2011 IEEE International Parallel & Distributed Processing Symposium

Offline Scheduling of Multi-Threaded Request Streams on a Caching Server

Veronika Rehn-Sonigo
Franche-Comté University
Besançon, France
Email: veronika.sonigo@lifc.univ-fcomte.fr

Denis Trystram
Institut Universitaire de France
Grenoble University, France
Email: denis.trystram@imag.fr

Frédéric Wagner
Grenoble University
France
Email: frederic.wagner@imag.fr

Haifeng Xu, Guochuan Zhang
Zhejiang University
Hanghzou, China
Email: {xuhaifeng, zgc}@zju.edu.cn

***Abstract*—In this work, we are interested in the problem of satisfying multiple concurrent requests submitted to a computing server. Informally, there are users each sending a sequence of requests to the server. The requests consist of tasks linked by precedence constraints. Tasks may occur several times in the same sequence as well as in a request sequence of another user. The computing server has to execute tasks with variable processing times. The server owns a cache of limited size where intermediate results of the processing may be stored. If an intermediate result for a task is stored into the cache, no processing cost has to be paid and the result can directly be fetched from the cache.**

The goal of this work is to determine a schedule of the tasks such that an optimization function is minimized (the only objective studied up to now is the makespan). This problem is a variant of caching which considers only one sequence of requests. We then extend the study to the minimization of the mean completion time of the request sequences. Two models are considered. In the first model, caching is forced whereas in the second model caching is optional and one can choose whether an intermediate result is stored in the cache or not. All combinations turn out to be NP-hard for fixed cache sizes and we provide a formulation as dynamic program as well as bounds for inapproximation. We propose polynomial time approximation algorithms for some variants and analyze their approximation ratios. Finally, we also devise some heuristics and present experimental results.

***Index Terms*—scheduling chains; complexity; inapproximation;**

I. Introduction

A. Presentation of the problem

We consider in this work a scheduling problem of satisfying multiple concurrent requests on a computing server. Informally, there are users each sending a sequence of requests to a server. The requests correspond to tasks linked each other by precedence relations depending on their positions in the request chain. Some tasks may occur several times in the same sequence as well as in a request sequence of another user. Each task is associated with a processing time and the computing server treats the tasks in any order compatible with the precedence relations. The server also owns a cache of bounded capacity where intermediate results of the processing may be stored. As far as an intermediate result for the treatment of a task is currently stored into the cache, no processing time is incurred in the use of this task since its result can be directly fetched from the cache. The goal is to determine a schedule of all the tasks so that a certain objective function is minimized.

This problem is called multi-threaded caching, denoted as MTC. A typical application of this problem is in geographic information sciences and has been developed in the frame of the French Geobench Project. The analysis of the functionalities has been presented to the parliament of the European Community in 2007 [14]. Today, a lot of statistical information are available on every European territory at different scales (levels of refinements, from countries, regions, departments to cities). Such information is a great potential for computing indicators that help to take economical and political decisions. Basic numerical and statistical methods have been implemented on a server and users (who are usually decision makers in territorial collectivity or large companies) submit a sequence of requests to compute some indicators on a given geographic region. Examples of such computations can be found in [20]. Several tasks may be the same at various refinement levels in the same sequence and often, several users are submitting similar tasks in their requests.

B. Our contribution

The main contribution of this work is to initialize the study of this interesting problem by exploring the various models from the complexity to solution methods. We first provide a complete classification of MTC. There exist many results for variants of this problem depending on the basic model for managing the cache and on the values of the problem parameters. The literature mainly considers two models (forced and optional saving of intermediate results into the cache) which will be detailed in the next section.

Next, for minimizing the final completion time (*makespan*),

1530-2075/11 $26.00 © 2011 IEEE
DOI 10.1109/IPDPS.2011.111

we prove for both the forced and optional models that MTC is NP-hard and that there are no constant-approximation algorithms unless $P = NP$, even in the simplest case where the cache capacity is equal to 1. Then, we derive new dynamic programming schemes for the two variants of the model. To our best knowledge, all existing studies concern the makespan minimization, which is a global objective by regarding all users as one. However, if we take care of the interest of each user, it is more reasonable to consider minimizing the sum of completion times of all the request sequences. Then, for this objective, we show that the problem is NP-hard as well for both models by a reduction from the problem with the makespan objective. We also propose polynomial time approximation algorithms and analyze their approximation ratio (which depends on the number of sequences).

Finally, we devise some heuristics for the model minimizing makespan, and present experimental results based on some random datas.

II. Related work

If there is only one user submitting the requests, it is reduced to the classical caching problem that has been well studied. Basically, the caching problem originates in an online setting, in which the requests arrive one by one and the server has to make an immediate decision upon arrival of a request without knowing the subsequent ones. There are rich results for online caching. However, it has also attracted much attention on the offline version where we know all the requests (in an order) in advance. In the following we review the results on the caching problem which give us the basic understanding of our problem.

A. Caching

Formally, the setup for the caching problem is as follows: we have a cache of capacity K and a set of tasks, $\mathcal{T} = \{T_i : 1 \leq i \leq N\}$. Given a list of requests $\gamma = \gamma_1 \cdot \gamma_2 \cdots \gamma_n$, where each request specifies a task to be processed, we must serve these requests in the arrival order. In other words, the precedence is a chain. For each task T_i, a processing time, denoted by p_i, and a size, denoted by s_i, are associated.

If some requested task T_i is already in the cache, then the processing time is zero, otherwise a processing time of p_i is incurred. At any time, if a recent task T_i is loaded into the cache then some older tasks may be evicted from the cache to satisfy the capacity constraint: $\sum_{T_i \in cache} s_i \leq K$. The goal is to decide which tasks are kept in the cache at each step so as to minimize the total processing time to serve the request chain γ.

