适合度 $Fitness_{Ser\text{-}Vm}$(以下简写为 Fit_{ij})概念，适合度是指待迁移虚拟机需求性能和服务器空闲性能之间的适合程度，公式为：

$$Fitness(VM_i, Ser_j)=\frac{SerA_{cpu}-Vm_{cpu}}{Vm_{cpu}}\frac{SerA_{memory}-Vm_{memory}}{Vm_{memory}}\frac{SerA_{net}-Vm_{net}}{Vm_{net}}$$

其中，$SerA_{cpu}$、$SerA_{memory}$ 和 $SerA_{net}$ 分别代表服务器 j 剩余的 CPU、内存和带宽资源。Vm_{cpu}、Vm_{memory} 和 Vm_{net} 则分别代表虚拟机 i 需要的 CPU、内存和带宽资源。若 $Fitness_{Ser\text{-}Vm}<0$，则说明虚拟机需要的资源超过了服务器的剩余资源。

虚拟机与服务器进行匹配的算法流程如图 5 所示。

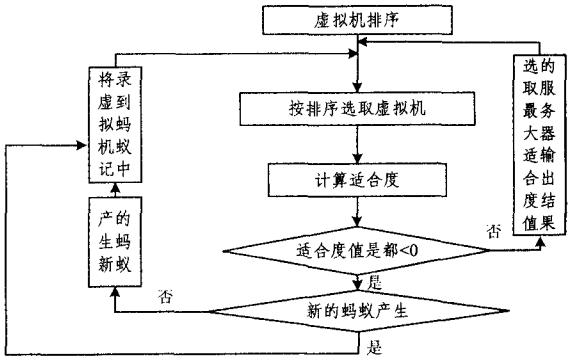

图 5 匹配算法流程图

2.6 框架的适应力确定

蚂蚁开始新一轮的搜索前，要进行初始化。为了提高搜

索的灵活性与适应性，蚂蚁的步长依据系统的负载状态进行动态设定。若搜索到的服务器信息中，过载的服务器个数较少，应该增加蚂蚁的步长来降低搜索的频率，而若过载服务器的个数较多，则要相应减短蚂蚁的步长。

由于模糊逻辑在处理具有多个参数的目标问题时具有很大的灵活性，因此本文使用此方法来确定蚂蚁新一轮的搜索步数 *NextM*，将蚂蚁步数作为一个语言变量、*NextM* 的确定要依据两个因素：1. 上一轮蚂蚁搜索到的过载服务器的个数 *OLC*；2. 上一轮蚂蚁的步长 *LastM*。*NextM* 的确定公式如下：

$$R:OLC\langle L,M,H\rangle * LastM\langle TL,L,M,H,TH\rangle \rightarrow NextM\langle TL,L,M,H,TH,Dead\rangle$$

设计模糊专家系统来确定蚂蚁的适应能力，其具体实现主要包括 4 个步骤：输入变量的模糊化、规则构造、聚合规则的输出以及最终的逆模糊化。其具体的结构如图 6 所示。

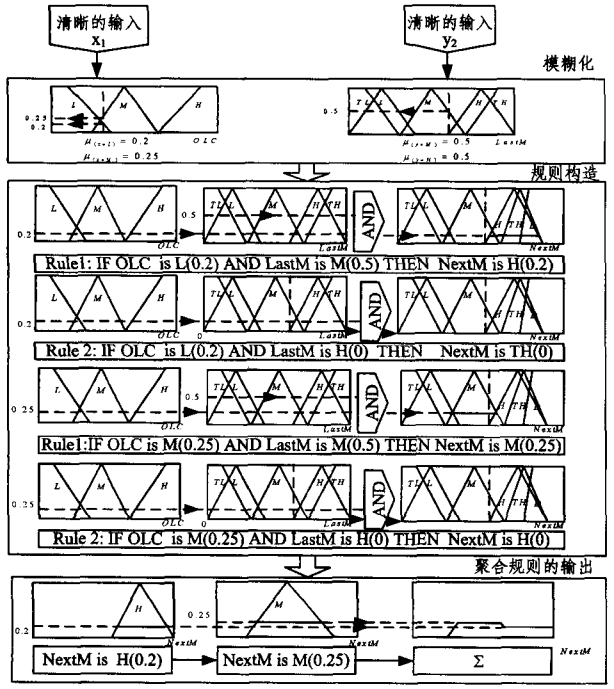

图 6　模糊专家系统的基本结构图

3　仿真实验与分析

本文采用澳大利亚墨尔本大学的网格实验室和 Gridbus 项目提出的云仿真平台 CloudSim[12] 作为试验仿真工具，使用的版本为 CloudSim-2.1.1。

