文章编号: 1674-7070(2012) 02-0163-07

一种对资源不稳定性敏感的 EASY-backfill 算法

王征宇[1]　肖南峰[1]

摘要

网格将分布式的计算节点连接起来,形成一个集中的计算和资源环境. 网格环境中的任务调度对于网格的运算效率和整体性能有很大的影响. EASY-backfill 算法作为经典的动态网格任务调度算法,有着算法简单、运算量小、调度性能优秀等诸多优点,但其算法条件对于计算资源的假设是理想绝对稳定的,同时认为任务的性能预测是精确可靠的,这显然不符合实际情况. 首先建立不稳定计算资源的模型,在该模型下改进 EASY-backfill 算法,使其能够在感知计算资源的不稳定性的条件下,保持算法原有的效果; 然后,把经典 EASY-backfill 算法与改进算法作了比较; 最后,就不稳定计算资源模型的相关参数对算法造成的影响进行了讨论.

关键词

网格计算; 任务调度; EASY-backfill 算法

中图分类号 TP391. 41

文献标志码 A

0　引言

网格计算(grid computing) 的概念 1995 年被提出[1] ,而关于网格以及网格计算本身还没有一个公认的定义. 根据 Foster 博士[2] 早期的定义,网格是一个集成的计算与资源环境,或者说是一个计算资源池,网格能够充分吸纳各种计算资源,并将它们转化成一种随处可得的、可靠的、标准的同时还具有经济的计算能力. 除了各种类型的计算机,这里的计算资源还包括网络通信力、数据资料、仪器设备等资源. 针对网格概念模糊的现象,Foster[3] 还提出了判断网格 3 条标准: 非集中式协同控制资源; 使用标准、开放、通用的协议和接口; 提供非平凡的服务质量. Foster 进一步指出,网格计算关心的是在动态的、多机构的虚拟组织中协调资源共享和协同解决问题.

网格计算的发展经历了 3 个阶段(图 1) ,目前是网格计算发展的第 3 阶段,也是迅速发展的阶段,有关网格的研究、开发和应用项目大量涌现.

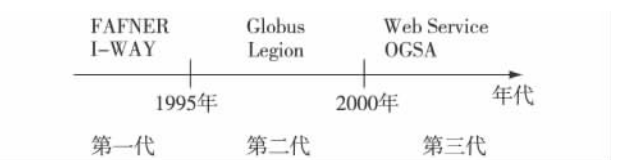

图 1　网格技术的发展阶段

Fig. 1　The development stages of grid technology

在网格计算环境中,资源是分散在各个不同地域和管理域中,由不同的组织拥有和操作,并且在使用策略和安全机制上各不相同. 网格资源包括计算资源、网络资源、存储资源等,本文所指资源,主要指计算资源,即网格结点的 CPU、内存等. 资源管理是网格计算的核心问题,它包括资源的组织、定位、发现、调度、分发等. 用户通过作业管理功能向网格系统提交作业,根据资源的可用性、性能、任务的完成时间要求、Qos 要求等,为作业分配相关计算资源,并安排运行日程和策略,这就是网格的任务调度.

网格环境下的任务调度问题主要解决的问题就是面向用户,协调网格资源和应用程序,以实现高效的资源 < -> 任务的映射,提高网格资源的使用效率,同时也提高应用任务的运行效率和可靠性. 因此,用于实现资源任务映射的网格任务调度策略就变得十分钟重要,

收稿日期 2010-10-25

资助项目 国家自然科学基金(61171141) ; 广东省自然科学基金重点项目(8251064101000005)

作者简介

王征宇,男,博士生,主要研究方向为网格计算与计算机应用. balladist@ gmail. com

肖南峰(通信作者) ,男,博士,教授,博士生导师,主要研究方向为智能计算与计算机应用. xiaonf@ scut. edu. cn

1　华南理工大学 计算机科学与工程学院,广州, 510006

研究并提出一个好的任务调度策略将会在很大程度上提高网格资源的利用率,推动网格技术的进一步发展.

1 网格任务调度算法

1.1 任务调度算法的介绍

NWS 服务[4](Network Weather Ser-vice,即网络天气预报服务)有非常典型的任务调度流程.