In the caching problem, after serving some request which is not in the cache, if we are forced to put the corresponding task into the cache, we say it is *"Forced Caching Problem"*. Otherwise, we say it is *"Optional Caching Problem"*.

Now we are ready to briefly present the variants of caching starting from the general case.

General Model: $p_i \in Z^+$ and $s_i \in Z^+$ for each task T_i.

Offline: The problem is strongly NP-hard [11]. Bay-Noy *et al.* [6] proposed a 4-approximation algorithm for the forced caching problem, where the objective is, instead of minimizing C_{max}, to minimize the number of cache misses. Albers *et al.* [3] presented a $\frac{(1+\epsilon)}{\delta}$-approximation deterministic algorithm for the forced caching problem by increasing the cache size by $\delta \times (1 + \frac{1}{\sqrt{1+\epsilon}-1}) \times \max_i\{s_i\}$.

Online: For the forced case, Young [19], Cao and Irani [9] independently showed $\frac{K}{\min_i\{s_i\}}$-competitive algorithms, which are best possible(c.f. [15]). For the optional case, Albers [2] presented a $\frac{K}{\min_i\{s_i\}} + 1$-competitive algorithm, which is also best possible (c.f. [15]).

Next we introduce a special variant in which all tasks are of the same size in the cache. The problem becomes much easier and well solved.

Cost Model: $p_i \in Z^+$ and $s_i = 1$ for each task T_i.

Offline: The forced model is a special case of the *K-server* problem, and is thus polynomially solvable [10].

The optional model is equivalent to the *maximal weight interval scheduling* problem, which is also polynomially solvable [8]. The reduction is as follows: Given an instance of the optional caching problem, we have K identical machines. If there exist two indices x and y $(x < y)$ such that σ_x and σ_y request the same task, say T_i, and T_i is not requested between σ_x and σ_y, then in the viewpoint of the *maximal weight interval scheduling problem*, there is a task with start time x, end time y and weight p_i.

Online: For the forced case, we can get a best possible K-competitive online algorithm from [19] and [9]. For the optional case, it admits a best possible online algorithm with competitive ratio $(K + 1)$ [15].

Finally we present the simplest model.

Uniform Model: $p_i = 1$ and $s_i = 1$ for each task T_i.

Clearly this problem can be well solved as a special case of the cost model. We just want to remark that the forced model admits a simple optimal policy, called Belady's rule [7]: always evict the task whose next request is furthest in the future.

In addition, there are more variants, say the *Bit model* and *Fault model* in the literature. Interested readers may refer to [15] and [11].

B. Multi-threaded caching

In the multi-threaded caching problem, there is a set γ of Q finite request chains, where each chain γ^i, $1 \leq i \leq Q$,

consists of n_i requests belonging to $\mathcal{T}$, $\gamma^i = \gamma^i_1 \cdot \gamma^i_2 \cdots \gamma^i_{n_i}$. An example is illustrated in Fig. 1, in which the task set is $\mathcal{T} = \{T_1, T_2, T_3, T_4\}$ and there are three request chains.

As in the classical caching problem, each request must be served due to the constraint of chains. However, different from the classical problem, there are several requests available at a time and we have to decide which chain should be served next. It makes the problem much more difficult.

There is very little to know about the model. Feuerstein and Loma [12] investigated the online problem of multi-threaded caching for the forced uniform model. They showed a KQ-competitive online algorithm by employing the best possible online algorithm for the chains one by one. A lower bound of $K + 1 - \frac{1}{Q}$ was also derived, which is slightly better than the bound for one chain. For the very restricted case $K = 1$ and $Q = 2$, Alborzi et al. [4] showed a lower bound of 1.8, for any online algorithm, while the best known upper bound is still 2.

As already pointed out, to the best of our knowledge, existing work only considered makespan minimization. But for multiple chains, minimizing the total completion time of all chains is of great interest. This observation together with the little previous work on multi-threaded caching motivates us to tackle this hard problem from several ways. In the next sections we will provide a full picture on what we have achieved along this line.

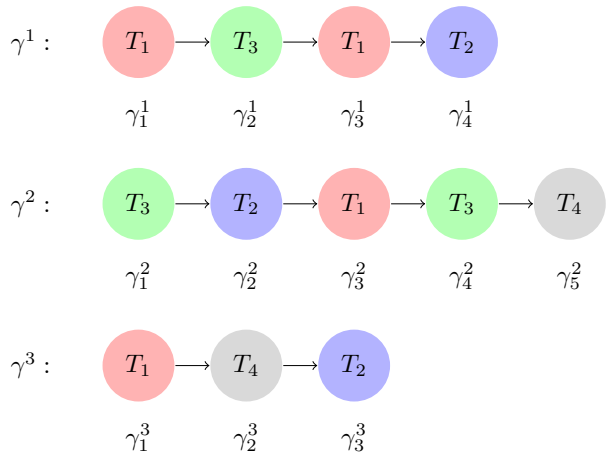

Fig. 1. An example of the multi-threaded caching problem

III. Framework and Definitions

A. Objective functions

In this paper we study two objective functions. First, we aim at minimizing the makespan. Second, we are interested in minimizing the total completion time of all chains. Formally, for each request chain γ^i, its completion time, denoted by C_i, is the completion time of its last request. Then $C_{\max} = \max_{1 \le i \le Q} C_i$, and $\sum C_i = \sum_{i=1}^{Q} C_i$.

B. Notation

For convenience, the problem is described by three fields in the sequel: the first tells that the model is a forced or optional multi-threaded caching problem (FMTC or OMTC); the second describes the model, e.g., *uniform*, *cost*, or *general* (the default term is general, which would be omitted if it is not ambiguous); the third represents the objective function (C_{max} or $\sum C_i$).

