同构网络可分任务调度问题

一、问题描述

设有N+1台处理机通过星型网络互连，如图1所示，其中P_0为主处理机，其余为从处理机，主处理机只负责数据的切分和传输，从处理机负责数据的处理。l_i是链接P_0和P_i的通信链路,其中$0 < i \leq N$。由于该网络为同构网络，所以从处理机的能力一样，即每个从处理机之间数据传输速度与处理速度相等。

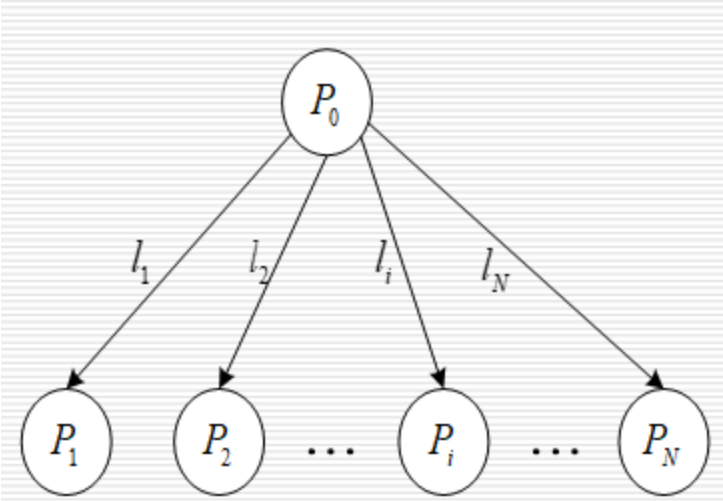


图1 同构星型网络平台

主处理机P_0将大小为W_{total}的任务数据切分为n个子数据块,大小为$\alpha_1, \alpha_2, \ldots, \alpha_n$，并且均为正，有$1 \leq n \leq N$，满足$\sum_{i=1}^{n} \alpha_i = W_{total}$。依次传输给n个相同的按照任意顺序排列的从处理机$P_1, P_2, \ldots, P_n$,主处理机$P_0$同一时刻只能给一个从处理机$P_i$传输任务，$P_i$只有在完全接收子任务后才开始处理该任务。并不要求所有的处理机都必须参与处理任务。

令z表示从主处理机P_0传输单位数据到P_i所需要的时间，则传输大小为α_i的数据所需的时间为$z\alpha i$。又因为从处理机传输数据存在启动开销e，所以子任务的传输时间为$e + z\alpha i$。

令w表示从处理机P_i处理单位数据的时间，则处理数据α_i的时间为$w\alpha i$。又因为从处理机处理数据存在启动开销f，所以子任务的处理时间为：$f + w\alpha i$。

设从处理机P_i的释放时间为r_i,即从开始到时刻r_i,从处理机P_i是非空闲的。从处理机从时刻r_i开始空闲，可以给其安排任务。记从处理机P_i开始接受任务的时刻为s_i，根据数据的传输顺序，从处理机P_1最先开始接受传输任务。所以对于处理机P_1,由于其在时刻r_1时开始空闲，故对于从处理机P_1有$s_1 = r_1$。当$2 \leq i \leq n$时，从处理机P_i的开始时间s_i由上一从处理机P_{i-1}传输的完成时间以及该处理机的空闲时间决定，在两者之间取最大值。前一个处理机P_{i-1}的开始时间s_{i-1}加上接受子任务α_{i-1}的传输时间$e + z\alpha i$，为$si - 1 + e + z\alpha i$;从处理机Pi的空闲时间为r_i，所以从处理机P_i接收数据的开始时间为$s_i = max s_{i-1} + e + z\alpha i, ri$

从处理机P_i完成数据处理的时间为$s_i+e+z\alpha i+f+w\alpha i$，当且仅当每个从处理机都处理完数据时整个同构网络可分任务才算完成，则最终完成数据处理的时间为
$\max_{1\leq i\leq n}\left(s_i+e+z\alpha_i+f+w\alpha_i\right)$
。由于对于从处理机的排列顺序也会影响到最终数据处理的完成时间，所以在若干种可能的排列顺序中选择最优解作为我们的调度方案，即有$\min_{P_1,P_2,\ldots,P_n}\max_{1\leq i\leq n}\left(s_i+e+z\alpha_i+f+w\alpha_i\right)$
。

对上面的条件进行整理有

$$\text{s.t.}\begin{cases}1\leq n\leq N\\0\leq\alpha_i\leq W_{total},1\leq i\leq n\\\sum_{i=1}^{n}\alpha_i=W_{total}\\s_{i-1}+z\alpha_{i-1}+w\alpha_{i-1}=s_i+z\alpha_i+w\alpha_i,\ 2\leq i\leq n\\C=(c2,c3,\ldots,cn)\\s_i=\begin{cases}r_1 & \text{if } i=1\\\max\left\{s_{i-1}+e+z\alpha_i,r_i\right\} & \text{if } 2\leq i\leq n\end{cases}\end{cases}$$

