基于代理群的网络管理中群首选举算法的研究*)

李 航 赵志刚 王光兴
(东北大学信息科学与工程学院 沈阳110004)

摘 要 基于代理群的网络管理是一种动态的分布式管理模式,群首在代理群中是最关键的节点,群首的选举则是动态群管理中的最关键问题之一。针对选举问题,本文在建立的部分同步系统模型基础上,提出了一种三段式的群首选举算法,该算法具有较高的效率和一定的容错能力;同时应用故障检测器相关理论解决了选举的触发问题,并对相关参数的确定进行了讨论。

关键词 分布式网络管理,代理群,群首,选举,容错

Group Leader Election Algorithm for Agent Groups Based Distributed Network Management

LI Hang ZHAO Zhi-Gang WANG Guang-Xing
(School of Information Science and Engineering, Northeastern University, Shenyang 110004)

Abstract It is a dynamic distributed paradigm that network management is based on agent groups. In an agent group, group leader is the most important node and group leader election is pivotal problem of dynamic group management. To illustrate leader election, a partial synchronous system molel is presented for describing agent group. Based on the model, a 3-phase fault-tolerant leader election algorthm is put forward, which is efficient and tolerates crashed agents and lossy links. fault-tolerant trend. To meet election trigger, the heartbeat failure detector is implemented and several crucial parameters are discussed.

Keywords Distributed network management, Fault-tolerant, Agent groups, Group Leader, Leader election

针对大规模分布式网络中被管元素分散性、异构性和动态性的特点,一种基于代理群的分布式网络管理机制被提出。代理群的概念认为:从网络管理角度来看,被管网络由基本的被管元素——代理组成。许多代理依据某种原因集合成不同的群,由一个特殊的代理——群首对群内代理进行管理。群首和代理之间依据集中式管理模式[1]协作,即群首指示代理进行特定的操作或提供特定的信息,代理返回操作结果或被要求的信息;群首和群首之间则依据一定的分布式协同模式[2]完成管理任务。

显然,在代理群中群首是进行组群和协同管理最关键的节点,正因为群首的重要性,一旦群首出现故障,将导致整个群管理功能的失效。因此,基于代理群的管理采用了一种基于动态冗余的重构容错机制[3],即在代理群中设置多个具备群首部分或全部功能的代理实体作为冗余,如果群首不能正常工作,通过选举产生群中性能最优冗余代理替代原群首,并重构代理群。

本文首先对群首选举问题进行描述并建立相应分布式系统模型;然后重点对群首选举算法进行说明和分析。

1 问题描述

在一个代理群中,群首不仅要管理所属群成员,获得代理的运行状况,还要将收集、计算和处理的监控信息结果及时通告给其它群首;群首的生存关系着其它成员的组织变化,群首的状态决定组群的策略。因此,有必要在群中设置一些高性能代理,使其具有群首的功能,可以在群首失效情况下,部分或完全替代群首工作,这里称之为分布式代理,选举就是在分布式代理之间进行。

正常工作时,群首每隔一定时间 $T_{Collect}$ 向群内所有分布式代理收集其负载信息,并形成负载优先级列表,在优先级列表里记载了当前群中各个分布式代理负载优先级别。每隔周期 T_{LstPRI} 群首向群内每个分布式代理发送该负载优先级列表。如果某个分布式代理在时间 T_W 内没有收到优先级列表,认为群首失效,启动选举过程,最终选出负载性能最好的分布式代理来替代原群首完成管理工作。

群首选举要解决以下几个问题:

(1)T_W 的确定。一般来说,T_W 主要由 T_{LstPRI} 决定。T_{LstPRI} 时间设置得越短,选举者对群中代理的信息了解得就越及时和准确,但考虑到管理信息和实际应用共用同一个网络,当周期很短时,网管信息可能导致网络资源占用过高,从而严重影响到网络的性能和功能。但是如果周期很长,则可能造成管理信息过时,不能反映代理群真实情况。另外一个决定 T_W 的因素是网络的传输状况,消息的延迟和丢失导致了 T_W 的不确定性,因此需要引入一些相对确定的参数来定量决定 T_W。