In the following, we will deal with Q request chains and the cache capacity is K. As an example, FMTC$||C_{max}$ is the case to serve Q request chains for the *forced general model* with the objective function C_{max} using a cache of capacity K.

IV. Minimizing C_{max}

One major goal of this paper is to assess the complexity analysis of the different versions of the Multi-threaded Caching Problem as described in Section III.

A. Complexity of FMTC|uniform|C_{max}

To prove this case is NP-hard, we use a reduction from the Shortest Common Supersequence problem (SCS) [13] and we first review this problem.

Given a finite alphabet $\mathcal{A}$, and two sequences $\Omega = \omega_1 \cdots \omega_m$ and $X = x_1 \cdots x_n$ with $\Omega, X \in \mathcal{A}^*$, Ω is a *supersequence* of X (and equivalently, X is a *subsequence* of Ω) if there exist some indices $1 \le f_1 < f_2 < \cdots < f_n \le m$ with $\omega_{f_j} = x_j$ $(1 \le j \le n)$.

Given a finite set of sequences $\mathbb{X} = X^1, X^2, \ldots, X^Q$, a *common supersequence of* $\mathbb{X}$ is a sequence Ω such that Ω is a supersequence of every sequence X^l $(1 \le l \le Q)$ in $\mathbb{X}$. Then, a shortest common supersequence (SCS) of $\mathbb{X}$ is a supersequence that has minimum length. Furthermore, a SCS problem is *perfect*, if for each sequence $X^i \in \mathbb{X}$, any two consecutive symbols in X^i are *not* the same. For instance, $X = \{abc, bbca\}$ is not perfect since there are two consecutive letters: b.

Maier [17] proved that the SCS problem is NP-hard. Timkovskii [18] treated the special case SCS(2,3), where each sequence in $\mathbb{X}$ contains exactly two symbols and each symbol of $\mathcal{A}$ appears totally in all the sequences of $\mathbb{X}$ at most three times. He proved that this problem is also NP-hard. The proof also indicates that SCS(2,3) is a *MAX SNP*-hard problem [16]. Arora et al. [5] proved that there is no *PTAS* for *MAX SNP*-hard problem, unless $P = NP$. Jiang and Li [16] showed that a constant approximation algorithm for the PERFECT SCS problem would result a *PTAS* for SCS(2,3).

To summarize, there is no constant approximation algorithm for the PERFECT SCS problem, unless $P = NP$.

In addition, Jiang and Li [16] also proved that if there exists an approximation algorithm for SCS with a perfor-

mance ratio of $\log^{\delta} Q$, where $\delta > 0$ is some constant, then $DTIME(2^{poly(\log Q)})$ contains NP.

In the following we prove that the PERFECT SCS problem is exactly FMTC|uniform|C_{max} when the cache capacity is equal to one.

Lemma 1. *A common supersequence of* PERFECT $\mathbb{X}$ *is a feasible schedule of* γ *when* $K = 1$*, and vice versa.*

Proof: To prove this, we just need to check the feasibility of the solution.

($\Rightarrow$) Let Ω be a supersequence of $\mathbb{X}$. Because of the construction, we know that each chain γ^i $(1 \le i \le Q)$ is also a subsequence of Ω, which means the precedence constraint relation in γ^i still holds in Ω.

Then, we schedule γ by loading the tasks into the cache according to Ω. So Ω is also a feasible schedule of γ.

($\Leftarrow$) Let σ be a feasible schedule of γ. For each chain γ^i, which is also perfect, we must serve it respecting the precedence constraints. Thus we can find some sub-indices of σ corresponding to γ^i, from which we can conclude that σ is a supersequence of γ^i $(1 \le i \le Q)$. ■

Theorem 2. *The* FMTC|*uniform*|C_{max} *problem is NP-hard, and admits no constant approximation algorithm unless* $P = NP$.

Proof: Due to Lemma 1, minimizing C_{max} is equivalent to finding a shortest common supersequence when $K = 1$. Thus, the problem is NP-hard. Moreover, it does not admit any constant-approximation algorithm as in the discussion above. ■

B. Dynamic programming for FMTC|cost|C_{max}

We provide a dynamic program for FMTC|cost|C_{max}, which is, in fact, also valid for the forced general model. Before going further, we introduce some necessary notations.

Definition 3. *Assume that each chain* γ^i *is of length* n_i $(1 \le i \le Q)$*. A* position *is a vector* $\overrightarrow{Y} = [y_1, y_2, \cdots, y_Q]$ *with* $0 \le y_i \le n_i$.

We say that we are at position $\overrightarrow{Y}$*, if we have already served all the requests from* γ^i_1 *up to and including* $\gamma^i_{y_i}$ *for the chain* γ^i*, i.e.,* $\gamma^i_{y_i+1}$ *is the first request which is not served in the chain* γ^i $(1 \le i \le Q)$.

In this dynamic programming, the objective function is $OPT(\overrightarrow{Y}|F)$, which represents the minimum processing time when we are at position $\overrightarrow{Y}$ with the set F of tasks stored in the cache.

Definition 4. *We say position* $\overrightarrow{Z}$ *is a* one-step-backward *of* $\overrightarrow{Y}$ *if there exists exactly one index* i *such that* $z_i = y_i - 1$*, and* $z_j = y_j \ \forall j \ne i$*. We denote this by* $\overrightarrow{Z} = \overrightarrow{Y} - \mathbb{1}_i$.