二、问题建模优化

1. 从处理机的执行顺序优化

对于给出的若干个从处理机，由于各个处理机之间的性能相同，所以为了尽可能减少任务调度所花的时间，就需要尽可能让空闲时刻早的先进行传输处理数据任务。如果从处理机按照乱序排列，必然存在一种情况，使得释放时间$r_i>r_{i+1}$，导致尽管后面的处理机已经释放处于空闲状态，但是任务却因等待当前处理机进入空闲状态而不进行任何操作，浪费了大量的等待时间。所以不妨对从处理机根据释放时间进行从早到晚的排序，让早释放的处理机先执行任务使得所花等待释放时间最小，有
$P_1,P_2,\ldots,P_n$满足$r_1\leq r_2\leq\ldots\leq r_n$。

2. 参与处理任务的从处理机数目优化

星型同构网络上共有N个从处理机，现在我们需要对启用的处理机数量n进行讨论。如果不考虑从处理机的释放时间，在原则上，当启用了尽可能多的从处理机时，每个从处理机所分配到的任务被均摊，使得最后处理所花费的时间更少。但是由于存在从处理机的传输和处理任务的启动开销，当启动开销大于均摊任务所需要的时间时，增加处理机的数量得到的收益为负。同理可知，当从处理机需要考虑释放时间时，增加处理机所需要花费的时间包括启动开销以及可能的等待空闲开销。我们需要在解决问题模型时需要考虑确定一个临界值使得从处理机工作的效率最大化。

3. 处理任务的结束时间优化

根据上文，从处理机最终完成数据处理的时间为$\max_{1\leq i\leq n}\left(s_i+e+z\alpha_i+f+w\alpha_i\right)$，并且只有当所有处理机同时完成计算时任务所需要的完成时间最短。否则可以调度尚未完成计算任务的处理机，

将部分任务分配给先完成计算的处理机上执行。于是我们根据上文知$s_{i-1}+e+z\alpha i-1+f+w\alpha i-1=\ s_i+e+z\alpha i+f+w\alpha i$，整理后得到

$s_{i-1}+z\alpha i-1+w\alpha i-1=\ s_i+z\alpha i+w\alpha i,\ 2\le i\le n$ （1）

4. 子任务的分配方案优化

根据上文，从处理机P_i开始接受任务的时刻为$s_i=\begin{cases}r_1 & \text{if } i=1\\ \max\left(s_{i-1}+e+z\alpha_i,r_i\right) & \text{if } 2\le i\le n\end{cases}$，不妨进行讨论。

(1)若上一个子任务的分配完成时间均在释放时间之后，定为约束条件1

即有

$s_i=s_{i-1}+e+z\alpha_i\ ,\ 2\le i\le n$ （2）

将（1）式与（2）式整理得

$$\alpha_i=\frac{w}{z+w}\alpha_{i-1}-\frac{e}{z+w},2\le i\le n$$

(2) 若上一个子任务的分配完成时间均在释放时间之前，定为约束条件2

即有

$s_i=r_i,1\le i\le n$ (3)

将（1）式与（3）式整理得

$$\alpha_i=\alpha_{i-1}-\frac{r_i-r_{i-1}}{z+w},1\le i\le n$$

累加后可得

$$\alpha_i=\alpha_1+\frac{r_1-r_i}{z+w},1\le i\le n$$

（3）混合时序约束

我们不妨对混合时序约束进行讨论.任意两个相邻的从处理机之间可能满足约束条件(1),也可能满足约束条件(2).若有n台处理机参与计算,所有可能的情况有2^{n-1}种。设定混合时序约束为C=(c2,c3,…,cn)。

对于给定的某一种时序约束，可以得到相邻处理机开始时间之间的关系:

$$
\begin{gathered}
s_1 = r_1, \\
s_2 = s_1 + e + z\alpha_1, \\
\cdots \\
s_k = s_{k-1} + e + z\alpha_{k-1}, \\
s_{k+1} = r_{k+1}, \\
s_{k+2} = r_{k+2}, \\
\cdots \\
s_{k+m} = r_{k+m}, \\
s_{k+m+1} = s_{k+m} + e + z\alpha_{k+m}, \\
\cdots \\
s_n = s_{n-1} + e + z\alpha_{n-1}
\end{gathered}
$$

(4)

