High Multiplicity Scheduling of File Transfers with Divisible Sizes on Multiple Classes of Paths

Mugurel Ionut Andreica, Nicolae Tapus

HIGH MULTIPLICITY SCHEDULING OF FILE TRANSFERS WITH DIVISIBLE SIZES ON MULTIPLE CLASSES OF PATHS

Mugurel Ionut Andreica, Nicolae Tapus

Polytechnic University of Bucharest, Computer Science Department, Bucharest, Romania

ABSTRACT

Distributed applications and services requiring the transfer of large amounts of data have been developed and deployed world wide. The best effort model of the Internet cannot provide these applications with the so much needed quality of service guarantees, making necessary the development of file transfer scheduling techniques, which optimize the usage of network resources. In this paper we consider the high multiplicity scheduling of file transfers over multiple classes of paths with the objective of minimizing the makespan, when the files have divisible sizes. We also consider another objective, that of maximizing the total profit, in the context of some special types of mutual exclusion constraints (tree and clique constraint graphs).

Index Terms— file transfer scheduling, divisible sizes, high multiplicity, makespan minimization, greedy, binary search, mutual exclusion, tree, clique, dynamic programming.

1. INTRODUCTION

Providing QoS guarantees for the transfer of multiple files over several network paths is a challenging task, because of the wide range of parameters which need to be considered and the many problems that need to be solved. In the first part of this paper we present optimal algorithms for minimizing the maximum completion time of the file transfers (the makespan) in a particular situation, in which the paths are disjoint and the files have divisible sizes. In the second part, we present efficient algorithms (polynomial, pseudopolynomial and exponential) for maximizing the profit of the scheduled file transfers, under special types of mutual exclusion constraints (tree or clique constraint graphs). We connect the two parts at the end of the paper.

This paper is organized as follows: in Section 2 we define the makespan minimization problem for files with divisible sizes and in Section 3 we present an optimal scheduling algorithm for it. In Section 4 we improve the algorithm, considering a special case. In Section 5 we present the model of scheduling file transfers with mutual exclusion constraints. In Sections 6, 7 and 8 we present efficient solutions to scheduling problems with special types of mutual exclusion constraints. In Section 9 we present some practical applications of the techniques proposed in this paper. In Section 10 we discuss related work and in Section 11 we conclude and present future work.

2. MAKESPAN MINIMIZATION WITH DIVISIBLE FILE SIZES

We consider C different file types: for each type, the size of the file ($sz_i>0$) and the number of files ($nf_i>0$) are given. We consider the file types sorted in decreasing order of their sizes: $sz_1>sz_2>\ldots>sz_C$. The sizes further satisfy the condition that $sz_C|sz_{C-1}|...|sz_1$, i.e. $sz_i=sq_i \cdot sz_{i+1}$ ($1 \le i < C$), where $sq_i>1$ is an integer number. We are concerned with scheduling the transfer of all the files using P classes of disjoint paths, subject to makespan minimization. For each class, the number of paths in the class ($np_i>0$) and their slowdown factor ($sd_i>0$) are given. The transfer of a file of type j on a path of type i will take $sd_i \cdot sz_j$ time units to complete. We consider the path classes sorted in increasing order of their slowdown factor: $sd_1<sd_2<...<sd_P$. On each path, only one file can be transferred at one time and the file transfers are non-preemptive.

3. OPTIMAL FILE TRANSFER SCHEDULING

We will present a polynomial-time algorithm for the makespan minimization problem, with time complexity *O(P·C·log(Tmax))*, where Tmax is an upper bound for the value of the makespan. In order to minimize the makespan, we use binary search between $Tmin=sd_1 \cdot sz_1$ and

$$Tmax = sd_1 \cdot \sum_{i=1}^{C} sz_i \cdot nf_i \qquad (1)$$

where we select a potential value T for the makespan. Then we will perform a feasibility test, checking if all the transfers can be scheduled and completed on the P classes of paths within time T. If the values of the file sizes and slowdown factors are integers, then this algorithm finds an exact solution. If they are real numbers, then it will find a solution which is arbitrarily close to the optimum. For two positive real numbers A and B, such that A=q·B+r, where q≥0 is an integer and 0≤r<B, we denote the value of *q* by *A div B* and the value of *r* by *A mod B*. The function that tests if T is a feasible value for the makespan is based on a greedy strategy; its pseudocode is showed below:

ChkT(T,C,sz_1,...,sz_C,nf_1,...,nf_C,P,sd_1,...,sd_P,np_1,…,np_P)
 for *j=1* **to** *C* **do** $qtotal_j=0$
 for *i=1* **to** *P* **do**
  $q_{i,1}=T$ **div** $(sd_i \cdot sz_1)$
  $r_{i,1}=T$ **mod** $(sd_i \cdot sz_1)$
  $qtotal_1 = qtotal_1 + np_i \cdot q_{i,1}$
  for *j=2* **to** *C* **do**
   $q_{i,j}=r_{i,j-1}$ **div** $(sd_i \cdot sz_j)$
   $r_{i,j}=r_{i,j-1}$ **mod** $(sd_i \cdot sz_j)$
   $qtotal_j = qtotal_j + np_i \cdot q_{i,j}$
 $maxperiods_1=qtotal_1$
 for *j=1* **to** *C* **do**
  if $(maxperiods_j<nf_j)$ **then return** *"no"*
  if $(j<C)$ **then**
   $maxperiods_{j+1}=(maxperiods_j-nf_j) \cdot sz_j/sz_{j+1}+qtotal_{j+1}$
 return *"yes"*

For every path class i and file type j, we compute the values $q_{i,j}$ and $r_{i,j}$, defined like in the pseudocode above. $q_{i,1}$ represents the maximum number of type 1 files that can be transferred on a path of class i within the time T. In general, $q_{i,j}$ represents the maximum number of files of type j that can be transferred on a path of type i, considering that $q_{i,j'}$ files of each type j'<j have already been transferred on that path. With these values, we will iteratively compute another set of values, $maxperiods_j$, representing the maximum number of time periods into which files of type j can be scheduled, considering that all the files of type i<j have already been scheduled on some paths. For each file type, $maxperiods_j$ needs to be at least equal to nf_j.

The running time of the algorithm **ChkT** is $O(P \cdot C)$. The algorithm can be implemented such that it uses only $O(P+C)$ memory, because the $q_{i,j}$ and $r_{i,j}$ values do not need to be stored after moving from one class of paths i to the next.

4. IMPROVED FEASIBILITY TEST FOR C=1

In this section we present an improved feasibility test for the case C=1, when the sz_i and sd_i values are integers and the sd_i values form an arithmetic or geometric progression, i.e. $sd_i= sd_{i-1}+K$ or $sd_i=sd_{i-1} \cdot K$, for $1<i\leq C$ and some known constant K. In this case, only the values $q_{i,1}$ will be computed. The P classes of paths can be split into G groups, such that group 1 contains the classes 1,…,nc_1, group i ($1<i\leq C$) contains the classes $nc_{i-1}+1$,..,nc_i, $P=nc_G$ and for any two classes of paths i and j in the same group, we have $q_{i,1}=q_{j,1}$. G can be at most 2·sqrt(T) (where sqrt(T) is the square root of T), because: there are at most sqrt(T) classes i for which $(sd_i \cdot sz_1)$ is at most sqrt(T) and each of these classes can form a group all by itself; all the other classes have $(sd_i \cdot sz_1)>sqrt(T)$ and for each such class i, the value $q_{i,1}$ is less than sqrt(T), so there can only exist sqrt(T) different possible values for $q_{i,1}$, thus forming at most sqrt(T) groups. We also compute the values:

$$sumnp_i = \sum_{j=1}^{i} np_j \qquad (2)$$

ChkTC1(T,C=1,sz_1,nf_1,P,sd_1,…,sd_P,np_1,…,np_P,K)
 firstCls=1; $qtotal_1=0$; $sumnp_0=0$
 for *i=1* **to** *P* **do** $sumnp_i=sumnp_{i-1}+np_i$
 while *(firstCls≤P)* **do**
  $q=T$ **div** $(sd_{firstCls} \cdot sz_1)$
  if *(q=0)* **then break**
  $r=T$ **mod** $(sd_{firstCls} \cdot sz_1)$
  $dif=(r / q) / sz_1$

$$extraCls = \begin{cases} \left\lfloor \frac{dif}{K} \right\rfloor, & \text{for an arithmetic progression} \\ \left\lfloor \frac{\log\left(1+dif/sd_{firstCls}\right)}{\log(K)} \right\rfloor, & \text{for a geometric progression} \end{cases}$$