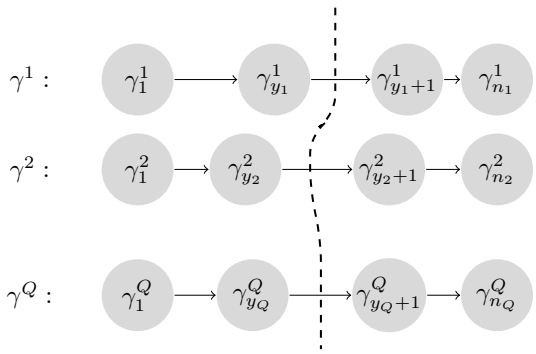

Fig. 2. Position $Y = [y_1, y_2, \ldots, y_Q]$

No doubt, each position $\overrightarrow{Y}$ has at most Q one-step-backward positions. For each task set F, after going backward for one step, we could replace at most one task in the cache, so there are at most N candidates for task set F.

Given two sets F and F', define $F \backslash F' := \{x \colon x \in F, x \notin F'\}$.

In the very beginning, $OPT([0, \ldots, 0]|\text{empty cache}) = 0$. Thus we can compute $OPT(\overrightarrow{Y}|F)$ with the dynamic program shown in Fig. 3.

Note that during the process, some configuration transformations may be infeasible. More precisely, the result of the request which is being served has to be stored into the cache, since it is a forced model. Hence we add a $+\infty$ to indicate such a case.

Finally, we can compute the optimum solution by checking all the reasonable cache states when arriving at the end of all the chains.

$$C_{max} = \min_{F\colon \sum_{T_i \in F} s_i \le K} \{\ OPT(\ [n_1, \ldots, n_Q] \mid F)\ \}.$$

Next, we analyze the complexity of the dynamic program. There are at most $\prod_{i=1}^{Q}(n_i + 1) \times \sum_{j=1}^{K} \binom{N}{j}$ objective functions, since we have $\prod_{i=1}^{Q}(n_i + 1)$ different positions, and at most $\sum_{j=1}^{K} \binom{N}{j}$ candidates for a task set F whose size is not greater than K. In addition, to compute each function, we need to consider at most $Q \times N$ possibilities.

Thus, we can find an optimum solution within the time

$$O\left(Q \times N \times \prod_{i=1}^{Q}(n_i + 1) \times \sum_{j=1}^{K} \binom{N}{j} \right).$$

Remarks: Notice that if both K and Q are constant, then this dynamic programming is polynomial. Since $\binom{N}{K} = \binom{N}{N-K}$, it is still polynomial, provided both $N - K$ and Q are constants.

$$OPT(\overrightarrow{Y}|F) = \min_{\substack{\overrightarrow{Z}:\ \overrightarrow{Z}=\overrightarrow{Y}-\mathbb{1}_i\ (1\le i\le Q)\\ F':\ |F\setminus F'|\le 1,\ |F'|\le K}} \begin{cases} OPT(\overrightarrow{Z}|F') & \text{if } \gamma^i_{y_i} \in F = F' \\ OPT(\overrightarrow{Z}|F') + p_j & \text{if } \gamma^i_{y_i} \in F\setminus F' \text{ and } \gamma^i_{y_i} = T_j \\ +\infty & \text{if } \gamma^i_{y_i} \notin F \end{cases}$$

Fig. 3. Dynamic programming for FMTC|cost|C_{max}

C. Approximation algorithm for FMTC|cost|C_{max}

We will present a straightforward approximation algorithm with a ratio of Q, based on the result that an optimum solution for the cost model with only one request chain can be found in polynomial time [10].

Algorithm 1: Concatenating

Input : A set of request chains
Output: A feasible schedule

1 **concatenate** all the chains to one chain, say $\gamma^1 \cdots \gamma^Q$
2 **return** an optimal schedule of the new request chain

Proposition 5. *Algorithm 1 is a Q-approximation algorithm, and the bound is tight.*

Proof: Denote by OPT the minimum processing time for all the chains, namely the makespan, and by OPT_i the minimum processing time for the i_{th} chain. No doubt, $OPT \geq OPT_i\ \forall i$.

Let C_A be the optimum solution for the new concatenated request chain, then

$$C_A \ \leq\ \sum_i OPT_i \ \leq\ Q \times OPT\ .$$

The following instance shows that the bound is tight. ■

Example 6. *Let σ be an arbitrary request chain whose optimal processing time, denoted by $OPT(\sigma)$, is large enough. Then we generate a number of new chains by inserting some unique tasks. More precisely, $\{\gamma^i := x_i \cdot \sigma\}$ (c.f. Fig. 4). Thus, $OPT_i = 1 + OPT(\sigma)$ $(1 \leq i \leq Q)$ and $C_A = Q \times OPT(\sigma) + Q$. While the optimal schedule can serve these new dummy requests first, then serve all the σ together, which gives that $OPT = Q + OPT(\sigma)$.*

As a result, we get

$$\frac{C_A}{OPT} \to Q \qquad (as \quad OPT(\sigma) \to +\infty)$$

D. Complexity of OMTC|uniform|C_{max}

Theorem 7. *The* OMTC|*uniform*|C_{max} *problem is NP-hard, and admits* no *constant approximation algorithm unless $P = NP$.*

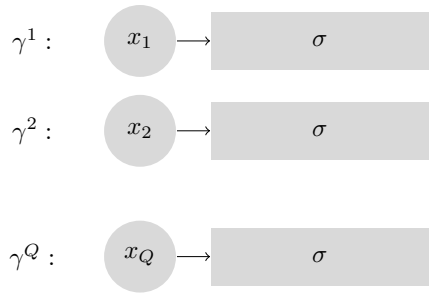

Fig. 4. Each request x_i $(1 \leq i \leq Q)$ appears only once, and all the request chains are the same except for the first request.

Proof: We get a reduction from FMTC|uniform|C_{max} for $K = 1$. For each request chain γ^i $(1 \leq i \leq Q)$, we use three copies, say γ'^i, γ'''^i and γ''''^i with $\gamma^i_j = \gamma'^i_j = \gamma'''^i_j = \gamma''''^i_j$, $\forall j$.