为了实现仿真，首先要实例化出一个云计算数据中心。数据中心包含 30 台物理服务器，每台物理服务器选取 CPU、内存和网络带宽 3 个性能来描述，服务器的配置统一为 4G ram、10000 MIPS、1000 Mb/s，将服务器分为 3 个域的阈值设为 $M_1=0.45$，$M_2=0.75$，蚂蚁游荡的步数设为 $m=6$。在构建完成一个云计算数据中心之后，要提交虚拟机进行部署。因此，实例化 DatacenterBroker，提交一个具有 30 台虚拟机请求的虚拟机部署列表。与服务器相同，也选取 CPU、内存和网络带宽 3 个性能指标来描述，虚拟机的配置统一为 1G ram、2500 MIPS、250Mb/s。虚拟机在服务器上的部署如表 3 所列。

表 3　虚拟机部署表

服务器	虚拟机	L_{host}
1	1	0.5
	2	
2	3	0.5
	4	
3	5	0.5
	6	
4	7	0.5
	8	
5	9	0.5
	10	
6	11	0.5
	12	
7	13	0.5
	14	
8	15	0.5
	16	
9	17	0.5
	18	
10	19	0.5
	20	
11	21	0.25
12	22	0.25
13	23	0.25
14	24	0.25
15	25	0.25
16	26	0.25
17	27	0.25
18	28	0.25
19	29	0.25
20	30	0.25
其它	0	0

可以发现实验用的 30 台服务器中，10 台处于平衡区，20 台处于低负载区。为了实验需要，设定每台服务器每 10 秒有 5% 的概率增加部署一台相同配置的虚拟机。增加的虚拟机编号从 31 开始依次增加。

3.1　算法参数选定分析

表 4 为挥发系数 ρ 对算法结果的影响。在实验环境中进行 30 次负载均衡操作，改变 ρ 的值，分别实验 20 次取平均值，计算负载均衡因子，负载均衡因子定义如下：

$$L_{datacenter(k)}=\sqrt{\frac{\sum_{i=1}^{30}(\bar{L}_k-L_{ik})^2}{30}}$$

其中，L_{ik} 代表第 i 台服务器在第 k 秒（$k=0,10,20,\cdots,300$）时的负载均衡值。$\bar{L}_k$ 为数据中心在第 k 秒时的平均负载，定义如下：

$$\bar{L}_k=\frac{\sum_{i=1}^{30}L_{ik}}{30}$$

表 4　算法参数对结果的影响（迭代 30 次）

初始化参数值	负载均衡最优结果	负载均衡平均结果	负载均衡最差结果
$\rho=0.9$	0.1825	0.2179	0.2579
$\rho=0.8$	0.1345	0.2064	0.2358
$\rho=0.7$	0.0960	0.1646	0.2262
$\rho=0.6$	0.0786	0.1582	0.2041
$\rho=0.5$	0.0902	0.1728	0.2162
$\rho=0.4$	0.1265	0.1876	0.2386
$\rho=0.3$	0.1596	0.1936	0.2521
$\rho=0.2$	0.1865	0.2256	0.2698

不同的 ρ 值得到的负载均衡因子如表 4 所列，从表中可

以看出初始时随着 ρ 减小，负载均衡因子逐渐减小，但是超过 0.6 时，又不断增大，这是因为当 ρ 值太大或太小时，各节点信息素值间的差异不明显，降低了信息素在指导蚂蚁搜寻过程中所起的作用。

3.2 负载均衡比较分析

为了验证分布式迁移框架的有效性，将其与集中式迁移框架、未采用任何迁移措施的方案进行比较，其中集中式迁移框架通过增加一台服务器作为管理节点，其匹配策略与分布式迁移框架中的匹配策略完全相同，其配置为 8G ram、20000mips、2000bw。每隔 10 秒统计整个数据中心的平均负载，实验进行 30 次。仿真实验结果如图 7 所示。

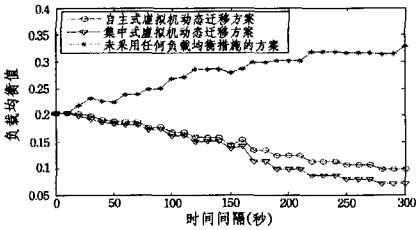

图 7 负载均衡效果比较

从图中可以看出，未采用任何负载均衡措施的方案，其负载均衡因子不断增大。而集中式虚拟机迁移方案和自主式虚拟机迁移方案的负载均衡因子则不断降低，其整个数据中心的负载越来越均衡，不会出现某些服务器因负载过高而影响服务质量，而另一些服务器则空闲的情况，不会造成资源浪费。另外从图中可以看出，自主式虚拟机迁移方案的负载均衡因子略低于集中式的虚拟机迁移方案，但是相差不大，因此本文提出的自主式虚拟机迁移方案适用于云环境下的负载均衡需求。