如图 2 所示,首先任务划分模块将应用程序划分成一个个子任务,任务分解的原则与调度排序算法的制定要根据应用程序的自身组织逻辑和时间顺序确定. 比如对于一个并行程序而言,在程序运行以前组织逻辑已经分解到程序的各个子进程当中. 把任务分解完毕之后调度模块首先从网格资源中的信息采集模块 MDS 收集必要信息,然后对任务按照一定的策略进行分发. NWS 主要用于获取当前网格环境中网络的一些重要信息,当主机节点安装这种服务后,就可以跟踪现有的网格环境中资源的使用状况,对网络状况进行分布式监控,并能提供短期的网络性能预测. NWS 将节点的相关信息提供给 MDS,从而为任务调度算法提供及时地节点相关数据.

简单地说,任务调度算法考虑的是在网格环境下一组相互独立、任务之间没有通讯和数据依赖的元任务(metatask)映射. 任务调度问题的实质就是在网格环境下,把 m 个任务 $T=\{t_0,t_1,\cdots,t_{m-1}\}$ 以合理的方式调度到 n 个主机 $H=\{m_0,m_1,\cdots,m_{n-1}\}$ 上去,并希望得到尽可能小的总执行时间(makespan).

1.2 任务调度算法的分类和特点

现有的任务调度算法可以分为静态调度算法和动态调度算法.

静态调度算法是指所有的任务-机器映射策略在执行任务调度前就已经全部确定,常用的静态调度算法包括 OLB、MET、MCT、Min-min、Max-min、Duplex、GA(遗传算法)、SA(模拟退火算法)、GSA(遗传模拟退火算法)、A* 等. 静态调度算法基于所有任务信息已知,拥有较好的算法结果. 但是该算法属于 NP 完全问题.

动态调度算法是指部分任务-机器映射策略在执行任务调度期间根据实际情况进行确定. 动态调度算法又分成在线模式(online mode)和批模式(batch mode)两种. 在线模式是指任务一到来就映射到机器,该模式对每一个任务的映射只考虑一次,也就是说一旦任务被映射就不会再改变. 批模式下,任务到来并不立即映射到机器,而是把任务收集起来组成一个任务集合,等映射事件到来后才对该集合中的任务进行集中映射. 动态调度最明显的缺点是运行期的开销,但可以获得更加灵活的调度策略.

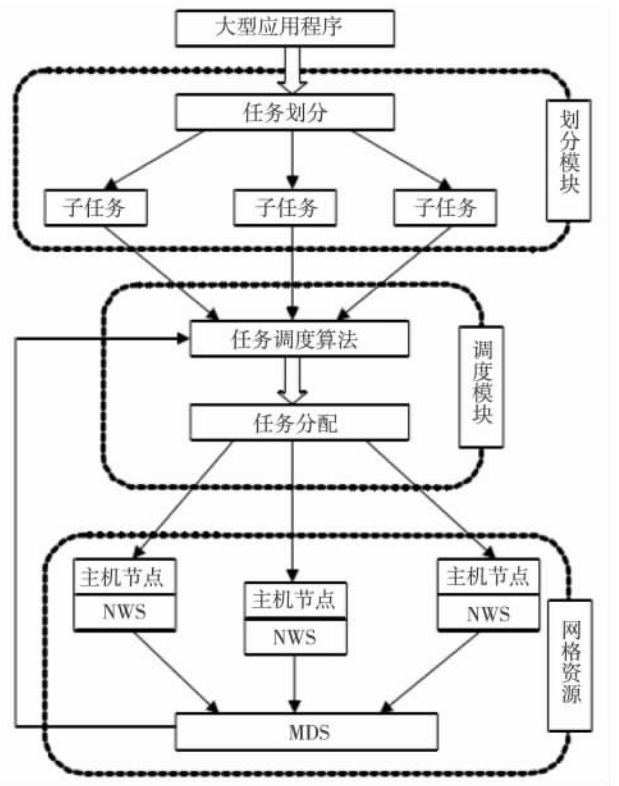

图 2 典型的网格任务调度流程

Fig. 2 A typical workflow of job scheduling in grid

1.3 EASY-backfill 算法

在线模式的任务调度算法中,FCFS(先来先服务)是最常用简单的算法(图 3). EASY-backfill[5-6]则是在 FCFS 的基础上,加上回填策略,将队列中较小的作业回填(Backfill)到空闲 CPU,以提高 CPU 利用率. 注意到由于小作业的回填有可能使得后提交的小作业在先提交的大作业之前执行(图 4 的 J3 迟于 J2 提交但早于 J2 执行).