(2)选举算法容错性。因为在群首失效时,不能保证其它分布式代理和链路不发生故障,所以提出的选举算法必须具备很好的容错性,同时还要兼顾算法的复杂度。

(3)决定负载优先级的负载指标的确定。

(4)代理实体间可靠、有序的组播通信能力。

本文重点关注前两个问题,后两个问题将另文讨论。下面给出解决问题的系统模型。

2 系统模型

考虑一个代理群内有 n 个分布式代理($n\geq 2$),其中群首被看作是一个特殊的分布式代理,这些代理构成一个分布式

*)本文得到国家863课题资助(No. 2003AA712032)。李 航 博士研究生。王光兴 博士生导师。

系统。每个分布式代理都具有唯一标识，并作为先验知识被其它代理所知，因此分布式代理的集合可以用 $\Pi=\{0,1,2,\cdots,n-1\}$ 表示；代理实体间的消息传递通过一组链路 Λ，任意两个分布式代理间都可以通过消息传递相互通信 $\Lambda=\Pi\times\Pi$，即群内分布式代理形成的网络为全连接拓扑；从代理到代理的链路记为 $p\rightarrow q(p,q\in\Pi)$；系统是部分同步的[4]，消息从 p 到 q 的时延是 δ（但无明确上限），代理处理消息的时间相对于 δ 可以忽略不计；代理间不存在同步的时钟，但本地时间和实际时间无时钟漂移。

分布式代理一旦发生故障即停止发送消息，而且故障为永久故障，不可恢复。一个链路 $p\rightarrow q$ 满足以下条件：(1) p 在时刻 $t-\delta$ 传递消息 m 给 q，如果链路 $p\rightarrow q$ 在 $[t-\delta,t]$ 内链路无故障，那么 q 最终能接到 p 传递的消息 m。(2)没有超时传输，如果 p 时刻 $t-\delta$ 传递消息 m 给 q，则 q 在时刻 t 之后不会收到信息 m。

群首每隔周期 $T_{LastPRI}$ 发送的负载优先级列表中，优先级别最高的记为 Max(ListPRI)，对于出现故障的代理优先级设为空 Φ。优先级列表的时间戳(timestamp)用自动递增的正整数表示。p.ListPRI 表示代理 p 存储的优先级列表，p.timestamp 表示代理 p 存储优先级列表的时间戳，p.timestamp=0表示代理 p 无优先级列表；p.ListPRI[k]代表代理 p 存储优先级列表中标识为代理 k 的优先级别。

每个分布式代理还要维护一个故障检测模块来检测群首是否失效。因为各个代理维护的故障检测模块完全相同，将群首失效检测简化为只有群首 *leader* 和一个代理 p 的模型 (*leader*, $p\in\Pi$)。又因为群首和代理间的链路处于实际网络环境中，所以认为其链路状态(主要是消息丢失和延迟)呈现一定概率特性，其中链路消息丢失概率记为 P_L，链路消息延迟的均值和方差记为 $E[\delta]$ 和 $Var[\delta]$。

根据代理 p 的时钟，群首 *leader* 发送的每个优先级列表的时间点记为 $\sigma_1,\sigma_2,\sigma_3,\cdots$；$p$ 接受每个优先级列表时间点记为 $A_1,A_2,A_3,\cdots$；模型还引入了被称为刷新点 $\tau_1,\tau_2,\tau_3\cdots$ 的时间点。*leader* 发送的每个优先级列表记为 $m_1,m_2,m_3,\cdots$。如图1所示，若存在 $j\geqslant i$，如果 $A_j<\tau_i$，p 认为 *leader* 正常(图1a、b)；$\tau_i\leqslant A_i\leqslant\tau_{i+1}$，$p$ 怀疑 *leader* 失效，但未认定(图1c)；$\tau_{i+1}<A_i$，p 认为 *leader* 失效(图1d)。

