An optimal batch scheduling algorithm for OBS networks

Gustavo B. Figueiredo, Eduardo C. Xavier, Nelson L. S. da Fonseca
Institute of Computing
University of Campinas – UNICAMP
P.O. Box 6176 – 13.084-971
Campinas – SP – Brazil
Email: {gustavo, ecx, nfonseca} @ic.unicamp.br

Abstract— This paper introduces an optimal batch scheduling algorithm for the scheduling of batches of bursts in optical burst switching networks. The algorithm, called BATCHOPT, considers both the requests being processed in the current batch and the requests previously scheduled in the search for an optimal solution. Moreover, an extended version of the JET reservation protocol is proposed for efficiently handling batches of bursts. Results obtained via simulation show that the BATCHOPT algorithm produces good performance when compared to other proposed existing algorithms.

I. INTRODUCTION

In Optical Burst Switching (OBS) networks, packets are aggregated at edge nodes to form transmission units called bursts. A control packet is transmitted out-of-band, ahead of the burst so that resources can be reserved to the data burst. The control packet carries information about the burst such as the time interval that separates its arrival to the arrival of the burst itself, called offset time, as well as the burst length. At each node in the network core the scheduling mechanism reserves bandwidth of the node output channels for the burst based on the information carried in its control packet. At the network boarder, it is not necessary to wait for the confirmation of a resource reservation request to start transmitting a burst. However, if resources cannot be reserved at a core node, the burst is discarded.

Just-Enough-Time (JET) [1] is a prevailing protocol for resource reservation in OBS networks. JET reserves the channel for the duration of the transmission of a burst, starting at the burst expected arrival time given by the offset time minus the burst processing time. If reservation is successful, a new offset time is calculated and forwarded to the next hop in the route.

Since bursts have different offset time, they may arrive in a different order than that of their control packets yielding to the fragmentation of the output channels of a core node. The pattern of occupation of output channels typically alternates between periods of occupancy and idle periods which are called void intervals. These void intervals can be used for accommodating the transmission of a new burst. A Void interval, $I_j$, represented by its starting, $s_j$, and its ending time, $e_j$, can be allocated to a burst with arriving time, $t'$, and departing time, $t''$ if and only if $s_j \leq t' \leq e_j \geq t''$.

Most of existing scheduling algorithms schedule bursts individually, in a greedy fashion [2]–[6]. However, this approach can lead to the blocking of incoming request which would not be blocked if previous knowledge of its arrival exists. One way to ameliorate such blocking is to gather reservation requests in an interval and schedule the bursts in the formed batch with full knowledge of information about the batches. This can be accomplished by having a bin to store all control packets arrived during an interval so that their information can be accounted for in the computation of a reservation scheme. The objective is to maximize the number of burst transmitted, i.e, to minimize the loss of burst. The occupancy of the output channels by the bursts can be modeled as an interval graph and the solution of the burst assignment is given by the solution of a problem of coloring an interval graph.

Previous work [7], [8] on batch scheduling in OBS networks assumed a formulation of a job scheduling problem with non-identical machines which leads to an NP-hard problem. Therefore, only heuristic solutions were proposed. In this paper, we show how to formulate the problem of batch scheduling in OBS networks as a job scheduling problem with identical machines which yields to an optimal algorithm with polynomial time complexity. Results derived via simulation show that besides the low computational complexity, the algorithm proposed gives lower blocking probability than previously proposed ones. Moreover, we extend the JET protocol to efficiently operate with batch scheduling. Such extension can be introduced in networks employing the JET protocol since its operation is exactly the JET one when scheduling individual bursts.

This paper is organized as follows. Section II presents some concepts related to batch scheduling. Section III reviews related work. Section IV presents the new algorithm called BATCHOPT. Section V presents an extended version of the JET protocol. Section VI presents some numerical examples and Section VII draws some conclusions.