经典的 EASY-backfill 算法[5]流程如下.

1) 寻找阴影时间与额外节点

① 把正在运行的任务根据期望完成时间进行排序;

② 在可用节点的列表中收集足够数量的节点已满足等待队列中的第一个任务;

③ 在找到满足上面条件的节点的时间点即为阴影时间;

④ 如果在这个时间的可用节点数超过了队列

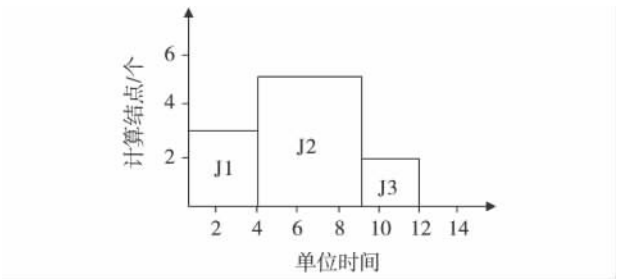

图3 FCFS 算法的任务序列

Fig. 3 The job sequence of FCFS algorithm

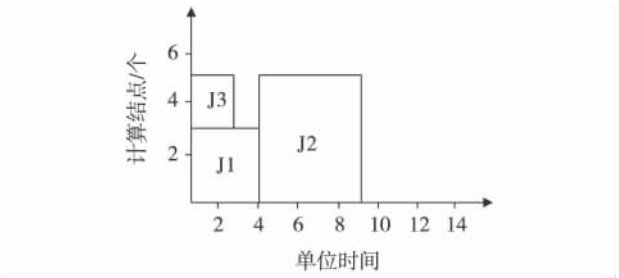

图4 EASY-backfill 算法的任务序列

Fig. 4 The job sequence of EASY-backfill algorithm

中第一个任务所需要的节点数,那么这些多出来的节点被称为额外节点.

2) 寻找回填的任务

① 按照任务到达的顺序检索等待队列中的任务;

② 对于等待队列中的每一个任务,检查是否满足以下条件:该任务需要的节点数不大于额外节点的数目,并且将在阴影时间前完成,或者该任务所需要的节点数不大于当前最小的空闲节点数目和额外节点数目;

③ 符合条件的第一个任务即可用来进行回填(backfilling).

EASY-backfill 作为一种简单但较高效的算法,引起了很多关注,并有很多围绕改进该算法的研究在进行中. 如对于该算法的性能研究[7],或者针对任务的期望完成时间对算法的性能影响进行探讨[8]. 研究的角度除了从任务的各类参数出发外,也有部分从网格资源的角度出发的研究,如提出来的动态调整任务 CPU 数目以获得更多的额外节点用于回填[14].

2 算法设计

2.1 问题的提出

EASY-backfill 算法的实现,一般来说基于理想均衡稳定的计算资源,即任务在选择空闲计算结点时,是随机的和无差别的. 算法仅考虑是否有足够的空闲结点时间供任务使用,在空闲结点数大于需求数时,是无偏向性地随机选择结点的. 但是由于网格本身的异构特性,在实际运行环境中各个结点的计算资源在数量、速度、稳定性上都会有所不同. 因此可以对该算法进行改进,对空闲计算资源结点按照一定的方法进行选择,避免使用性能较低或者稳定性差的结点,避免任务的运行延迟和运行失败,提高任务的 Qos 级别.

2.2 不稳定资源的模型

由于任务调度设计的资源类型很多,为简化问题并不失一般性,本文仅讨论不稳定计算资源的模型.

网格任务的相关参数如表 1 所示.