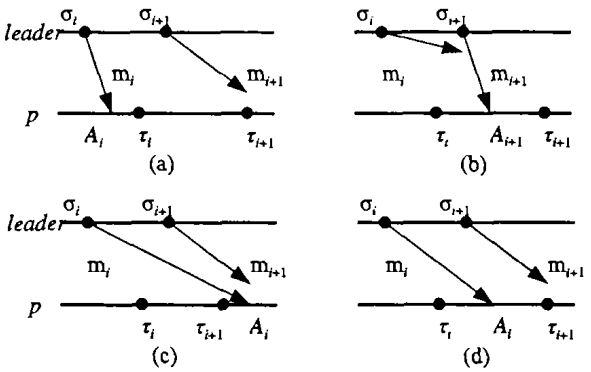

图1 代理对群首状态的判断

因此，时间 T_W 可以定义为从收到上个优先级列表到下一个刷新点这段时间，可以用 p 时间轴上 $[A_i,\tau_{i+1}]$ 表示，问题转化为确定 τ_{i+1}。参考 W. Chen 在提出的无同步时钟系统中有 QoS 保证的 Heartbeat 策略[5]，有 $\tau_i=\sigma_i+E[\delta]+T_D-T_{LastPRI}$，其中 T_D 是从群首实际失效到被代理 p 检测到的时间，称之为群首故障检测时间。参数 T_D 的大小决定了群首选举的效果，一般根据管理需要特别指定。

这里要注意的是，由于链路状态的不确定性，p 对 *leader* 的失效检测不可避免地会产生误报，为了降低误报的影响，引入了误报间隔时间 T_{MR}，其均值记为 $E[TMR]$。有公式(1)存在：

$$\begin{cases}\alpha=T_D-T_{LastPRI}\\ k=\dfrac{\alpha}{T_{LastPRI}}-1\\ \beta=\prod_{j=0}^{k}\dfrac{Var(\delta)+P_L(\alpha-jT_{LastPRI})^2}{Var(\delta)+(\alpha-jT_{LastPRI})^2}\\ E[T_{MR}]=\dfrac{T_{LastPRI}}{\beta}\end{cases}\tag{1}$$

这样，根据参数 T_D、$E[T_{MR}]$、P_L、$E[\delta]$ 和 $Var[\delta]$ 就可以定量地确定 T_W 和 $T_{LastPRI}$。

3 选举算法

3.1 算法提出

分布式系统中的选举算法大致可以分为两类：一类是工作在同步模型下的选举算法，如同步环中基于比较的 LCR 算法，非基于比较的时间片算法和变速算法[6]，全连接拓扑的占领算法等[7]；一类是工作在异步模型下的选举算法，如 GHS 算法[8]、邀请算法[9]等。同步模型下的算法因为严格遵循锁-步机制，所以算法效率较高，但其算法的正确性严重依赖时序假设。而异步模型是符合大多数实际情况的，但在完全异步的模型中，存在太多的未知特性，导致无法解决具体问题。因此本文算法采用部分同步模型，加入时间特性。在部分同步模型中，Ω 选举算法[10]和 Ω^+ 选举算法[11]讨论了利用故障检测器[12]保证算法的容错性的思想，因此在算法中引入故障检测器以保证算法的容错性是可行的。另外，前人的选举算法中很少讨论选举如何触发，本文则在 NFD-U 算法[5]和 Ω^+ 选举算法的基础上，提出了一种三段式的选举算法。该算法能够适应节点故障和链路故障的状况，并解决了选举的触发问题。

3.2 算法说明

算法共分三个阶段：第一阶段，利用 NFD-U 算法的变体检测群首失效状况，触发选举；第二阶段，利用回合制和对分布式代理的先验知识，选举产生整个系统中唯一的决策者(*judge*)；第三阶段，利用第二阶段选举产生的决策者和信任集，决策者收集信任集中所有代理的优先级列表，并更新决策者的优先级列表，从而选出优先级最高的分布式代理为群首(*leader*)。算法形式化描述如图2。