Clearly, although the number of request chains increases, the minimum processing time does *not* increase for FMTC|uniform|C_{max}.

Then we argue that, after generating three copies for each chain, both OMTC and FMTC have the same minimum processing time, if the cache capacity is one. That means each request has to be load into the cache to save the computation of the replicated tasks.

Considering a request, say γ'^i_j, which is the first request in some chain and is not in the cache at some step. The optimal solution has to put it into the cache. If so, it would save the total processing time by two since it has two other copies, after that we can re-put the result evicted into the cache if necessary, which incurs a cost of processing time one. As a result, we will save the total processing time at least by one. If we do not store γ'^i_j in the cache, we do not decrease the total processing time. ■

E. Dynamic programming for OMTC||C_{max}

The dynamic programming is almost the same as the one for the forced model, except that we don't need to load every task into the cache.

Note that if we don't load the result, which is being processed, into the cache, then the set F should not be changed. For more details, c.f. Fig. 5.

Thus we can compute an optimum solution by checking all

$$OPT(\overrightarrow{Y}|F) = \min_{\substack{\overrightarrow{Z}:\ \overrightarrow{Z}=\overrightarrow{Y}-\mathbb{1}_i\ (1\le i\le Q)\\ F':\ |F\setminus F'|\le 1,\ |F'|\le K}} \begin{cases} OPT(\overrightarrow{Z}|F') & \text{if } \gamma^i_{y_i} \in F = F' \\ OPT(\overrightarrow{Z}|F') + p_j & \text{if } \gamma^i_{y_i} \in F\setminus F' \text{ and } \gamma^i_{y_i} = T_j \\ OPT(\overrightarrow{Z}|F') + p_j & \text{if } \gamma^i_{y_i} \notin F, \text{ and } F = F' \\ +\infty & \text{otherwise} \end{cases}$$

Fig. 5. Dynamic programming for oMTC|cost|C_{max}

the reasonable cache states to the end.

$$C_{max} = \min_{F:\ \sum_{T_i\in F} s_i \le K} \{\ OPT(\ [n_1, \ldots, n_Q] \mid F)\ \}.$$

Next, we analysis the complexity of the dynamic programming. It is not hard to see that the arguments for the forced model are still valid here, so the running time of the dynamic programming is:

$$O\left(\ Q \times N \times \prod_{i=1}^{Q}(n_i+1) \times \sum_{j=1}^{K}\binom{N}{j}\ \right).$$

F. Approximation algorithm for oMTC|cost|C_{max}

The approximation algorithm is almost the same as the one for the forced model. We compute the optimum completion time for each chain using the approach of [8] respectively, then sum them up.

Proposition 8. *Algorithm 1 is a Q-approximation algorithm for* oMTC|*cost*|C_{max}*, and the bound is tight.*

V. Minimizing $\sum_i C_i$

A. Complexity of FMTC|uniform|$\sum C_i$

We will get a reduction from FMTC|uniform|C_{max}. Given an instance of $\mathcal{I}$ of FMTC|uniform|C_{max}, we transform it to an instance $\mathcal{I}'$ of FMTC|uniform|$\sum C_i$ by appending some requests to each request chain.

More precisely, let $\mathcal{I}$ be $\gamma = \{\gamma^i \colon 1 \le i \le Q\}$ with $|\gamma^i| = n_i$, we will append $M = Q \times \sum_i n_i$ new requests to each chain and get $\mathcal{I}'$,

$$\overline{\gamma} = \{\ \overline{\gamma}^i = \gamma^i \cdot \overline{T}_1 \cdots \overline{T}_M\ :\ 1 \le i \le Q\ \}.$$

Note that all the chains share a common "tail".

Lemma 9. *If $K = 1$, then the optimum solution schedules all the new requests, $\overline{T}_i$ $(1 \le i \le M)$, after the original ones, i.e., each new request needs to be processed exactly once.*

Proof: Let OPT be an optimal solution for the instance $\mathcal{I}'$. As a contradiction, we assume that there exists a new task which is processed at least twice. Note that if $\overline{T}_1$ is processed once, then all the new tasks $\overline{T}_i$ $(i \ge 2)$ could not be processed twice. Thus we only need to focus on $\overline{T}_1$.

Without loss of generality, we assume that $\overline{\gamma}^1$ is the first request chain finished by OPT. Denote by C_1 the completion time of $\overline{\gamma}^1$.

In the following we will argue two cases depending on whether or not all the new tasks $\overline{T}_1$ have been finished before C_1.

Case 1: All the new tasks $\overline{T}_1$ have already been finished before C_1.

We call a request *bad* if it incurs cost during the process. Consider two bad requests, say r_i and r_j, which ask for the same task $\overline{T}_1$, and there is no request asking for $\overline{T}_1$ between them. There must be some "old" tasks between them, otherwise OPT should serve both of them simultaneously.

Next we will change the schedule of OPT, which would result a better one.

Instead of serving r_i as OPT, we delay it and serve all the "old" tasks between r_i and r_j first, then serve r_i and r_j together.

Since the cache capacity is one, the schedule after interchanging of r_i and r_j is still feasible, and the completion time of all the request chains is decreased by one, which is a contradiction.

Case 2: There is still a request asking for $\overline{T}_1$ that needs to be served after C_1.

Let $\overline{\gamma}^Q$ be the last chain finished by OPT. Then its completion time is at least $C_{max} + 2M$, where C_{max} is the minimum processing time for $\mathcal{I}$, since each of the new tasks is processed at least twice.

In addition, the sum of the completion time of all the other chains is strictly greater than $M \times (Q-1)$. Thus, $OPT \ge C_{max} + M \times (Q+1)$.