3.3 性能评估

本节选用负载均衡效率作为指标来为自主式迁移框架的性能进行评估。计算出整个数据中心的负载均衡因子以后，计算负载均衡效率，定义如下：

$$Eff_k=\frac{L_0-L_k}{T_k}$$

其中，L_0 代表数据中心初始的负载均衡值，而 T_k 代表达到负载均衡值 L_k 时所花费的时间。影响负载均衡效率的最重要的两个因素是：蚂蚁的内存空间 Mem 和步长 m。若 $Mem \ll m$，那么蚂蚁的内存空间被占用，从而产生大量新的蚂蚁，这样系统中的信息素会大量增加，而减慢系统的负载均衡速度，但是会降低整个系统的负载均衡值。而若 $Mem \gg m$，则会降低产生新蚂蚁的概率，这样会提高系统的负载均衡速度，但是也会增大整个系统的负载均衡值。基于以上分析，本文进行仿真实验分析，确定内存空间 Mem 和步长 m 的比例关系。如表 5 所列，设定蚂蚁的步长 m 为 15，实验时间为 60 秒，分析不同内存的蚂蚁对负载均衡值的影响。从表中可以看出内存值应设定在步长的一半附近。

表 5 不同内存值对负载均衡值的影响

初始化参数值	负载均衡最优结果	负载均衡平均结果	负载均衡最差结果
Mem=15	0.1936	0.2379	0.2879
Mem=10	0.1545	0.2264	0.2658
Mem=7	0.0860	0.1846	0.2362
Mem=5	0.0986	0.2082	0.2541
Mem=3	0.01856	0.2382	0.2641

接下来，本文选取 3 组不同的蚂蚁内存空间和步长值对负载均衡效率进行仿真分析，如图 8 所示。

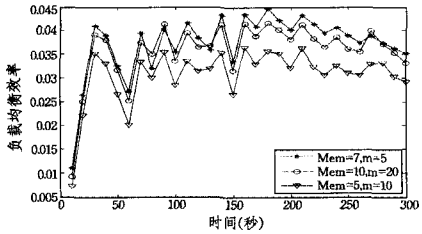

图 8 负载均衡效率

从图中可以看出，若蚂蚁的初始步数较长，如 $m=20$，则蚂蚁平衡单个服务器节点的时间会增加，即 T_k 会增大，因此降低了系统的负载均衡效率。而若蚂蚁的初始步数较短，如 $m=10$，则会在一定程度上减少平衡单个服务器节点的时间，但是由于搜索范围受限，导致负载均衡值 L_k 增大，因此也会降低系统的负载均衡。

结束语 借鉴蚁群算法的思想，提出了一种面向负载均衡的自主式虚拟机动态迁移框架，该框架不需要中央管理模块，能够实现服务器的自主迁移，避免了单点失效。利用智能蚂蚁的搜索，实现了自主式框架的迁移机制，而且使用模糊逻辑推理，根据系统的负载状况自动地调整智能蚂蚁的搜索半径，提高了搜索性能。针对自主式框架，制定了虚拟机的动态迁移策略：将服务器按照负载划为 4 个不同的负载状态域，并根据其所在的状态域进行迁移决策；根据监测数据对服务器未来的负载进行预测；选择待迁移虚拟机时综合考虑资源需求与迁移成本因素。最后在 CloudSim 平台上进行仿真实验，验证了框架的可行性，确定了合适的框架参数，并且通过仿真实验与比较分析，验证了本文提出的自主式虚拟机迁移框架具有良好的负载均衡能力。

参 考 文 献

[1] Above the clouds: a Berkeley view of cloud computing [EB/OL]. http://www.eec.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[2] 王庆波，何乐，赵阳. 虚拟化与云计算[M]. 北京：北京电子工业出版社，2010：115

[3] Aggarwal G, Motwani R, Zhu An. The load rebalancing problem [C]// SPAA03: Proceedings of the 15th annual ACM symposium on Parallel Algorithms and architectures. June 2003

[4] Nelson M, Lim B-H, Hutchins G. Fast Transparent Migration for Virtual Machines[C]// Proceedings of USENIX ATC. 2005

[5] Clark C, Fraser K, Hand S, et al. Live Migration of Virtual Machines[C]// Proceedings of the 2nd Conference on Symposium on Networked Systems Design and Implementation. 2005, 2: 273-286

[6] Travostino F, Daspit P, Gommans L, et al. Seamless live migration of virtual machines over the MAN/WAN [J]. Future Generation Computer System, 2006, 22(8): 901-907