3.3 算法分析

在第一阶段，算法的运行时间和代理群内参加选举的分布式代理数量没有关系，只由群首故障检测时间 T_D 决定。一般来讲，$T_D\gg\delta$，如果假设网络链路延迟 δ 满足指数分布，$P_L=0.01$，$E[\delta]=0.02$时，T_D 和误报间隔均值 $E[T_{MR}]$ 关系如图3所示。

算法的第二和第三阶段，其运行时间和消息复杂度则与分布式代理数量 n 有着直接的关系。第二阶段，利用代理标识这一先验知识，算法保证了在群首失效时可以很快地选举出一个系统中连通性最好的决策者，这阶段选举能够容忍代理故障和链路故障。算法的最大运行时间，在系统无故障时，最大运行时间为 3δ，系统出现故障时，其最大时间为 $(3n+2)\delta$。算法的第三阶段在保证了所选举出的群首属于连通性最好的

集合之外，还保证了群首是当前负载最轻、性能最好的代理。其最大运行时间为3δ。算法第二和第三阶段的消息复杂度为$O(n+n*f)$。在系统有m个正常工作的代理时，所需的最大消息数为$2n+m+2(n-1)$，当出现f条链路故障时，最大消息数为$(n-1)+(3n+m-1)*f$。

```
{ phase 1  对于群内的每个分布式代理 p∈ Π 有：}
1    on initialization:
2      τ0 = 0
3      l = 0
4     if p = leader then
5       for all i ≥ 1 at time i* T_LinPRI do
6        send (MESSAGE , p.ListPRI, p.timestamp) to all in  Π \{leader}
7     else
8       upon τ_{i+1} = the current time
9        leader is wrong
10      upon receive(MESSAGE , *, j) at time t
11       if  j > l  then
12        l← j
13        τ_{l+1} ← σ_l + E[δ] +T_D−T_LinPRI
14         if t<τ_{l+1} then
15           leader still is OK
{ phase 2  对于群内的每个分布式代理 p∈ Π 有：}
16   Procedure StartRound(s)
17     if p≠ s mod n then send (START , s ) to all
18    r← s
19    judge← s mod n
20    trust ←    Π
21     restart timer
22    on initialization:
23      StartRound(0)
24      start tasks 0 and 1
25   task 0:
26     loop forever
27      if p = r mod n and have not send(OK ,r,*) within δ then
28       if p has been in round r for at least 2δ time then
29         for each q∈ trust s.t. p did not receive(ACK, r)
                 from q in the last 2δ time do trust ←trust \q
30        send (OK ,r ,trust) to all
31   task 1:
32     upon receive (OK, k, tr) with k = r do
33      if p∉ tr then StartRound(r+1)
34       else
35          trust ←tr
36          send (ACK, r) to r mod n
37          restart timer
38     upon timer >2δ do
39         StartRound(r+1)
40     upon receive(OK, k, tr) or (START, k) with k > r do
41       if received(OK, *, *) then send(ACK, k ) to k mod n
42          StartRound(k)
43     upon receive (OK, k, tr)or (START ,k) from q with k < r do
44       send(START, r) to q
{ phase 3  对于群内的每个分布式代理 p∈ trust 有：}
45   on initialization:
46     for each i∈ Π \ trust do ListPRI[i]← Φ
47     start tasks 0 and 1
48     restart timer
49   task 0:
50     loop forever
51      if p=judge and have not send(COLLECT , *) then
52       send(COLLECT, judge) to all in trust
53       if judge .ListPRI =p.ListPRI   for at lest timer>2δ time then
54        if judge .ListPRI≠Φ then
55          leader←Max (judge .ListPRI)
56          p.timestamp=0
57          send(LEADER,leader) to all in trust
58   task 1:
59     upon receive(REPORT, k, *, *)do
60       if k.timestamp > p.timestamp   then
61        send(ACK,p.ListPRI, p.timestamp) to judge
62        p .ListPRI← k.ListPRI
63        judge .ListPRI ← p.ListPRI
64     upon receive((LEADER,k)
65       leader← k
66       p.timestamp=0
67     upon receive(COLLECT, judge) do
68       if p≠judge then send(REPORT, p ,p.ListPRI, p.timestamp) to judge
```

