一种高效的网格资源监测算法 ACTC 的改进算法

陈君臣[1], 王成良[1], 柳玉炯[1], 杨会友[2]
(1. 重庆大学 计算机学院,重庆 400044; 2. 重庆市垫江县教育委员会 电教信息中心, 重庆 408300)

摘 要: 资源监控是网格计算的关键组成部分。资源监控算法既要保证网格系统中资源信息高保真度又要考虑资源信息在网络传输中的负担最低。ACTC 算法就是为实现以上目的而被提出,但是算法中对资源信息更新量的动态门槛值(*d*_threshold)计算方式不合理,有可能导致算法演变成低级的单纯依靠时间敏感机制(TSM)来发送资源更新通知,并且算法中遗漏了一些极端情况的考虑。提出了新的 *d*_threshold 计算公式和限制条件,进行性能改进和完善,并对改进前后两算法进行实验对比。实验表明,改进后算法提高了资源信息准确性,并使监控系统负担减轻。

关键词: 网格资源监控; ACTC; 网格; 算法
中图分类号: TP309 **文献标志码**: A **文章编号**: 1001-3695(2010)01-0114-03
doi:10.3969/j.issn.1001-3695.2010.01.034

One high efficiency grid resources monitoring algorithm improved on ACTC

CHEN Jun-chen[1], WANG Cheng-liang[1], LIU Yu-jiong[1], YANG Hui-you[2]
(1. *College of Computer Science & Engineering, Chongqing University, Chongqing* 400044, *China*; 2. *Education & Information Center of Dianjiang County Education Commission, Chongqing* 408300, *China*)

Abstract: Grid resources monitoring is a key component of grid. Resources monitoring algorithm not only need ensure the high fidelity of the resources information, but also take account into the lowest burden in the network transmission. This paper proposed ACTC algorithm based on achieving this aim, however, the algorithm used a improper formula of calculating dynamic threshold amount of change for resource status(*d*_threshold), probability results to evolve into a inefficient algorithm sends resource update information based only on TSM, and ACTC don't take into account some special situation maybe happen in experiment. This paper put forward a new formula for calculating the value of *d*_threshold and add other limiting condition to perfect and improve the performance of ACTC algorithm, and comparing the original algorithm with the improved algorithm in the experiment. The experiment improves algorithm better on accuracy of resources information monitoring and alleviating the burden of monitoring system than former ACTC algorithm.

Key words: grid resources monitoring; ACTC; grid; algorithm

网格技术是一种在异步网络实现资源共享的解决方案。它能够把分散在不同网络的计算能力、任务处理能力、数据访问能力等组合在一起,使任务处理和数据访问效率提高,减少数据收集时间。网格资源是一种分布在广域网中的资源,通过某种网格服务把资源合并在一起去实现某种应用。这样就需要对资源信息进行有效管理,这就是人们常说的资源监控机制[1~3],要求在查找跟踪资源信息时资源保真率高并且减少在资源状态信息的更新和维持上的开支,而怎么达到减少开支与资源信息数据准确性之间的平衡是资源监控机制要考虑的问题。ACTC[1]算法是基于上推模型的 offset-sensitive mechanism (OSM)[1]和 time-sensitive mechanism(TSM)[1]的混合机制。在更新时间和更新信息量上都作了考虑,但是算法中忽略了一些特殊情况,如有可能出现资源信息更新量的动态门槛值 *d*_threshold 比最小资源信息更新量小的情况,同时对 *d*_threshold 的计算方式并不是很合理,查看资源更新量的动态时间在算法中有变小的趋势,会导致算法变成 TSM 机制。本文主要是在原 ACTC[1]算法的基础上提出了新的 *d*_threshold 计算公式,添加条件限制保证动态周期时间变化波动均匀合理,使原算法性能得到改进和更加完善,并对两者性能做了实验对比。实验表明,改进后的算法动态周期时间更加均匀,*d*_threshold 的值更能随当前资源更新量来调节。在减少不必要的资源更新信息的同时资源更新精度和更新时间更加精确合理,实现了资源信息更新准确性和系统负担的更好的平衡。

1 ACTC 算法原理

1.1 ACTC 算法背景

ACTC 算法是基于 GRIM 监控原型的,由三层组成(图 1),

收稿日期: 2009-04-15; **修回日期**: 2009-05-25

作者简介:陈君臣(1986-),男,硕士研究生,主要研究方向为网格计算、Linux 系统设计、嵌入式(junchen.mail@qq.com);王成良(1964-),男,教授,博士,主要研究方向为网格计算、Linux 系统设计、图形图像及多媒体技术、企业信息化、数据库、软件工程、自动控制;柳玉炯(1983-),男,硕士研究生,主要研究方向为网格计算、神经网络;杨会友(1971-),男,主要研究方向为网格计算.