表1 网格任务的相关参数

Table 1 Some parameters of job in grid

参数	说明
RPE	运行所需的 CPU 总数
ST	任务提交时间.
AST	任务开始时间
EFT	期待任务结束时间,EFT = AST + EL(不考虑网络、I/O 延迟)
AFT	实际任务结束时间
EL	期待任务长度
AL	实际任务长度,AL = AFT - AST(不考虑网络、I/O 延迟)
QR	Qos 系数,QR = EL/AL

假设网格的共有 N 个主机(计算资源),为了体现计算资源的不稳定性和异构性,对于每一个计算资源 Machine_i 引入以下几个参数:

MR: CPU 的指令运行速度,这个是固定参数.

PE: CPU 的数量.

p: 稳定系数($0 < p < 1$). 结点以($1 \sim p$)的概率出现不稳定状态,影响在其上面运行的所有任务,p 为 1 时表示结点绝对稳定.

d: 延迟系数. 结点在出现不稳定状态时,将导致正在上面运行的所有任务的运行时间变为原期待运行时间的($1 + d$)倍,d 为 0 时表示不会出现延迟.

u: 实际性能系数. 假设在 Machine_i 实际运行的任务共有 c_i 个,分别是 $\text{job}_1, \text{job}_2 \cdots \text{job}_{Ci}$,$c_i$ 大小与 i 无关,而是根据算法实际运行结果而定. u 可以表示为 $u = \left(\sum_{j=1}^{C_i} R_{Q,\text{job}_{Ci}}\right) / C_i$.

结点的不稳定性主要由 p,d 体现,同时随着结点的不稳定,任务 R_Q 参数将反馈影响结点的 u.

2.3 算法框架

改进算法与经典算法的不同主要在于在任务调度的过程中,增加了资源监视器组件. 其主要流程如下.

1) 用户向网格提交计算任务. 任务中包括有 ST、ALT 等参数. 任务调度器接收到计算任务,并开始进行调度.

2) 任务调度器根据 EASY-backfill 算法对进行处理,确定该任务是开始执行还是加入队列. 如果任务开始执行,则向资源监视器询问计算资源的稳定性参数,包括 p,d,u 等.

3) 任务调度器根据从资源监视器获得的信息,在可用计算结点中选择稳定性最佳的结点满足任务需求.

4) 任务完成后,更新所在计算结点的稳定性参数 u.

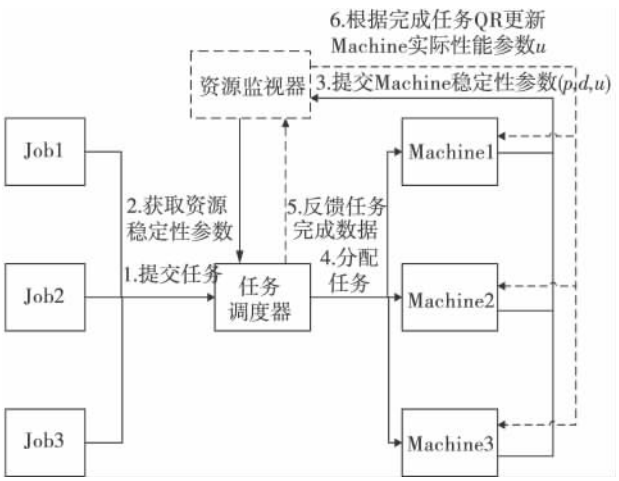

图 5 改进的 FCFS 算法流程

Fig. 5 Workflow of improved FCFS algorithm

2.4 算法流程

根据框架设计,得到算法流程(伪代码). 任务调度器主流程,负责接收用户提交的任务. 如果根据计算资源的空闲情况,该任务当前可以开始回填执行(backfill),则直接执行任务,否则将任务加入到队列中. 在经典算法中,每次回填仅考虑队列的第 1 个任务,即 SingleBackfill 策略. 本算法也沿用此策略.