将式(4)带入式(1),可得每台处理机分配的任务α_i:

$$
\begin{gathered}
\alpha_2 = \frac{w}{z+w}\alpha_1 - \frac{e}{z+w}, \\
\cdots \\
\cdots \\
\cdots \\
\alpha_k = \frac{w}{z+w}\alpha_{k-1} - \frac{e}{z+w}, \\
\alpha_{k+1} = \alpha_1 + \frac{r_1 - r_i}{z+w}, \\
\alpha_{k+2} = \alpha_1 + \frac{r_1 - r_i}{z+w}, \\
\cdots \\
\cdots \\
\cdots \\
\alpha_{k+m} = \alpha_1 + \frac{r_1 - r_i}{z+w}, \\
\alpha_{k+m+1} = \frac{w}{z+w}\alpha_{k+m} - \frac{e}{z+w}, \\
\cdots \\
\cdots \\
\cdots \\
\alpha_n = \frac{w}{z+w}\alpha_{n-1} - \frac{e}{z+w}.
\end{gathered}
$$

(5)

将式(5)中的n-1个等式同$\sum_{i=1}^{n}\alpha_i = W_{total}$共n个等式表示成标准形式,即

$$
A \bullet \alpha = b
$$

其中α表示的是待求的n×1维解变量(α1,α2,…,αn),A是n×n的系数矩阵,b是n×1维的向量。

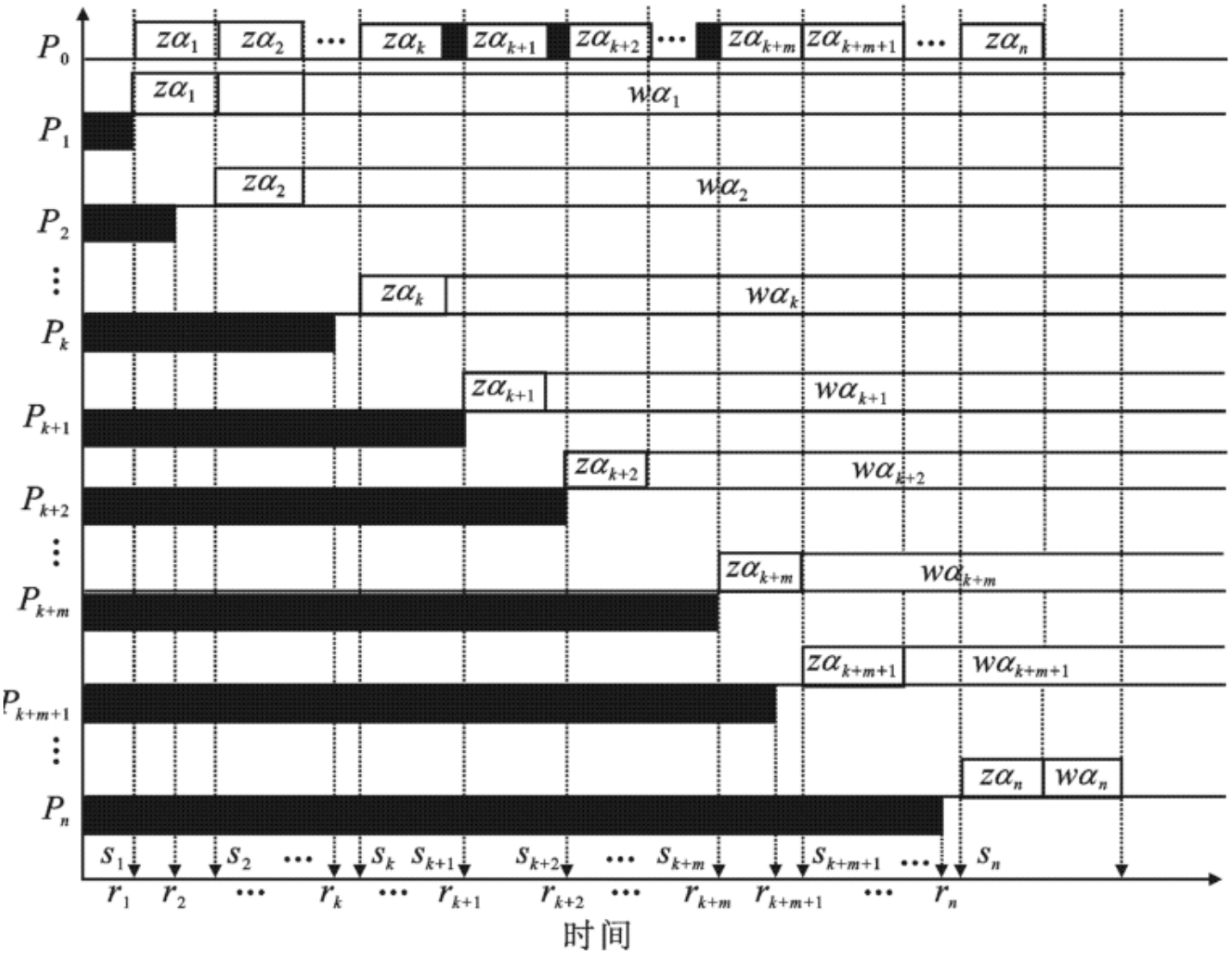
$$
b = [-\frac{e}{z+w}, \ldots, -\frac{e}{z+w}, \frac{r_1 - r_i}{z+w}, \frac{r_1 - r_i}{z+w}, \ldots, \frac{r_1 - r_i}{z+w}, -\frac{e}{z+w}, \ldots, -\frac{e}{z+w}, W_{total}]
$$