即查询层、中间调节层、信息提供层。查询层负责各种类型用户查询自己感兴趣的网格资源信息;中间调节层是用来调节用户的查询和来自主机的资源信息更新;信息提供层由被监测的主机组成,任何一个提供网格服务的资源都可以看成一个主机,主机不断探测资源信息,并不断地把信息分发到中间层。

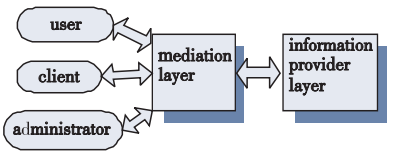

图1　GRIM原型

中间调节层与主机之间的交流又可以分成为下拉、上推、混合模型。下拉模型是由中间层的调节器来被动地请求主机发来的资源信息,主机收到请求后把资源信息发送到中间层。上推模型则是资源信息有更新时,主动地把更新信息发送到中间层,其主要面对的问题是上推更新信息的时间间隔值的合适取值。混合模型既可以被动地由中间层来提出请求资源状态的信息,同时定时上推资源更新信息。

ACTC 算法是基于上推模型的资源更新信息量敏感机制(OSM)[1]和时间敏感机制(TSM)的混合机制。OSM 机制分两种,即资源改变立即通知策略(announce-on-change scheme, AOC)和资源改变达到下限立即通知策略(announce-absolute-change scheme, AAC)。TSM 又分固定时间间隔通知策略(announce-regular-interval, ARI)和动态时间间隔通知策略(announce-dynamic-interval, ADI)两种。为了减少无用的更新和带宽的浪费,同时也考虑了时间间隔和更新的资源信息量,并设置了资源信息更新的门槛值等方法。资源信息改变量大于动态资源改变量下限(*d*_threshold 的计算公式为 $d\_threshold = (1/NA) \times \sum_{i=1}^{NA} AVC_i$),马上更新这个资源改变信息到中间层,同时,如果动态期满时间(timer 为当资源信息更新到中间层由动态时间间隔 DTI 赋值,反映资源改变的时间间隔,计算公式为 $DTI = \lfloor (1/NC) \times (T_{NC} - T_0) \rfloor$)到,并且资源信息量更新量大于定义的资源更新信息量最小值(min_threshold),更新资源改变信息到中间层。

1.2　ACTC 算法架构

```
timer = 无穷,NC = 0;
while(TRUE)
  A_y = 最后一次更新时资源的状态值;
  C_x = 目前的资源状态值;
  if(期满即 Timer 用完)
    if(C_x - A_y > min_threshold)
      发送资源更新信息到中间层;
      A_{y+1} = C_x;
    计算新的 d_threshold = (1/NA) × Σ_{i=1}^{NA} AVC_i;
    end if
    timer = DTI;
  end if
  if(资源状态信息值改变)
  if(NC = 0)
    发送资源更新信息到中间层;
      A_1 = C_1;NC = 1;
     DTI = ⌊T_1 - T_0⌋
      重置 timer = DTI;
    计算新的 d_threshold = (1/NA) × Σ_{i=1}^{NA} AVC_i;
  else
    NC = NC + 1;
     DTI = ⌊(1/NC) × (T_NC - T_0)⌋;
     if(C_x - A_y > d_threshold)
        更新资源信息到中间层;
        A_{y+1} = C_x;
      计算新的 d_threshold = (1/NA) × Σ_{i=1}^{NA} AVC_i;
      timer = DTI;
      end if
    end if
  end if
end while
```