任务调度器主流程如下:

```
While( hasJobSubmited( ) {
  currentJob→Scheduler;
  pivotJob = Scheduler. queue. header
  if ( privot. canStart( ) )  privotJob. start( ) ;
  if ( currentJob. canStart( ) )
    currentJob. start( ) ;
  else   currentJob. queue( ) ;
}
```

Job. start( ) 是启动任务 job 的过程. 首先获取当前可用的计算资源列表,该列是按照所有的计算资源的实际性能参数 u 降序排列的. 依序为 job 分配计算资源,直到满足 job. RPE 的要求. 算法可以保证 job 所选择的计算资源是可用列表中性能最优的.

Job. start( ) 流程如下:

```
AvaMachineList = 当前时间可用的计算资源列表
Foreach( machine in AvaMachineList) {
    Job. RPE = Job. RPE-machine. PE;
    //为 Job 分配资源 Machine
    machine. Run( job) ;
If ( job. RPE = =0)  break;
    }
    Job. ST = now;
    Job. AST = now;
    Job. EFT = Job. AST + job. EL;
    Job. setStatus( RUNNING) ;
    Job. startWork( ) ;
```

Job. schdule( ) 是将任务加入等待队列的流程. 如果等待队列不为空,直接将任务加入到队列结尾.

Job. schdule ( ) 流程如下:

```
    Schedule. queue. add( job) ;
    If schedule. queue. header!  = null return;
    Job. ST = now;
    Job. setStatus( QUEUE) ;
```

Job. onFinsh( ) 是在任务完成后进行调用的流程. 在该流程中,需要修改 Job 的相关参数. 同时根据此次 Job 的执行情况(QR)来反馈计算资源的实际性能参数.

Job. onFinish( ) 流程如下:

```
    Job. AFT = now;
    Job. AL = job. AST + job. AFT;
    Jor. QR = AL/EL
    Foreach( machine in job. delayMachineList) {
    machine. u = (ma chine. u* ma chine. finishcount + job. QR) / (ma chine. finishCount + 1)
    }
```

3 实验

3.1 实验环境及几个限定条件

本文采用 GridSim[9] 作为模拟仿真试验环境.

GridSim 是澳大利亚墨尔本大学开发的一个基于 Java 离散事件的网格仿真工具. 该工具支持对异种网格资源(资源可以分时和分空间共享)、用户和应用的建模与仿真. 它提供了创建应用任务、映射任务到资源和管理的方法. 所有或部分操作的统计数据都可以记录下来,并使用 GridSim 的工具进行分析. 实验所使用的负载数据来自 http://www.cs.huji.ac.il/labs/parallel/workload/.

为了简化实验流程,在不影响实验结果的条件下,设定以下几个限定条件:1) 仅考虑网格任务的计算时间,不考虑网络、I\O 等其他延迟时间;2) 回填算法仅考虑等待队列的第一个任务,即使用 GridSim 所提供的 SingleBackFill 回填策略;3) 网格任务的延迟仅取决于其稳定系数(p)和延迟系数(d);4) 当网格任务在多个计算资源上运行时,其延迟仅取决于对于其延迟影响最大的一个计算资源. 每个任务仅被延迟一次;5) 计算资源对于网格任务的影响仅限于其运行时间. 其他的诸如 MR、PE 等参数在仿真过程中均不变化.

3.2 关于网格任务延迟的仿真方法

由于原始的负载并不存在计算资源导致任务延迟的参数,所以需要在 GridSim 仿真的过程中根据预先设定的参数自行模拟计算资源的不稳定以及任务的延迟,同时完成对于计算资源实际性能参数 u 的反馈.

具体的方法是在每次有新网格任务提交时,检视所有正在运行的任务,并遍历任务所运行的所有计算资源节点,以概率 $1-p$,计算延迟后的任务长度 DL = EL(1 + d),并取 DL 最长的做为实际结果. 流程如下:

```
When a job submitted:
Foreach( job_i in RunningJobs) {
    Foreach( machine_ij in job_i. machineList) {
        以概率( 1-machine_ij. p) 运行以下代码:
        DL = job_i. EL* (1 + machine_ij. d)
        If (DL > job_i. AFT) {
          job_i. AL = DL;
          job_i. AFT = job_i. AST + job_i. AL;
        }
    }
}
```

