

网格环境下基于分块的任务执行时间预测算法

吉 勤,李培峰,朱巧明,马锋明

(苏州大学 计算机科学与技术学院,江苏 苏州 215006)

(jq712@163.com; 064227064006@suda.edu.cn)

摘 要:针对当前已有预测算法不实时、对负载变化不敏感的问题,结合网格中任务的特点,提出新的基于分块的预测算法。该方法从子节点任务执行出发,提出把执行的任务分成两个部分,用前一部分的执行情况来预测剩下部分的执行时间。实验证明,该算法比已有预测算法有更高的效率和通用性。

关键词:预测算法;网格;自调度;分块

中图分类号: TP393 **文献标志码:** A

Task executing-time prediction algorithm based on chunk in grid

Ji Qin, Li Pei-feng, Zhu Qiao-ming, Ma Feng-ming

(School of Computer Science and Technology, Soochow University, Suzhou Jiangsu 215006, China)

Abstract: Concerning the issues of no real-time and not sensitive to load change, this paper proposed a new prediction algorithm based on chunk taking account of the task's characteristics in grid. Considering the task execution on grid nodes, this algorithm divided the task into two parts and used the execution time of the first part to predict that of the remaining part. The experimental results show that this method can achieve higher efficiency and generality.

Key words: prediction algorithm; grid; self-scheduling; chunk

0 引言

网格能够大规模地共享计算资源、存储资源、数据资源、软件资源等,为大数据量、高性能的计算提供了一个平台。网格系统是多用户的,系统中通常会运行多个应用,为了获得高性能,调度成为网格环境中的一个关键问题。自调度^[1]是调度的一种,它是一种在应用层的调度技术,指一个任务可以分解成多个子任务,然后每个任务分布到各个节点并行执行,从而提高响应时间的技术。在自调度中,调度节点和被调度节点间要进行多次的数据传输,由于网格环境中各节点具有分布、异构、动态、自治等特点,在每一时刻各节点的负载都是不同的,因此在不适当的时刻传输数据可能会造成整个系统性能的下降。但是,如果在调度前就知道节点上当前任务的执行时间,则这个问题就迎刃而解。在目前的大多数文献中,所给出的调度算法都是假定任务的执行时间是已知的^[2],但这种假定是不符合实际的。在网格环境中,提前确切地知道任务的执行时间是不可能的,本文采用一种根据节点的实时执行情况来预测任务的执行时间,实验证明该方法能取得比较好的系统性能。

1 网格环境下任务执行时间预测算法

目前,对网格环境下任务执行时间的预测研究得比较多,但总体来说可以分为两大类:基于时间序列的预测和基于因果关系的预测。

1.1 基于时间序列的预测

所谓基于时间序列的预测是指在预测当前任务的执行时

间时,利用之前已经执行完的相似任务的执行时间,通过一定的计算方式来估计当前任务的执行时间。按照具体计算方式的不同,可分为以下几种,公式中 t_j 是第 j 个任务的实际执行时间, T_{n+1} 是当前任务的预测执行时间。

1) 平均值法。该方法用当前任务之前已经完成的任务的执行时间的平均值作为当前任务的预测执行时间,公式为:

$$T_{n+1} = \sum_{j=1}^n t_j / n \quad (1)$$

2) k 近邻法^[3-4]。该方法在预测时,选择离当前任务最近的 k 个任务的执行时间,对它们进行加权平均来作为当前任务的预测执行时间,公式为:

$$T_{n+1} = \frac{\sum_{j=1}^k w_j \times t_j}{\sum_{j=1}^k w_j} \quad (2)$$

其中, w_j 为权值,实际在计算时,由于离当前任务越近,影响越大,所以其 w_j 的值选得越大,越远则 w_j 越小。在 k 近邻法中,还有一个问题就是 k 值的选择,通常让其等于 $n^{4/5}$, n 为总任务量。

3) 指数平均值法^[5]。计算公式为:

$$T_{n+1} = \alpha t_n + (1 - \alpha) T_n \quad (3)$$

其中, α ($0 \leq \alpha \leq 1$)可以看成是 t_n 的权值,与 k 近邻方法不同的是, k 近邻中需要确定 k 个 w_j 的值,而这里只需要确定一个 α 值,其他的都可以通过展开公式计算得到。该公式的展开形式为:

$$T_{n+1} = \alpha t_n + (1 - \alpha) \alpha t_{n-1} + \cdots + (1 - \alpha)^j \alpha t_{n-j} + \cdots + (1 - \alpha)^{n-1} T_0 \quad (4)$$