2　ACTC 改进算法

2.1　ACTC 算法不足

a)从算法中 *d*_threshold 的计算公式 $d\_threshold = (1/NA) \times \sum_{i=1}^{NA} AVC_i$ 可见算法对第一次的 *d*_threshold 特别依赖,并且没有限制 *d*_threshold 的大小,导致可能会有 *d*_threshold 小于 min_threshold的错误现象,如表 1 所示。

b)多次发送资源更新信息到中间层后,由于 *NA* 已经很大,*d*_threshold 的计算公式 $d\_threshold = (1/NA) \times \sum_{i=1}^{NA} AVC_i$ 不能很好地随着实际中 AVC_i 的变化情况而调整 *d*_threshold 的值(表 2);同时,当期满时间比较长,可能出现 *d*_threshold 逐渐递增的现象。

c)当信息发生改变,*NC* 的值就增加 1,会逐渐使 DTI 变小,这样算法大部分时间在执行期满的处理,演化成了 TSM[1]机制,如表 2 所示。

2.2　对 ACTC 的改进方法

a)在资源发生更新时,增加限制条件 $C_x - A_y > \text{min\_threshold}$,条件成立才可增加 *NC* 的值和判断是否可以更新。这样就避免了上面提到 a)和 c)的不足,使更新时间更加均匀。

b)更改了 *d*_threshold 的计算公式为 $d\_threshold = \frac{AVC_i + \text{min\_threshold}}{2}$(期满中计算 *d*_threshold 的值)和 $d\_threshold = \frac{d\_threshold + AVC_i}{2}$(更新量大于 *d*_threshold 的值时)。这样就解决了上面提到的 b)的不足,并且使 *d*_threshold 更能反映目前的资源更新信息量。

2.3　ACTC 改进算法架构

```
timer = 无穷;NC = 0;
while(TRUE)
  A_y = 最后一次更新时资源的状态值;
  C_x = 目前资源的状态值;
  if(期满即 timer 值用完)
    if(C_x - A_y > min_threshold)
      发送资源更新信息到中间层;
      A_{y+1} = C_x;NA + +;
      d_threshold = (AVC_i + min_threshold)/2;
      重置 timer = DTI;
    end if
  end if
```

```
if(资源状态信息值改变)
  if(Cx - Ay > min_threshold)
    if(NC = 0)
      发送资源更新信息到中间层;
      A1 = C1;NC = 1;DTI = ⌊T1 - T0⌋;
      重置 timer = DTI;
      计算新的 d_threshold = (1/NA) × Σ(i=1..NA) AVCi 值;
    else
      NC = NC + 1; DTI = ⌊(1/NC) × (TNC - T0)⌋;
    if(Cx - Ay > d_threshold)
    更新资源更新信息到中间层;
    Ay+1 = Cx;
    计算 d_threshold = (d_threshhold + AVCi)/2;
      计算 DTI = ⌊(1/NC) × (TNC - T0)⌋;
      timer = DTI;
    end if
    end if
  end if
end if
end while
```

3　两个算法实验结果比较

为了证明改进后算法比原算法在动态资源更新门槛值 *d*_threshold 计算公式更加合理,进而在保持高资源信息保真率的同时比原算法占用更少网络带宽,同时,验证改进后的算法弥补了原算法遗留的漏洞。故做以下两个实验,让原算法和改进后算法同时运行在同一台 Linux 客户端,并把资源更新信息发送到 Linux 服务器。由服务器记录下两个算法的资源信息更新情况。

以下实验运行环境是:Redhat Linux 9.0,测试的资源为可用的内存空间大小。表 1 中数据是 ACTC 和改进算法在同一时间的运行结果,"/"前是 ACTC 算法数据,"/"后是改进算法得到的数据。min_threhold 为 10。表 1 中的数据由于篇幅所限,只列出了前几次的数据,这些远不能明显看出算法的优劣,但是从实验中大量数据分析,完全可以体现出改进后的算法在 *d*_threshold 调节的速度优于 ACTC 算法;同时,*d*_threshold 更能体现出当前更新信息量的变化情况,也避免了 *d*_threshold 小于 min_threshold 的错误情况。整个系统在更新时间和更新信息量上都更加合理,对资源信息保真率和网络开销更加平衡。