3.3 稳定系数参数有效性

首先通过实验证明稳定系数对于 makespan(总完成时间)的影响,即证明不同的稳定系数,将对全部任务的总完成时间有影响. 通过选取一个计算结点数固定的计算资源集,改变在该资源集中的不稳定资源的数量,得到不稳定率不同的计算资源集. 实验使用改进算法,将同一个 Workload(sdsc-par-1996)在这些资源集上进行仿真模拟,得到不同的 makespan. 对于稳定节点,设置稳定系数 $p=1$,即以不会对在之上运行的任务产生影响;对于不稳定节点,设置不稳定系数 $p=0$,即以 1 的概率对在之上运行的任务产生影响.

表 2 不同的不稳定率测试数据

Table 2 Test data of different unstability rate

节点总数	稳定节点数	不稳定节点数	不稳定比率
96	96	0	0
96	24	72	0.75
96	48	48	0.50
96	72	24	0.25
96	0	96	1.00

根据图 6 的实验结果可以看出,随着计算资源集不稳定率的上升,整个 Workload 的 makespan 是随之增大的. 这和我们所理解的常识是一样的,即计算资源越不稳定,任务所完成的总时间就越长. 说明不稳定资源各系数的导入,正确反映了对于网格任务影响的事实.

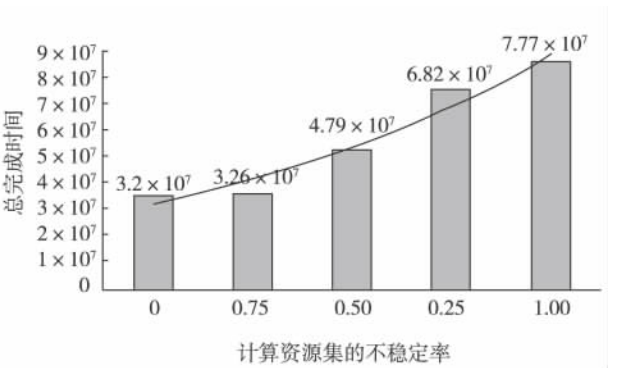

图 6 计算资源集的不稳定率对 makespan 的影响

Fig. 6 The influence on makespan of different unstability rate of computing resource

接下来测试使用改进算法计算资源的稳定系数的影响. 取一个节点数量固定的计算资源集,其中 10% 为不稳定资源. 设置不稳定资源的稳定系数(p)分别为 0,0.1,0.3,0.7,0.9,1 进行实验,选择的 Workload 为 sdsc-par-1996. 实验数据见图 7,随着稳定系数的提高,任务的 QosRate 和 makespan 均明显下降. 说明在仿真试验中,计算资源的稳定系数确实

对于网格任务有可预知的影响.

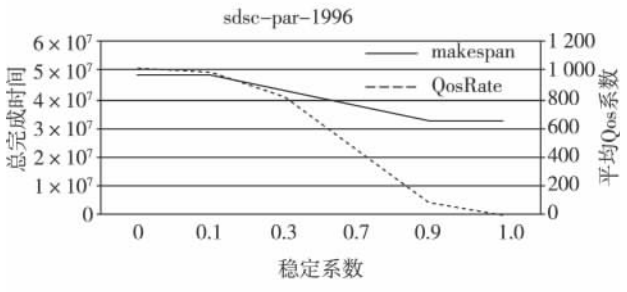

图 7　不稳定资源的稳定系数 p 对 makespan 的影响

Fig. 7　The influence on makespan of different p of unstable computing resources

3.4　算法对比

首先使用 Parallel Workload Archive 的实际网格任务日志来比较在不稳定计算资源的环境中，经典算法和改进算法的 makespan 以及任务的平均 Qos 系数. 同时考察计算资源数量、质量对算法的影响. 选择的 Workload 如下:

表 3　实验使用的 Workload

Table 3　The workload used in experiment

	Sdsc_blue_2	Sdsc-par-1996-2.1	LLNL-Atlas-2006-1.1
总任务数	1 250	38 719	60 332

同时考虑到计算资源对于算法的影响，选取数量不同的计算资源集合进行仿真. 每组资源中的所有计算结点均为不稳定资源，并且以不同的稳定性按照 90% 和 10% 的比例分成两类.