On the other hand, we are seeking a trivial upper bound of OPT. Note that the cache capacity is 1, hence we can process all the original requests one by one without considering the order, which leads to a cost of at most $n = \sum_i n_i$. After that we serve all the new requests, which incur a cost of M. Clearly, the total cost of the trivial approach is an upper bound of OPT, therefore $(n+M) \times Q \ge OPT$.

As a result,

$$\begin{aligned} OPT &\geq C_{max} + M \times (Q+1) \\ &> M \times Q + M \\ &= (n+M) \times Q \\ &\geq OPT \ , \end{aligned}$$

a contradiction. ∎

Theorem 10. *The problem* FMTC|*uniform*|$\sum C_i$ *is NP-hard.*

Proof: Given an instance of $\mathcal{I}$ of FMTC|uniform|C_{max}, we transform it to an instance $\mathcal{I}'$ of FMTC|uniform|$\sum C_i$ as above.

Next we show that the optimum solution for $\mathcal{I}$ is C_{max} if and only if the one for $\mathcal{I}'$ is $(C_{max}+M) \times Q$.

($\Rightarrow$) We schedule all the requests due to the optimum solution for $\mathcal{I}$, then serve all the new requests, from which we get that $\sum C_i = (C_{max}+M) \times Q$.

($\Leftarrow$) Due to Lemma 9, all the chains are completed at the same time, namely $\frac{\sum C_i}{Q} = C_{max} + M$. There are exactly M new requests for each chain, so the optimum solution for $\mathcal{I}$ is C_{max}. ∎

B. Dynamic programming for FMTC$||\sum C_i$

The approach presented here is based on the one for minimizing C_{max}. Denote by $SUM(\overrightarrow{Y}|F)$ the minimum sum of the completion time when we are at position $\overrightarrow{Y}$ and cache is with the set F of tasks stored in the cache.

Definition 11. *For any position $\overrightarrow{Y}$, we say some chain is* active *if there are still some requests to be served for this chain, and the active number with respect to this position, denoted by $n_{\overrightarrow{Y}}$, is the number of active chains.*

Note that at each step, all the active chains would contribute to the objective function. C.f. Fig. 6 for more details.

We can compute the optimum solution by checking all the reasonable cache states until the last request.

$$\sum C_i = \min_{F:\ \sum_{T_i \in F} s_i \leq K} \{\ SUM(\ [n_1, \ldots, n_Q] \mid F)\ \}.$$

Again, the complexity of the dynamic programming is

$$O\left(Q \times N \times \prod_{i=1}^{Q} (n_i + 1) \times \sum_{j=1}^{K} \binom{N}{j} \right).$$

C. Approximation algorithm for FMTC|cost|$\sum C_i$

Before introducing the approximation algorithm, we prove a lemma first.

Lemma 12. *Given a sequence $\{x_i\}$ of positive number such that $x_1 \leq x_2 \leq \cdots \leq x_n$, we have:*

$$\sum_{i=1}^{n} (n+1-i) \times x_i \leq \frac{n+1}{2} \times \sum_{i=1}^{n} x_i \ ,$$

where the equality holds if and only if $x_1 = x_2 = \cdots = x_n$.

Proof: Define $y_m := \sum_{i=1}^{m} x_i$. Since $\{x_i\}$ is non-decreasing, we have

$$\begin{aligned} y_m &= \frac{m}{m+1} y_m + \frac{1}{m+1} y_m \\ &\leq \frac{m}{m+1}(x_1 + x_2 + \cdots + x_m) + \frac{1}{m+1}(m \times x_{m+1}) \\ &= \frac{m}{m+1}(x_1 + x_2 + \cdots + x_m + x_{m+1}) \\ &= \frac{m}{m+1} \times y_{m+1} \ . \end{aligned}$$

As a result,

$$\frac{y_1}{1} \leq \frac{y_2}{2} \leq \cdots \leq \frac{y_n}{n} \ .$$

And thus,

$$\sum_{i=1}^{n} (n+1-i) \times x_i = \sum_{i=1}^{n} y_i \leq \frac{y_n}{n} \sum_{i=1}^{n} i = \frac{n+1}{2} \times y_n \ .$$

∎

Algorithm 2:

Input : A set of request chains
Output: A feasible schedule

```
for i ← 1 to Q do
    compute the optimal processing time OPT_i for γ^i
sort the chains by increasing order of OPT_i ( w.l.o.g., assume that OPT_i < OPT_j if i < j )
concatenate these chains to one chain: γ^1 ··· γ^Q
return an optimal schedule of this new request chain
```

Proposition 13. *Algorithm 2 is a $\frac{Q+1}{2}$-approximation algorithm for* FMTC|*cost*|$\sum C_i$*, and the bound is tight.*

Proof: Denote by OPT the optimum value minimizing $\sum C_i$. Clearly, $OPT \geq \sum_i OPT_i$.

On the other hand, let C'_j be the completion time of the request chain γ^j in the new concatenated one. Thus $C'_j \leq \sum_{i=1}^{j} OPT_i$.

$$SUM(\overrightarrow{Y}|F) = \min_{\substack{\overrightarrow{Z}:\ \overrightarrow{Z}=\overrightarrow{Y}-\mathbb{1}_i\ (1\le i\le Q)\\ F':\ |F\setminus F'|\le 1,\ |F'|\le K}} \begin{cases} SUM(\overrightarrow{Z}|F') & \text{if } \gamma^i_{y_i} \in F = F' \\ SUM(\overrightarrow{Z}|F') + p_j \times n_{\overrightarrow{Z}} & \text{if } \gamma^i_{y_i} \in F\setminus F' \text{ and } \gamma^i_{y_i} = T_j \\ +\infty & \text{if } \gamma^i_{y_i} \notin F \end{cases}$$

Fig. 6. Dynamic programming for FMTC|cost|$\sum C_i$

Due to Lemma 12, we have

$$\begin{aligned}\sum_{j=1}^{Q} C'_i &\le \sum_{i=1}^{Q}(Q+1-i)\times OPT_i \\ &\le \frac{Q+1}{2}\times\sum_i OPT_i \\ &\le \frac{Q+1}{2}OPT.\end{aligned}$$

We show the bound is also tight by Example 6.