即可求出任务分配方案α的解.下面给出考虑释放时间的可分任务调度模型:

$$\min_{n,C}(T) = \min\left(r_1 + e + f + (z+w)\alpha_1\right)$$

$$\text{s.t.}\begin{cases} 1 \le n \le N \\ 0 \le \alpha_i \le W_{total}, 1 \le i \le n \\ \sum_{i=1}^{n} \alpha_i = W_{total} \\ s_{i-1} + z\alpha_{i-1} + w\alpha_{i-1} = s_i + z\alpha_i + w\alpha_i, \ \ 2 \le i \le n \\ C = (c2, c3, \ldots, cn) \\ s_i = \begin{cases} r_1 & \text{if } i = 1 \\ \max\{s_{i-1} + e + z\alpha_i, r_i\} & \text{if } 2 \le i \le n \end{cases} \end{cases}$$

调度图：



梯度下降与牛顿迭代

对函数$f(x) = \sum_{i=1}^{5}\left[(1-x_i)^2 + 100(x_{i+1} - x_i^2)^2\right]$，初始点取为$x_0 = (0, 0, \ldots, 0) \in \mathbb{R}^6$，分别用最速下降法和牛顿编程迭代10次，结果记录：

迭代次数k	最速下降法	牛顿法
1	4.94211464	4.94211464
2	4.89815621	4.63350489
3	4.81260383	4.52764763
4	4.63223753	4.52086354
5	4.48031619	4.46103915
6	4.36349505	4.45437617
7	4.30909567	4.09120261
8	4.26579133	4.00735888
9	4.22601659	3.85641375
10	4.16691051	3.85707386
10 次迭代两个方法		
所花时间	1.449毫秒	2.005毫秒

分析

根据提供的计算结果，我们可以分析最速下降法和牛顿法在求解优化问题时的性能差异：

收敛性能

- 最速下降法 的函数值从初始的 4.94211464 逐渐降低到 4.16691051。这显示了最速下降法能够稳定地减少函数值，但下降速度较慢，这与最速下降法依赖于梯度方向作为搜索方向的特性一致。由于梯度方向并不总是指向最陡的下降方向，特别是在复杂的多维空间中，这可能导致路径出现“之字形”并减慢收敛速度。
- 牛顿法 的函数值从初始的 4.94211464 快速下降至 3.85707386。牛顿法在前几次迭代中迅速减少函数值，显示出比最速下降法更快的收敛速度。这是因为牛顿法考虑了函数的二阶导数信息（即Hessian矩阵），能够更准确地估计函数的局部曲率，从而在每一步选择更优的下降方向和步长。然而，值得注意的是，在第9次和第10次迭代中，函数值略有上升，这可能是由于Hessian矩阵的逆计算不准确或者步长选择不当导致的。

时间开销

- 最速下降法 花费了 1.449 毫秒完成 10 次迭代，而 牛顿法 花费了 2.005 毫秒。尽管牛顿法在每次迭代中要计算更多的信息（如Hessian矩阵及其逆），这自然导致了更高的计算成本，但从总体收敛速度来看，牛顿法仍然是更高效的选择。这表明牛顿法在求解此类优化问题时，尽管单次迭代的时间成本更高，但由于迭代次数少，总体上可能更有效率。

结论

- 收敛速度：牛顿法明显快于最速下降法，这反映了利用二阶导数信息能够更有效地指导搜索过程的优势。
- 稳定性：牛顿法在迭代过程中可能会遇到稳定性问题，如函数值的小幅回升，这需要通过适当的步长控制和Hessian矩阵的正定性调整来解决。
- 计算成本：虽然牛顿法的单次迭代成本更高，但其较快的收敛速度可能在许多情况下抵消了这一点，尤其是在需要快速收敛到高精度解的应用中。

建议

- 对于复杂或高维的优化问题，可以考虑使用拟牛顿法（如BFGS或L-BFGS），这些方法试图平衡牛顿法的快速收敛性和计算Hessian矩阵的高成本。
- 在使用牛顿法时，应当注意Hessian矩阵的正定性和逆的准确性，必要时可以采用正则化或修改Hessian矩阵的方法来提高稳定性。
- 动态调整步长（如通过线搜索）对于加速收敛和保证算法稳定性是非常重要的，尤其是在牛顿法中。