表 1　ACTC 中 *d*_threshold 小于 min_threshold 的错误特例

time	Sup_M/KB	Change_M/*d*_threshold/KB	KB	DTI/s	是否发送
17: 16: 04	2065372	0	0	0	
17: 16: 12	2065378	5	5/0	8/0	Y/N
17: 16: 19	2065371	6	5/11	7/15	Y/N
17: 16: 25	2065379	8	6/11	7/15	Y/N

表 1 中说明了 ACTC 算法出现的漏洞错误。*d*_threshold 的值小于 min_threshold 的情况下,即动态资源改变量的门槛值小于最小资源改变量的门槛值时,原算法不断地更新资源信息到中间层,如表 1 中的最后三条数据所示。但是这种更新信息是不合理的,因为它没有达到资源信息改变量的下限,而在改进算法中会得到更正。Sup_M 是内存的剩余量,Change_M 是内存的改变量。

从表 2 中可以看到,ACTC 算法中可能会使期满时间特别短,这样到后来程序不断地执行期满,如表 2 中的 DTI 可以看到原算法 DTI 的值由 8 s 变到 4 s,并且有继续变小的趋势。由于期满越来越小,算法不断执行期满内的处理,这样就演化成了 TSM[1] 机制,造成了不必要的更新,如表 2 中最后一次的更新;同时也可以从实验数据得到改进后算法的 *d*_threshold 的变化能反映出目前资源的改变量,同时可以使资源更新周期波动不大。从图 2 可以很好地看出改进后的算法在 *d*_threshold 的计算公式和资源更新在减少开支和资源信息数据的准确性的优势。

表 2　同一时间同一环境下 ACTC 和改进算法的运行实验结果

time	Sup_M/KB	Change_M/*d*_threshold/KB	KB	DTI/s	是否发送
17: 44: 13	2113760	0	0	0	
17: 44: 21	2113776	16	16/16	8/8	Y/Y
17: 44: 28	2113761	15		7/7	
17: 44: 29	2113761		6/13	7/7	Y/Y
17: 44: 35	2113791	30	20/21	7/7	Y/Y
17: 44: 41	2113823	32	23/26	7/7	Y/Y
17: 44: 45	2113814	9		6/7	
17: 44: 47	2113806	8		5/7	
17: 44: 48	2113806		22/18	5/7	Y/Y
17: 44: 51	2113815	9		5/7	
17: 44: 52	2113810	5		4/7	
17: 44: 53	2113810		20/14	4/7	Y/N

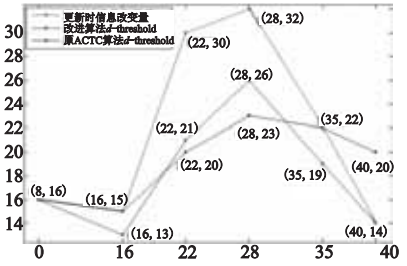

图2　变更时两种算法的*d*-threshold、资源改变量的曲线图

4　结束语

改进后的 ACTC 算法在实际实验中达到了减少开支和资源信息数据的准确性,使更新时间和更新信息量两者更加合理,同时也考虑到了 ACTC[1] 算法的不足之处。不过在 *d*_threshold 的计算公式上应该还有进一步完善的地方,以更好地运用到实际生活中去。

参考文献:

[1] CHUNG Wu-chun. A new mechanism for resource monitoring in grid computing[J]. Future Generation Computer Systems, 2009,25(1):1-7.

[2] KARL C, STEVEN F, IAN F, *et al*. Grid information services for distributed resource sharing[C]//Proc of the 10th IEEE International Symposium on High-Performance Distributed Computing. San Francisco,CA: IEEE Press, 2001:181-194.

[3] ADRIANA I, IAN F. On fully decentralized resource discovery in grid environments[C]//Proc of the 2nd International Workshop on Grid Computing. Berlin:Springer, 2001:51-62.

[4] MATTEW L M, BRENT N C, DAVID E C. The Ganglia distributed monitoring system: design, implementation, and experience[J]. Parallel Computing, 2004,30(7):817-840.

[5] SERASEIM Z, RIZOS S. A taxonomy of gridmonitoring systems[J]. Future Generation Computer Systems, 2005,21(1):163-188.