$extraCls=min\{P\text{-}firstCls, extraCls\}$

$qtotal_1=qtotal_1+(sumnp_{firstCls+extraCls} - sumnp_{firstCls-1})\cdot q$

$firstCls=firstCls+extraCls+1$

if $(qtotal_1<nf_1)$ **then return** "no"

else return "yes"

The feasibility test presented above has $O(sqrt(T))$ time complexity. If the sd_i values of the paths do not form an arithmetic or geometric progression, the algorithm can be changed so that it computes the value of extraCls by using a binary search (finding the largest class of paths i=firstCls+extraCls for which sd_i is not larger than $sd_{firstCls}$+dif), reaching a time complexity of $O(sqrt(T)\cdot log(P))$ for the test. The test can also be used when the sd_i and sz_i values are not integers, but then we cannot provide the $O(sqrt(T))$ guarantee for the number of groups.

5. SCHEDULING WITH MUTUAL EXCLUSION CONSTRAINTS

We are given C types of files which need to be transferred on a specific path: for each type i, the size of the file (sz_i) and the number of files (nf_i) are given. All the files of the same type have to be transferred sequentially, without preemption and at the maximum possible transfer rate r_{path} from the source to the destination. The profit obtained for transferring all the files of type i is p_i (this profit is 0 if we cannot transfer all the files of the same type). Since the files belonging to the same class have to be transferred together, we compute a parameter $d_i=(nf_i\cdot sz_i)/r_{path}$, representing the duration required for transferring all the files of type i. The paths of some pairs of file classes may intersect, which could degrade the transfer performance of both file types if they are scheduled during overlapping time intervals. Although more advanced scheduling techniques could be employed, we choose a simple, practical model, of constructing a mutual exclusion graph in which the vertices are the file types and there is an edge between two vertices if the paths of the corresponding file types intersect. The problem parameters are represented by the durations d_i, the profits p_i and the mutual exclusion graph. The scheduling objective is to maximize the total profit, under the restriction that two file transfers between which a mutual exclusion constraint exists are not scheduled during overlapping time intervals. From a practical point of view, each file type could correspond to a file transfer request submitted to a file transfer scheduler, which processes the requests in batches. For each batch, it computes the request parameters and the mutual exclusion graph. In the following sections, we will consider this practical model. We will present efficient solutions to several scheduling problems which fit into this model. The results we present next can be considered independent of the makespan minimization problem handled in Sections 2, 3 and 4. A connection between the two problems will be made in Section 9.

6. SCHEDULING WITH A TREE MUTUAL EXCLUSION GRAPH

We are given a common deadline T, meaning that each file transfer request i must be scheduled during a time interval $[t, t+d_i]$, included in [0,T]. The mutual exclusion graph is a tree (the results can be easily generalized to forests). We want to schedule some of the file transfer requests (and reject the others) in such a way that the total profit of the scheduled file transfers is maximum. We provide an $O(C)$ dynamic programming algorithm for this problem. First, we choose a root for the tree (the vertex r), which defines the parent-son relationships between the vertices of the tree. For each vertex i, we compute two values:

- A(i) = the maximum profit obtained by scheduling some of the requests in vertex i's subtree and letting vertex i be one of the scheduled requests.
- B(i) = the maximum profit obtained by scheduling some of the requests in vertex i's subtree with request i being rejected.