收稿日期:2009-02-05;修回日期:2009-03-20。

基金项目:国家863计划项目(2006AA01Z147);国家自然科学基金资助项目(60673041)。

作者简介:吉勤(1984-),女,江苏淮安人,硕士研究生,主要研究方向:网络计算、分布式系统;李培峰(1971-),男,江苏吴江人,副教授,主要研究方向:中文信息处理、网络计算;朱巧明(1963-),男,江苏昆山山人,教授,主要研究方向:中文信息处理、网络计算;马锋明(1981-),男,江苏南通人,硕士研究生,主要研究方向:网络计算与分布式系统。

大小的1/2,与前面所说的平均值法、 k 近邻法、指数平均值法和不用预测方法进行比较,以分词来进行实验,分别做了子节点空闲和子节点有负载的情况,如图2、3所示。

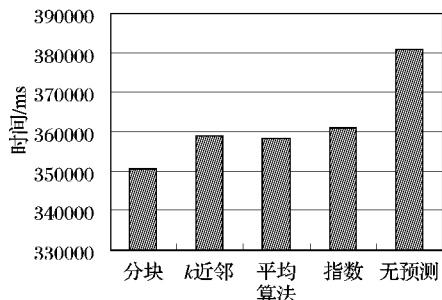


图2 各算法完成时间(无负载)

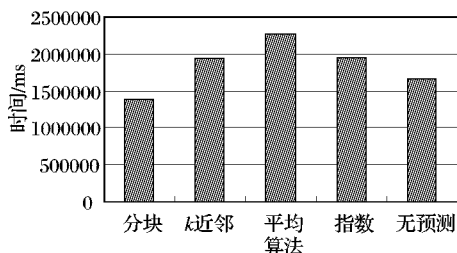


图3 各算法完成时间(有负载)

对比图2、3可以看出:

1) 在子节点没有负载的情况下,使用预测算法比没有用预测的性能要好,而在预测算法中,基于分块的预测性能最好,比 k 近邻、平均、指数三种算法性能平均提高2.4%,比不使用预测算法性能提高7.9%;

2) 在子节点有负载的情况下,由于子节点的负载在不停变化,且变化是不规律的,而现有预测方法都是基于历史执行的值来预测,导致误差较大,性能比没有预测要差,而使用数据块的前一部分的执行时间来预测剩下部分的执行时间,可以实时反映子节点的负载变化情况,误差较小,因此性能较好,比 k 近邻、平均、指数三种预测算法性能平均提高32.1%,比不使用预测算法性能提高16.7%。

3.3 最优块预测与非最优块预测的比较

从3.2节中可以看出,基于分块的预测算法无论是在子节点有负载的情况还是没有负载的情况都能取得比较好的性能。但是就基于分块的预测方法本身来说,预测块大小的不同也能影响最终的性能。这里,人为随机指定预测部分大小与按照2.2节中介绍的方法计算出的预测部分大小进行比较。

3.3.1 计算密集型应用

计算密集型是指需要占用大量的CPU时间而所需的数据量很小,通常在科学计算领域比较常见,譬如矩阵相乘、Mandelbrot等。本文采用矩阵相乘作为计算密集型的代表,其他计算密集型的应用与此相似。实验中选用的是两个 2000×2000 的矩阵相乘,分别在子节点有负载和无负载两种情况下做了实验,如图4、5。

图4、5中,8行、9行、11行指的是人为指定预测部分大小分别为8行、9行、11行,而计算指的是用2.2节中指定的方法算出来的来的预测部分大小。

从图4、5可以看出,在子节点无负载时,计算出预测块大小比随机指定块大小性能平均提高9.5%,在子节点有负载时,计算出预测块大小比随机指定大小性能平均提高1%,因此无论是有负载还是没有负载,在计算密集型的应用中,实时计算出来预测部分大小所取得的性能比随机指定的性能