[7] Bradford R, Kotsovinos E, Feldmann A, et al. Live wide-area migration of virtual machines including localpersistent state[C]// Proceedings of the 3rd International Conference on Virtual Execution Environments. San Diego, CA, 2007: 169-179

(下转第 102 页)

(2)口令验证

用户身份认证界面仍然由这一张风景图片作为背景。与口令设置界面不同,用户不能用手指点触界面,而是通过滚动位于屏幕中央的重力感应小球来实现口令的验证。用户通过摇晃所持设备,使小球按顺序通过在口令设置时所设点的有效区域。验证步骤如下:首先判断滚动球与第一个所选点有效区域的位置。当滚动球经过该有效区域时,第一步验证成功。继续判断滚动球与第二个所选点有效区域的位置,以此类推。例如当判断第二个有效区域时,滚动球经过第三个有效区域,则不处理,直到滚动球经过第二个有效区域则继续判断。若用户在间隔5秒钟内都没有滚至该步骤的有效区域,则认为验证失败,退出身份验证环节。当用户认为滚动球操作有误时,可将球滚至右上角的区域,执行重新验证操作,清空已操作的步骤,等待下次重新验证。验证口令效果如图9所示。

图9 重力感应认证方案验证口令

(3)密钥空间

该方案中背景为800×800像素的图片,而有效区域为半径40个像素的圆。

选择1个特殊点时,密钥空间约为$[800^2/(40^2)]=400$;

选择2个特殊点时,密钥空间约为$[800^2/(40^2)]\times([800^2/(40^2)]-1)=159600$;

选择3个特殊点时,密钥空间约为$[800^2/(40^2)]\times([800^2/(40^2)]-1)\times([800^2/(40^2)]-2)=63520800$。

对于背景为$M\times N$像素的图片,有区域半径为R,设置n个点作为口令时的密钥空间计算公式为:$[M\times N/R^2]!/([M\times N/R^2]-n)!$。此公式计算的密钥空间为估算值,实际的密钥空间将比公式计算的略大。

结束语 本文所提出的身份认证方案在设计时都摒弃了文本形式的密码认证模式,通过图像选择、绘制曲线、多点指划、重力感应等组合形式完成密码口令的设置和认证,方案具有一定的新颖性和吸引力。并且在认证过程中密码口令不会以明文形式出现,从而增强了认证方案的安全性。我们针对Android平台的智能手机和平板设备,利用Java语言和Eclipse编程软件进行了编程实现,下一步将扩展至其他主流的移动平台操作系统,以满足不同用户的需要。

参考文献

[1] Hu Wei, Wu Xiao-ping, Wei Guo-heng. The Security Analysis of Graphical Passwords [C]// 2010 International Conference on Communications and Intelligence Information Security. ICCIIS2010. 2010:200-203

[2] 胡卫,吴晓平,张昌宏,等. 图形密码身份认证方案设计及其安全性分析[J]. 计算机工程与设计,2009,30(14):3284-3287

[3] Suo X, Zhu Y, Owen G S. Graphical passwords: A survey[C]// 21st Annual Computer Security Applications Conference (ACSAC'05). 2005:463-472

[4] Chiasson S, Biddle R, van Oorschot P C. A Second Look at the Usability of Click-Based Graphical Passwords[C]// Symposium On Usable Privacy and Security (SOUPS) 2007. Pittsburgh, PA, USA, 2007:18-20

[5] Birget J C, Hong D, Memon N. Graphical Passwords Based on Robust Discretization [J]. IEEE Transactions on Information Forensics and Security, 2006, 1(3):395-399

[6] 汪永松. Android平台开发之旅[M]. 北京:机械工业出版社,2010

[7] 丰华,于松波. Eclipse开发技术详解[M]. 北京:中国铁道出版社,2010

(上接第85页)

[8] Khanna G, Beaty K, Kar G, et al. Application Performance Management in Virtualized Server Environments [C]// Proc. of Network Operations and Management Symp. 2006

[9] Wood T. Black-box and Gray-box Strategies for Virtual Machines Migration[C]// Proceedings of the 4th Int'l Conference on Networked Systems Design & Implementation. IEEE Press, 2007

[10] Bobroff N, Kochut A, Beaty K. Dynamic Placement of Virtual Machines for Managing SLA Violations[C]// Proc of the 10th IFIT/IEEE International Symp. on Integrated Network Management. 2007

[11] Montresor, et al. Messor: Load-Balancing through a Swarm of Autonomous Agents[C]// Proc. of 1st Int. Workshop on Agents and Peer-to-Peer Computing. Italy, 2002

[12] Calheiros R N, Ranjan R, De Rose C A F, et al. Cloudsim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services [R]. GRIDS-TR-2009-1. Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, March 2009