We will assume that d_i is at most T for all vertices i; otherwise, vertex i cannot be scheduled under any circumstance (A(i) will be set to $-\infty$). For each leaf i of the tree, we have $A(i)=p_i$ and B(i)=0. For a vertex i which is not a leaf, we have:

- $A(i) = p_i + \sum_{\text{json of i}} \begin{cases} \max\{A(j),B(j)\}, & \text{if } d_j \le T - d_i \\ B(j), & \text{if } d_j > T - d_i \end{cases}$
- $B(i) = \sum_{\text{json of i}} \max\{A(j),B(j)\}$

For the case of B(i), the equation is obvious. If we do not schedule B(i), then there is no restriction for the choices we can make for each son of i. If we choose to schedule i, then the chosen time interval will be $[0,d_i]$. For each son j of i, we may choose not to schedule it, in which case vertex i will not influence any vertex in vertex j's subtree, or to schedule it. In the second case, we only have the interval $[d_i,T]$ available, which does not include the interval $[0,d_j]$, which we considered when computing A(j). However, the time moments 0 and T are equivalent, so the interval $[0,d_j]$ is equivalent to the interval $[T-d_j,T]$. Thus, we can use the value of A(j), considering that j would be scheduled at the other end of the time horizon, i.e. in the interval $[T-d_j, T]$, if $d_i \leq T-d_j$. The maximum profit is $\max\{A(r), B(r)\}$.

7. SCHEDULING WITH A TREE MUTUAL EXCLUSION GRAPH AND OTHER TIME CONSTRAINTS

This problem is similar to the one presented previously. We are given a time horizon T, C file transfer requests with an assigned duration and profit and a tree mutual exclusion graph. Furthermore, each file transfer has an earliest start time $ES_i \geq 0$ and a latest finish time $LF_i \leq T$. This means that request i should be scheduled within an interval $[t, t+d_i]$ fully included in $[ES_i, LF_i]$. We will provide a pseudopolynomial dynamic programming algorithm for this problem, for the case when all the time values are integers and when the file transfers can only be scheduled at integer time moments. This may be the case if we consider the time horizon to be divided into unit time slots and the file sizes to be an integer multiple of the transfer rate. The second condition can be fulfilled by artificially increasing the file sizes. First, we will root the tree at a vertex r. This way, we define the parent-son relationships. For each tree vertex i, we compute two arrays:

- $S_i(t)$=the maximum achievable profit if the time interval available for scheduling request i is [0,t] (and considering only the vertices in vertex i's subtree)
- $F_i(t)$=the maximum achievable profit if the time interval available for scheduling request i is [t,T] (and considering only the vertices in vertex i's subtree)

For each vertex i, we have:

- $S_i(0)=F_i(T)=\sum_{\text{j son of i}} S_j(T)$
- $$S_i(t) = \max_{0<t\leq T}\left(S_i(t-1), \begin{cases} p_i + \sum_{\text{j son of i}} \max\{S_j(t-d_i), F_j(t)\} \\ \qquad\qquad (\text{if } ES_i \leq t-d_i \text{ and } t \leq LF_i) \\ 0 \qquad\qquad (\text{otherwise}) \end{cases} \right)$$
- $$F_i(t) = \max_{0\leq t<T}\left(F_i(t+1), \begin{cases} p_i + \sum_{\text{j son of i}} \max\{S_j(t), F_j(t+d_i)\} \\ \qquad\qquad (\text{if } ES_i \leq t \text{ and } t+d_i \leq LF_i) \\ 0 \qquad\qquad (\text{otherwise}) \end{cases} \right)$$

The maximum profit is $S_r(T)$. These values can be computed in $O(C \cdot T)$ time using a bottom-up traversal of the tree.

8. SCHEDULING WITH A CLIQUE MUTUAL EXCLUSION GRAPH

The problem parameters are identical to the previous case, except that any two file transfers cannot be scheduled simultaneously. Thus, the mutual exclusion graph is a clique. Considering that the requests can only be scheduled at integer time moments (and that all the time parameters are integers) and under the supplementary assumption that the number of requests is small (the requests are scheduled in batches whose size is bounded), we provide a dynamic programming algorithm for scheduling a subset of requests with maximum total profit. For every time moment t and subset of requests S, we compute P[t,S], the maximum profit gained by scheduling a subset of requests included in S, considering only the time interval [0,t]. We set P[0,S]=0, for all the subsets S. For t>0 and any subset S, we have:

$$P[t,S] = \max\left(\begin{cases} \max_{i\in S, ES_i \leq t-d_i, t\leq LF_i} \left(p_i + P[t-d_i, S\setminus\{i\}]\right) \\ P[t-1,S] \end{cases} \right)$$

The maximum profit is $P[T,\{1,2,\ldots,C\}]$. An $O(T \cdot C \cdot 2^C)$ algorithm is straight-forward.