表 4　300 个计算结点的计算资源集参数

Table 4　The parameters of computing resource set with 300 nodes

数量	MIPSRating	PE	稳定系数	延迟系数	实际性能系数
30	377	8	0.8	2	1
270	377	8	0.2	5	1

图 8 为在不同的计算资源集下，经典算法与改进算法在 makespan 方面的比较，数据均为 100 次仿真模拟的平均值. 根据图表显示，在计算结点较小的时候，改进算法比经典算法没有优势，甚至 makespan 更加大. 这是因为资源太少的时候，无法满足网格任务的需要，计算资源的时间片的占用比较拥挤，因此即使算法避免在不稳定计算资源上进行任务的运行，但是由于资源的有限性，任务无法避开不稳定资源，甚至由于改进算法本身的运算消耗，导致了改进算法的 makespan 反而更大. 但是随着计算节点的逐步增加，任务可以比较宽松地选择更加稳定的计算资源，因此在图中看来，改进算法的 makespan 开始比经典算法有了明显的减小. 计算节点继续增多，资源供大于求，基本不影响算法的效果.

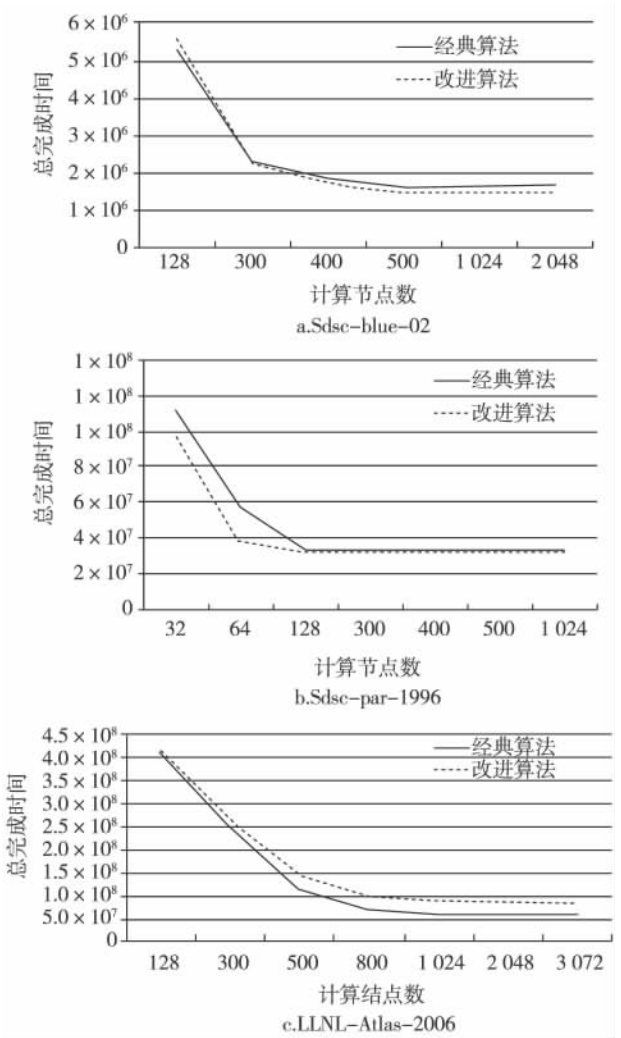

图 8　算法比较(makespan)

Fig. 8　The comparions of two algorithms(makespan)

根据图 9，改进算法对于网格任务的平均 Qos 系数的改进也有着同样的效果.