图2　容错的三段式群首选举算法形式化描述

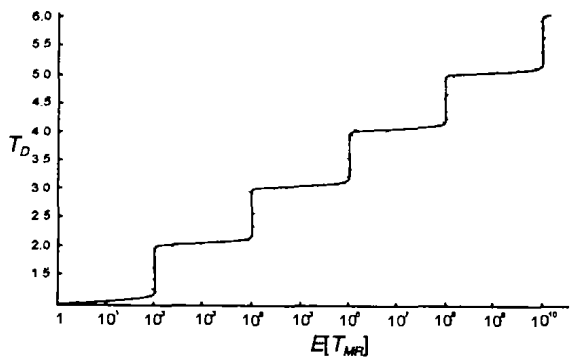

图3　T_D和误报间隔均值$E[T_{MR}]$的关系

结论　本文针对基于代理群的分布式网络管理中的选举问题建立了系统模型。并在解决了选举过程中选举触发问题的基础上，提出了一种容错的群首选举算法。分析表明该算法的时间效率较高，消息复杂度只有n的数量级。

下一步要针对选举算法中涉及到的决定优先级别的负载指标做进一步探讨；同时，如何保证选举过程中代理实体间可靠的、有序的组播通信也将是重点的研究方面。

参 考 文 献

1　Martin-Flatin J P, Znaty S, Hubaux J P. A Survey of Distributed Enterprise Network and Systems Management[J]. Journal of Network and Systems Management, 1999, 7(1): 9～26

2　Al-Shaer E. A Dynamic Group Management Framework for Large-Scale Distributed Event Monitoring[C]. In: Proc. of the 7th IFIP/IEEE Intl. Symposium on Integrated Network Management (IM'01), Seattle WA, 2001. 565～578

3　冯永新，赵林亮，王光兴. 一种基于移动网络的网络重组与恢复策略[J]. 《东北大学学报》自然科学版，2001，22(4)：397～400

4　Chandra T D, Toueg S. Unreliable failure detectors for reliable distributed systems[J]. Journal of the ACM, 1996, 43(2): 225～267

5　Chen W, Toueg S, Aguilera M K. On the quality of service of failure detectors[J]. IEEE Trans. Commun., 2002, 51: 566～582

（下转第70页）

(汇位置)都是一个位置并存在一个 in-set(out-set)集中;

3) 若不存在 in-set(out-set)集,则 N 的源结点是一个非空的变迁集合,且是 t 的映射的子集。

此时,N 为 t^+的细化,g 为细化函数。

对模型进行等价替换后,还应该保持模型的有效性(soundness),即满足基本要求[5]:(1) 一个用户任务只有一个源位置 I(起始状态)和一个汇位置 O(结束状态)。(2) 每个变迁/位置在一条从 I 到 O 的路径上。

定理1 使用等价网络方法进行替换后的信息系统模型满足有效性要求。

证明:可以用数学归纳法进行证明。

(1)用户任务的统一表示满足有效性要求。

(2)假设进行 n 次替换后得到的 Petri 网模型依然满足有效性要求。

(3)进行第 $n+1$次替换。由网络细化的第2条性质,可得出细化后模型依然满足只有一个源位置和汇位置的要求。从各种调用关系的网络模型,可以得到满足有效性第2点要求的结论。

证明完毕。

通过上述方法,我们可以把用户任务从图2所示的统一表示,逐步细化为以组成任务的事务的 Petri 网构成的一个完整的信息系统模型。在此模型的基础上,可以对信息系统的行为进行进一步的分析和设计。