9. PRACTICAL APPLICATIONS

The techniques presented in this paper can be implemented in a two-stage centralized file transfer scheduler, handling batches of file transfer requests in a network or a distributed system. In the first stage, file transfers are grouped according to the

source and destination pair. The requests in the same group are divided into several types, based on how close their file sizes are. The file sizes are then rounded up, such that all the files of the same type have the same size and the file sizes of different types are divisible. This rounding up may be performed efficiently in several situations, for instance when transferring multimedia content (with sizes being multiples of a CD or DVD size).

Each group is handled independently, using the makespan minimization algorithm. All the files of the same type which are scheduled on the same path are sent consecutively and at a constant rate, according to this algorithm. In the second stage, all the files of the same type which are scheduled on the same path are grouped into a virtual file transfer request. These virtual requests are assigned priorities (or profits) and are handled by the mutual exclusion scheduling algorithms presented in the second part of the paper. The techniques can also be used in distributed collaborative systems, like media streaming or file sharing peer-to-peer systems [5]. In this case, the peers organize themselves into small groups and all the transfers inside such a group are scheduled in order to improve the performance.

10. RELATED WORK

File transfer scheduling is related to the problems of job scheduling and resource allocation [1]. In [2], an $O(P \cdot logT)$ high multiplicity multiprocessor scheduling algorithm is presented for C=2, given the maximum value for the makespan T, but the sizes are not considered to be divisible. Other related problems are bin-packing with variable bin sizes and divisible item sizes [3] and the multiple knapsack problem [4], for which several objectives have been considered (minimizing the number of bins, minimizing costs, maximizing the profit). Mutual exclusion scheduling problems with different constraint graphs (trees, permutation graphs, comparability graphs) were considered in [6,7]. However, they only considered requests with unit duration and a common deadline. The algorithms we presented introduce variable durations and start and finish time constraints into mutual exclusion scheduling problems.

11. CONCLUSIONS AND FUTURE WORK

In this paper, we presented an optimal $O(P \cdot C \cdot log(Tmax))$ algorithm for scheduling file transfers with divisible file sizes, in order to minimize the makespan. In the case C=1, very large P and integer problem parameters, the algorithm's complexity was improved to $O(sqrt(Tmax) \cdot log(Tmax))$ if the slowdown factors form an arithmetic or geometric progression, and to $O(log(P) \cdot sqrt(Tmax) \cdot log(Tmax))$ otherwise. These algorithms perform better than other techniques which solve more general versions of the problem. As future work, we will try to extend in an efficient manner the improved feasibility test from Section 4 to the case C>1. We also presented some efficient scheduling algorithms handling simple mutual exclusion constraint graphs, like trees (forests) and cliques. The next step consists of finding efficient (exact or approximate) algorithms for more general classes of constraint graphs. In the end, we presented some potential practical applications of the techniques developed in this paper.

12. REFERENCES

[1] K. Pruhs, J. Sgall, and E. Torng, *Online Scheduling*, CRC Press, 2004.

[2] S.T. McCormick, S.R. Smallwood, and F.C.R. Spieksma, "A Polynomial Algorithm for Multiprocessor Scheduling with Two Job Lengths," *Mathematics of Operations Research*, pp. 31-49, 2001.

[3] E.G. Coffman, Jr., M.R. Garey, and D.S. Johnson, "Bin packing with divisible item sizes," *Journal of Complexity*, pp. 406-428, 1987.

[4] C. Chekuri, and S. Khanna, "A PTAS for the Multiple Knapsack Problem," *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms*, pp. 213-222, 2000.

[5] J.A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D.H.J. Epema, M. Reinders, M. van Steen and H.J. Sips, "Tribler: A social-based peer-to-peer system," *Concurrency and Computation:Practice and Experience*, vol. 20, pp. 127-138, 2008.

[6] K. Jansen, "The Mutual Exclusion Scheduling Problem for Permutation and Comparability Graphs," *Information and Computation*, vol. 180(2), pp. 71-81, 2003.

[7] B. Baker, and E.G. Coffman, Jr., "Mutual exclusion scheduling," *Theoretical Computer Science*, vol. 162(2), pp. 225–243, 1996.