All the chains have the same optimal solution $OPT(\sigma)+1$, so the optimum solution for all the chains should be $Q\times(OPT(\sigma)+1)$.

However, Algorithm 2 just serves those chains one by one. Except the first chain, all the other chains would incur a cost of $OPT(\sigma)-k$ at least, since the tasks stored in the cache may reduce some cost. More precisely, we have

$$\begin{aligned}&C'_1 = OPT(\sigma)+1\\ &C'_j = C'_1 + (j-1)(OPT(\sigma)-k) \qquad (2\le j\le Q).\end{aligned}$$

As a result,

$$\frac{\sum_j C'_j}{OPT} \to \frac{Q+1}{2} \quad (\text{as } OPT(\sigma)\to+\infty).$$

■

D. Complexity of OMTC|uniform|$\sum C_i$

We will get a reduction from FMTC|uniform|$\sum C_i$.

Theorem 14. *The problem* OMTC|*uniform*|$\sum C_i$ *is NP-hard.*

Proof: Given an instance $\mathcal{I}$ of FMTC|uniform|$\sum C_i$, which consists of a set of request chains $\{\gamma^i : 1\le i\le Q\}$.

For each request chain, we generate a new one, where each request is duplicated twice,

$$\overline{\gamma}^i := \underbrace{\gamma^i_1\cdot\gamma^i_1\cdot\gamma^i_1}_{\text{3 requests for the same task}} \cdot \underbrace{\gamma^i_2\cdot\gamma^i_2\cdot\gamma^i_2}\cdots\underbrace{\gamma^i_{n_i}\cdot\gamma^i_{n_i}\cdot\gamma^i_{n_i}},$$

and thus $|\overline{\gamma}^i| = 3\times|\gamma^i|$.

It is not hard to check that $\sum C_i$ does not increase for the forced model. We will argue that for the optional model, $\sum C_i$ is the same as the forced model if $K=1$, which means that each request should be loaded into the cache.

Consider a request, say γ^i_j, which is the first request in some chain and is not in the cache at some step. The optimal solution has to put it into the cache. If so, it would save the total processing time by two since it has two other copies, after that we can re-put the result evicted into the cache if necessary, which incurs a processing time of one. As a result, we will save the total processing time at least by one. If we do not store γ^i_j in the cache, the total processing time cannot decrease. ■

E. Dynamic programming for OMTC||$\sum C_i$

The approach presented here is based on OMTC||C_{max}. Denote by $SUM(\overrightarrow{Y}|F)$ the minimum sum of the completion times when we are at position $\overrightarrow{Y}$ and the set F of tasks is currently stored in the cache. We use the same definition for "active".

Note that at each step, all the active chains would contribute to the objective function, c.f. Fig. 7 for more details.

We can compute an optimum solution by checking all the reasonable cache states when all the processing is done.

$$\sum C_i = \min_{F:\ \sum_{T_i\in F} s_i\le K}\{\ SUM(\ [n_1,\ldots,n_Q]\mid F)\ \}.$$

Again, the complexity of the dynamic programming is

$$O\left(\ Q\times N\times\prod_{i=1}^{Q}(n_i+1)\times\sum_{j=1}^{K}\binom{N}{j}\ \right).$$

F. Approximation algorithm for OMTC|cost|$\sum C_i$

The approach is the same as the one for FMTC|cost|$\sum C_i$, since the optional cost model for one request chain is also polynomially solvable.

Proposition 15. *Algorithm 2 is a* $\frac{Q+1}{2}$*-approximation algorithm for* OMTC|*cost*|$\sum C_i$*, and the bound is tight.*

VI. Heuristics for FMTC|cost|C_{max}

In this section, we propose two heuristics and compare them with the approximation algorithm considered in Section IV-C.

Note that an optimal schedule for one request chain can be found in polynomial time [10]. Thus, the main idea of the two heuristics is to merge the sequences into one to which the optimal algorithm on a single sequence can be applied.

$$SUM(\overrightarrow{Y}|F) = \min_{\substack{\overrightarrow{Z}:\ \overrightarrow{Z}=\overrightarrow{Y}-\mathbb{1}_i\ (1\le i\le Q)\\ F':\ |F\setminus F'|\le 1,\ |F'|\le K}} \begin{cases} SUM(\overrightarrow{Z}|F') & \text{if } \gamma^i_{y_i} \in F = F' \\ SUM(\overrightarrow{Z}|F') + p_j \times n_{\overrightarrow{Z}} & \text{if } \gamma^i_{y_i} \in F\setminus F' \text{ and } \gamma^i_{y_i} = T_j \\ SUM(\overrightarrow{Z}|F') + p_j \times n_{\overrightarrow{Z}} & \text{if } \gamma^i_{y_i} \notin F, \text{ and } F = F' \\ +\infty & \text{otherwise} \end{cases}$$

Fig. 7. Dynamic programming for oMTC|cost| $\sum C_i$

A. Randomized merging

The first approach is rather simple, just merge all the request chains into one chain according to the precedence constraints.