要好。

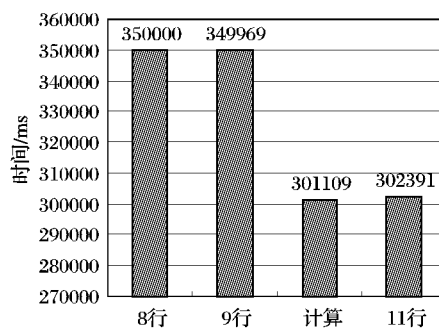


图4 矩阵相乘完成时间(无负载)

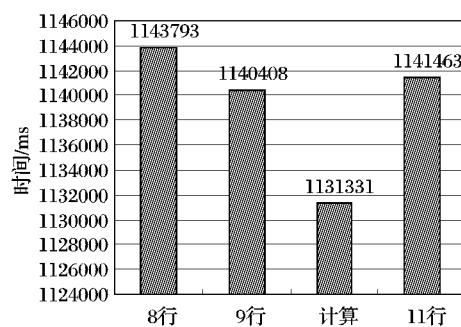


图5 矩阵相乘完成时间(有负载)

3.3.2 数据和计算均密集型应用

数据和计算均密集型是指任务不仅需要占用大量的CPU时间,还需要传输大量数据,通常在数据和信息处理领域比较常见。本文采用汉语分词作为代表来进行测试。汉语分词主要的功能是把汉语的自然文本进行以词为单位的切分,由于其算法一般采用统计和机器学习的方法,所以需要大量的计算。另外,如果进行大量文本的切分,也需要传输大量的数据。实验中采用的测试文本大小为20 MB,同样分别在子节点有负载和无负载两种情况下做了实验,如图6和7。

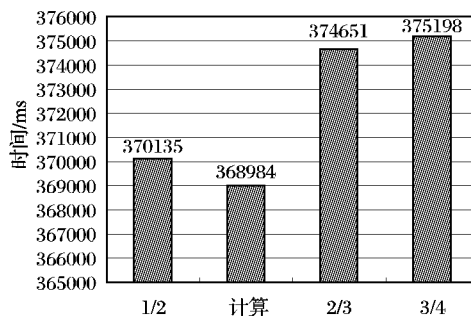


图6 分词完成时间(无负载)

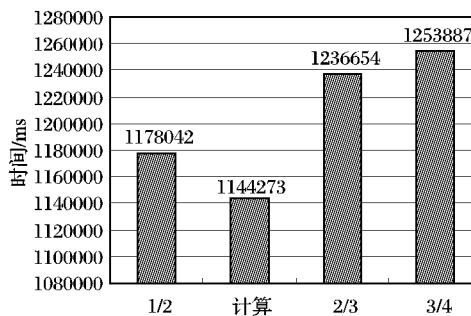


图7 分词完成时间(有负载)

图6和7中,1/2、2/3、3/4表示用来预测的部分是整个部分的1/2、2/3、3/4,而计算指的是用2.2节中指定的方法算出来的预测部分大小。

(下转第1889页)

458 143 330 105 214 494 169 208 189 492 153 121 221 373 414
 220 171 319 209 155 361 455 229 371 369 484 98 247 230 267
 443 469 305 66 344 413 200 263 144 405 91 165 271 5 299 26
 467 148 224 386 223 250 390 365 80 173 269 425 332 194 85
 210 280 428 160 297 195 135 379 394 473 205 67 113 359 463
 402 372 110 70 166 60 116 162 246 231 136 33 31 277 43 315
 272 150 57 309 289 415 301 2 4 471 36 19 438 74 199 498 216
 96 59 252 178 8 245 307 61 102 460 241 182 176 384 417 35
 196 115 253 483 279 477 53 185 286 447 211 29 44 292 306 40
 364 242 314 391 186 274 65 331 77 396 203 103 273 108 445
 183 429 406 380 6 497 322 407 177 295 472 111 450 397 235
 304 453 481 288 239 418 276 448 129 90 180 95 500 366 163
 325 62 68 464 248 78 22 491 339 466 362 174 181 38 64 122
 490 442 12 52 23 175 408 237 93 430 139 489 119 222 311 219
 100 156 480 63 46 81 432 329 94 352 161 377 347 437 499 327
 190 316 55 179 21 337 323 112 446 346 388 476 228 217 256
 207 487 215 449 486 120 427 341 37 333 79 308 275 451 89 356
 1 488 423 17 45 468 28 404 114 421 296 441 106 147 7 399 131
 298 284 435 202 470 152 34 358 360 291 461 382 71 16 482 374
 99 191 474 25 479 9 11 495 146 342 69 240 58 172 86 157 334
 376 475 75 32 82 313 293 262 439 20 227 478 30 395 125 410
 354 170 393 310 496 324 39 117 56 137 15 336 27 3 10 151 389
 357 431 84 266 411 392 132 140 124 212 403 370 159 184 109
 54 234 419 193 440 465 353 51 123 126 436 300 48 485 278 493
 422 232 50 343 142 243 149 188 127 282 387 133 164 351 76
 302 226 328 14 192 138 130 348 264 420 378 238 118 158 251
 401 225 197 257 290 433 141 452 24 398 244 368 72 285 134
 462 400 456 88 104 338 349 249 198 87 312 83 409 92 13 201
 326 459 270 49 97 457 218 261 73 281 145 260 268 320 444 340
 47 128 294 383 213 335 254 204 363 42 426 265 154 303 18 355
 321 385 350 259 283 412 187 167 255 41 317 424 287 375