4 设计实例

上述松散耦合信息系统体系结构和设计方法,已经在教育部重点项目"全国普通高校招生网上录取系统"中得到了很好的应用。该系统以全国各省市招生办公室为自治节点,在教育部设立全局应用服务器及数据仓库服务器,向各用户提供全局服务[7]。系统的用户主要包括省市招办的工作人员,他们主要使用本省市的自治系统;院校招办工作人员,他们的招生业务是分省进行招生,所以主要是面向一个自治系统的局部应用;院校主管领导,他们要掌握本院校在全国各省市的录取情况,其应用是一个面向全局的应用;教育部领导要掌握全国招生录取进展,其应用是一个复杂的面向全局的查询;教育部业务部门进行招生情况分析,由于实时性要求不是很高,他们的应用主要集中在数据仓库上进行。

下面,我们详细论述招生院校领导查询本院校在全国各省市录取情况这一全局分布式应用的实现。

用户的功能要求是查询本校在全国各省市招生录取的进展情况,在系统中,通过以下方案实现。

1) 用户层。用户层采用浏览器方案,主要工作是设计 Web 服务器,内容主要是用户界面。

2) 全局应用服务层要完成的工作包括:将用户查询任务分解到事务、并行调用各自治系统的相应接口、接收各自治系统返回的查询结果并根据用户要求进行组合和统计,提交给 Web 服务器。全局应用层和自治系统层的数据交换通过 XML 格式进行。

3)自治系统层要增加全局查询接口,主要功能是将查询要求进行全局/局部数据转换,调用本地事务进行查询,将查询结果进行局部/全局数据转换并生成 XML 数据格式提交给全局应用层。

通过三层体系结构,我们可以向用户提交最终的查询结果。

结论 本文介绍了一种在现有自治信息系统基础上构造出具有全局应用的新的分布式信息系统——松散耦合分布式信息系统的方法,提出了松散耦合分布式信息系统的体系结构,并以 Petri 网为工具,建立了松散耦合分布式信息系统的模型,为分析系统的行为打下了坚实的理论基础。最后,介绍了基于松散耦合信息系统体系结构的一个实例——全国普通高校招生网上录取系统。实践表明,该体系结构和模型可以充分利用现有自治的信息系统资源,构造出灵活有效,能够提供全局应用的分布式信息系统。

参 考 文 献

1 Adam N R, Atluri V, et al. Modeling and analysis of workflow using Petri Nets. Journal of Intelligent Information Systems, 1998, 10:131～158

2 林闯. 随机 Petri 网和系统性能评价. 北京:清华大学出版社,2000

3 Lausen G. Modeling and analysis of the behavior of information systems. IEEE Transactions on Software Engineering, 1998, 14(11):1610～1620

4 Knorr K. Dynamic access control through Petri net workflows. In: Computer Security Applications, 2000. ACSAC'00. 16th Annual Conf. 2000. 159～167

5 van der Aalst W M P, ter hofstede A H M. Verification of workflow task structures: A Petri-net-based approach[A]. Information systems, 2000, 25(1):43～69

6 刘卫东,徐恪,等. 松散耦合的分布式信息系统. 清华大学学报, 2002, 42(1):40～43

7 刘卫东. 全国普通高校招生网上录取系统技术报告. 北京:清华大学, 2001

(上接第66页)

6 Lynch N A. Distributed Algorithms[M]. Motgan Kaufmann Publishers, Inc. 1996. 1～62

7 Coulouris G, Dollimore J, Kindberg T. Distributed Systems: Concepts and Design, Third Edition[M]. Pearson Education Limited, 2001. 419～462

8 Wu J. Distributed Systems Design[M]. CRC Press LLC, 1999. 76～81

9 Tel G. Introduction to Distributed Algorithms[M]. Cambridge University Press, 2000. 227～260

10 Chandra T D, Hadzilacos V, Toueg S. The Weakest Failure Detector for Solving Consensus[J]. Journal of the ACM, 1996, 43(4): 685～722

11 Aguilera M K, Delporte-Gallet C, Fauconnier H, et al. Stable Leader Election[C]. In: Proc. of the 15th Intl. Symposium on Distributed Algorithms (DISC'01), Berlin Heidelberg. 2001. 108～122

12 Aguilera M K, Chen W, Toueg S. Using the heartbeat failure detector for quiescent reliable communication and consensus in partitionable networks. Theoretical Computer Science, 1999, 220(1): 3～30