More precisely, let RANDOM(γ^i, γ^j) be the function that returns randomly a supersequence of two request chains γ^i and γ^j. The pseudocode is as follows:

Algorithm 3: Randomized merging

Input : A set of request chains, γ^i, $1 \le i \le Q$.
Output: A feasible schedule

Initialization: $\gamma \leftarrow \emptyset$
for $i \leftarrow 1$ **to** Q **do**
  $\gamma \leftarrow \text{RANDOM}(\gamma, \gamma^i)$
return an optimal schedule of γ

B. SCS merging

The second approach is a little bit more involved. For any two request chains, since their shortest common supersequence (SCS) could be found in polynomial time [16], we devise a heuristic via SCS.

More precisely, let SCS(γ^i, γ^j) be the function that returns a shortest common supersequence of two request chains γ^i and γ^j. The pseudocode is as follows:

Algorithm 4: SCS merging

Input : A set of request chains, γ^i, $1 \le i \le Q$.
Output: A feasible schedule

Initialization: $\gamma \leftarrow \emptyset$
for $i \leftarrow 1$ **to** Q **do**
  $\gamma \leftarrow \text{SCS}(\gamma, \gamma^i)$
return an optimal schedule of γ

C. Experimental results

We have implemented Algorithm 1 (Concatenating), Algorithm 3 (Random merging) and Algorithm 4 (SCS merging) with C++, based on an open source graph library LEMON [1].

We present below a preliminary set of experiments with small instances. The number of different tasks is five and our cache capacity is two. The cost for each task is selected independently and uniformly at random from the interval [1, 1000]. There are three or four chains in each instance, where each request is chosen independently and uniformly at random from the task set.[1]

Part of the results are shown in Table I, from which we can see that Algorithm 1 (Concatenating) performs worse in most of time.

Concatenating	Randomized	SCS	OPT
197	**167**	199	159
234	216	**182**	182
206	216	**206**	183
284	**238**	274	238
338	243	**241**	241
1235	859	**849**	849
1316	**961**	**961**	961
1486	1340	**1330**	1193
1348	**1347**	1685	1253
2125	2125	**1642**	1570
3183	**2923**	3183	2200
2381	2381	**2235**	2235
2985	**2361**	2405	2361
4343	3525	**3394**	3108
3573	**3479**	4437	3479
5433	**4039**	4989	4039
37267	29744	**25764**	25097
27402	27412	31939	27402

TABLE I
EXPERIMENTAL RESULTS. BEST HEURISTIC RESULTS ARE PRINTED IN BOLD.

VII. Concluding remarks

This paper gives the first attempt on the offline multi-threaded caching problem. There are many open questions left. It is shown that if the number Q of the chains is not fixed, even with $K = 1$, the uniform model cannot be approximated with a constant ratio assuming that $P \neq NP$. On the other hand, if both Q and K are fixed, even the general model is polynomially solvable. The most interesting question in complexity is to seek an NP-hardness proof or a polynomial optimal algorithm for the case where Q ($Q \ge 2$) is a constant. It is also challenging to improve the approximation algorithms for the general model.

Note that in [12] there is a huge gap between the lower and upper bounds for online algorithms of the uniform model. It seems possible to improve the lower bound. However, to

[1]For more details of the source code, welcome to contact us.

design an online algorithm with a competitive ratio better than the trivial bound is not a piece of cake!

In addition to the theoretical work, it is worthy implementing the known approximation algorithms with real data instead of random data.

Acknowledgements. Partial support for Haifeng Xu was provided by National Nature Science Foundation of China 11071215 and by China Scholarship Council 2007101158.

The authors are grateful to the anonymous referees for their helpful comments and useful references, which have contributed to improving the presentation.

REFERENCES

[1] http://lemon.cs.elte.hu/trac/lemon.

[2] S. Albers. New results on web caching with request reordering. *Algorithmica*, 58:461–477, 2010.

[3] S. Albers, S. Arora, and S. Khanna. Page replacement for general caching problems. In *Proceedings of the tenth annual ACM-SIAM symposium on discrete algorithms*, pages 31–40. Society for Industrial and Applied Mathematics, 1999.

[4] H. Alborzi, E. Torng, P. Uthaisombut, and S. Wagner. The k-client problem. In *Proceedings of the eighth annual ACM-SIAM symposium on discrete algorithms*, pages 73–82. Society for Industrial and Applied Mathematics, 1997.

[5] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[6] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.

[7] L.A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[8] K.I. Bouzina and H. Emmons. Interval scheduling on identical machines. *Journal of Global Optimization*, 9(3):379–393, 1996.

[9] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the USENIX symposium on internet technologies and systems*, pages 193–206. Usenix Association, 1997.

[10] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. In *Proceedings of the first annual ACM-SIAM symposium on discrete algorithms*, pages 291–300. Society for Industrial and Applied Mathematics, 1990.

[11] M. Chrobak, G. Woeginger, K. Makino, and H. Xu. Caching is hard - Even in the fault model. In *The 18th Annual European Symposium on Algorithms*, 2010.

[12] E. Feuerstein and AS de Loma. On-line multi-threaded paging. *Algorithmica*, 32(1):36–60, 2002.

[13] M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness.* W.H. Freeman, 1979.

[14] C. Grasland, N. Lambert, and J. Vincent. Regional disparities and cohesion: What strategies for the future. Technical report, the European Parliament's committee on Regional, 2007.

[15] S. Irani. Page replacement with multi-size pages and applications to web caching. *Algorithmica*, 33(3):384–409, 2002.

[16] T. Jiang and M. Li. On the approximation of shortest common supersequencesand longest common subsequences. *SIAM Journal on Computing*, 24(5):1122–1139, 1995.

[17] D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25(2):322–336, 1978.

[18] V.G. Timkovskii. Complexity of common subsequence and supersequence problems and related problems. *Cybernetics and Systems Analysis*, 25(5):565–580, 1989.

[19] N.E. Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002.

[20] M. Yuan. Use of a three-domain repesentation to enhance GIS support for complex spatiotemporal queries. *Transactions in GIS*, 3(2):137–159, 1999.