4　结论及未来的工作

本文引入稳定系数、延迟系数、实际性能系数等参数对不稳定计算资源进行了描述，并在此基础上对于经典的 EASY-backfill 算法进行了改进，以实现在任务调度的过程中对于计算资源不稳定性的敏感. 在计算资源的不稳定性参数有着不同的设置时，

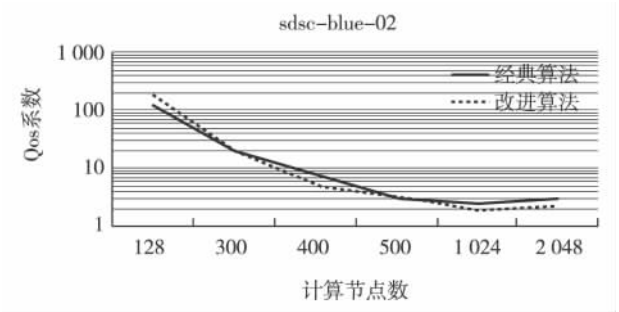

图9 算法比较

Fig. 9 The comparion of two algorithms

算法能够有效选择较稳定的计算资源进行使用. 实验证明该算法在计算资源相对充足的环境下,能够改善网格任务的 makespan 和平均 Qos 系数.

未来的研究工作主要集中在对于不稳定计算资源的进一步建模上,考虑使用更加符合实际情况的故障率正态分布函数来描述资源的不稳定性差别. 同时对算法进行改进,使 makespan 和 Qos 获得更大幅度的改善. 同时使得在计算资源紧张时,效率比经典算法有更大的提升.

参考文献

References

[1] Foster I. Internet computing and the emerging grid. NatureWeb Matters [EB\OL]. (2000-12-07). http://www.nature.com/nature/webmatters/grid/grid.html

[2] Foster I, Kesselman C. The Grid: BluePrint for a new computing inrfastruetuer [M]. San Fransisco: Morgan Kaufmann Publishers Inc,1999

[3] Foster I. What is the Grid? A three point cheeklist [J]. GRID Today,2002,1(6):22-25

[4] Wolski R,Spring N,Hayes J. The network weather service: A distributed resource performance forecasting service for metacomputing [J]. Future Generation Computer Systems,1999,15(5/6):757-768

[5] Lifka D. The ANL/IBM SP scheduling system [C] // Proceedings of JSSPP,1995:295-303

[6] Talby D,Feitelson D. Supporting priorities and improving utilization of the ibm sp scheduler using slack based backfilling [C] // Proceedings of the 13th International and 10th Symposium on Parallel and Distributed Processing,1999:513-517

[7] Wong A K L,Goscinski A M. Evaluating the EASY-backfill job scheduling of static workloads on clusters [C] // IEEE International Conference on Cluster Computing,2007:64-73

[8] Wong A K L,Goscinski A M. The impact of under-estimated length of jobs on EASY-Backfill scheduling [C] // The 16th Euromicro Conference on Parallel,Distributed and Network-Based Processing,2008:343-350

[9] Buyya R,Murshed M. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing [J]. Concurrency and Computation: Practice and Experience,2002,14(13/14/15):1175-1120

[10] Parallel Workload Archive [EB\OL] [2010-08-25]. www.cs.huji.ac.il/labs/parallel/workload

[11] 付云虹,白树仁,方俊. 基于 Backfilling 调度算法的"扩履适足"改进算法 [J]. 计算机工程与科学,2006,28(9):94-96
FU Yunhong,BAI Shuren,FANG Jun. An algorithm for Backfilling-based "Enlarge Five to Ten" parallel job scheduling [J]. Computer Engineering & Science,2006,28(9):94-96

Research on modified EASY-backfill algorithm for grid with unstable resource

WANG Zhengyu[1] XIAO Nanfeng[1]

1 School of Computer Science & Engineering,South China University of Technology,Guangzhou 510006

Abstract Grid connects distributed resource into a mass environment for computing. The scheduling algorithm has much influence over the efficiency and performance of grid. EASY-backfill is a classical scheduling algorithm with simple program,little computation workload and relatively high performance. But the algorithm is based on assumptions that all resources in the grid are absolutely stable,and the performance forecast of jobs is accurate and reliable. Yet these conditions are ideal and unpractical. In this paper,we use some unstable parameters to build a different model of unstable grid resources,and modify the EASY-backfill algorithm according to the model. We make effort to keep the performance and effectiveness of the algorithm on condition that the modified algorithm sensitive to the unstable and unpredictable grid resource. We also make comparison between the classical and modified algorithm and do some discussion on the influence of different unstable parameters.

Key words grid computing; job schedule; EASY-backfill