从运行的结果看,尽管遗传算法理论上的结果是近似解,

但本文算法所得的结论却是精确解,这充分反映了本算法的优点。当 $n = 100$ 时,耗时约 124 ms; 当 $n = 500$ 时,耗时约 19 s,并且多次在 1 s 以内找到问题的解,实验中没有发现不收敛的情况。另外,求解 1000 皇后问题所用时间平均只有约 3 min,其中多次在 1 min 以内找到该问题的解,而求解 3000 个皇后所用时间平均约为 55 min,有时能在几分钟内就能得到该问题的精确解。而对于没有基因评估函数的算法,其求解速度几乎要慢 10 倍以上,而且随着皇后数量的增加,其收敛速度的差别会更大。

5 结语

采用个体评估与基因评估双适应函数,将最差的基因进行变异,改善了单亲遗传算法搜索全局最优解的方向,避免了算法对个体基因的盲目变异,从而加快了演化速度。实验结果表明,该算法具有演化速度快、全局性能好、不易陷入局部最优解的特点。当然,由于该算法只对一个个体进行演化,初始群体的好坏对演化速度有一定的影响。另外 n 皇后问题较易设计出其基因评估函数,具有一定的特殊性,能否将个体评估和基因评估双重评估方法推广到一般的情形,使其能解决一般的优化问题,有待进一步研究。

参考文献:

(上接第 1876 页)

从图 6 和 7 可以看出,在子节点无负载时,计算出预测块大小比随机指定块大小性能平均提高 1.2%,在子节点有负载时,计算出预测块大小比随机指定块大小性能平均提高 6.4%,因此无论是有负载还是没有负载,在数据和计算均密集型的应用中,实时计算出来预测部分大小所取得的性能比随机指定的性能要好。

4 结语

本文针对现有网格环境中预测方法存在的问题,提出了基于分块的预测的方法。该方法的主要特征是在预测时利用数据块的前一部分的执行时间来预测剩下部分的时间,达到实时、精确的效果。实验证明,该方法与现有预测方法相比,具有更好的执行效率和通用性。

参考文献:

- [1] YANG CHAO-TUNG, CHENG KUAN-WEI, SHIH WEN-CHUNG. On development of an efficient parallel loop self-scheduling for grid computing environments[J]. *Parallel Computing*, 2007, 33(7/8):

- 467-487.
- [2] MAHESWARAN M, ALI S, SIEGEL H J, *et al.* Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems[C]// *Proceedings of the 8th IEEE Heterogeneous Computing Workshop*. Washington, DC: IEEE Computer Society, 1999: 30-44.
- [3] PHINIJAROENPHAN P, BEVINAKOPPA S, ZEEPHONGSEKUL P. A method for estimation the execution time of a parallel task on a grid node[C]// *EGC 2005*, LNCS 3470. Berlin: Springer, 2005: 226-236.
- [4] IVERSON M, OZGUNER F, POTTER L. Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment[C]// *Proceedings of the 8th Heterogeneous Computing Workshop*. Washington, DC: IEEE Computer Society, 1999: 15-29.
- [5] 栾翠菊,宋广华,郑耀,等.一种网格并行任务执行时间预测算法[J]. *计算机集成制造系统*, 2007, 13(9): 1806-1810.
- [6] 韩耀罗,罗雪梅.网格计算环境下任务执行时间的组合预测[J]. *计算机工程*, 2006, 32(21): 70-72.