II. BACKGROUND

Let $G = (V, E)$ be a graph, where $V(G)$ represents the set of vertices of $G$ and $E(G)$ the set of edges. Given a vertex $u \in V(G)$, we define the adjacency/neighborhood of $u$ by $\text{Adj}(u) = \{v \in V(G); (u, v) \in E(G)\}$. A subgraph $H$ of $G$ is a graph such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. The degree of vertex $u$ in some graph $G$, denoted by $d(u|G)$, is the size of
the set $\text{Adj}(u)$. A subgraph $H$ of $G$ is induced by $V(H)$ if for every pair $u, v \in V(H)$ we have that $(u, v) \in E(H)$ if and only if $(u, v) \in E(G)$.

A clique of a graph $G$ is a set $C \subseteq V(G)$ such that $\forall u, v \in C; (u, v) \in E(G)$. A clique $C$ is maximal if there is no other clique $C'$ in $G$ such that $C \subset C'$.

A graph $G$ is called an interval graph if there is a correspondence/bijection between the set of vertices and a set of intervals on the real line, such that there is an edge between two vertices if and only if the correspondent intervals intersect, i.e., $(u, v) \in E(G) \Leftrightarrow I_u \cap I_v \neq \emptyset$. Interval graphs have several properties that can be used to solve numerous problems in combinatorics. Among these properties, it is well known that interval graphs can be recognizable and colorable in linear time.

A typical use of interval graphs is in finding solutions for the following job scheduling problem. Let $I = \{J_1 = (s_1, e_1, w_1), \ldots, J_n = (s_n, e_n, w_n)\}$ be a list of $n$ jobs, where $(s_i, e_i)$ is the start and finishing/end time of the job $J_i$, and $w_i$ its weight. Moreover, there are $k$ machines with the same processing capacity. At first, all machines are free starting at time $0$. The problem is to select a sub-list $I' \subseteq I$ of jobs with maximum total weight such that there is a feasible scheduling of $I'$ on the $k$ machines, i.e., no two jobs allocated to a machine can intersect their processing intervals.

This problem is called job scheduling problem with identical machines if there is no restriction on which machine can process each job, otherwise it is called job scheduling problem with non-identical machines.

The OBS scheduling problem can be formulated as a job scheduling problem where each channel of an OBS network is a machine and each reservation request corresponds to a job. Formally in the OBS scheduling problem there are $k$ channels, and a list $I = \{J_1 = (s_1, e_1, w_1), \ldots, J_n = (s_n, e_n, w_n)\}$ of $n$ reservation requests, where $(s_i, e_i)$ is the start and ending time of the request $J_i$, and $w_i$ its weight. There is also a list $S$ of requests that have already been allocated to the channels. The list $S$ corresponds to requests accounted for in previous batches. The problem is to find a subset of requests of $I$ with maximum total weight that can be allocated on the $k$ channels. Two different requests cannot be allocated to the same channel if they intersect (here it doesn’t matter if the request was previously allocated or if it is a new one being processed).

In section III, the OBS scheduling problem is formulated as a job scheduling problem with non-identical machines which is an NP-hard problem [7]. In section IV, the solution of the OBS scheduling problem using a polynomial time algorithm for the job scheduling problem with identical machines is presented.

### III. RELATED WORK

In [7], the OBS scheduling problem was solved by a formulation of a job scheduling problem with non-identical machines. In such formulation, the set $S$ of previously allocated requests/jobs corresponds to intervals which cannot accept a request. So, if some request $r$ in the incoming list $I$ intersects with a request $r' \in S$ allocated in machine $m$, then $r$ cannot be allocated on machine $m$. Since some requests cannot be processed in some channels/machines, the OBS scheduling problem is solved as a job scheduling problem with non-identical machines [7].

In [9], it is presented an optimal algorithm for the job scheduling problem with non-identical machines. The computational complexity of this algorithm is $O(n^{k+1})$, which is prohibitive for the solution of the batch scheduling in OBS networks [7], since it is exponential in the number of channels ($k$). In [7], four heuristics solutions for the OBS scheduling problem are presented. The complexity time of these heuristics is $O(nk \log(N))$, where $n$ is the number of requests being processed, $k$ is the number of channels and $N$ is the number of previously allocated requests. Next, brief descriptions of these heuristics are provided. In all of these heuristics the requests are modeled as an interval graph $G$.

1. **Smallest Vertex Ordering (SLV):** The vertices $v_1, \ldots, v_n$ of $G$ are considered to be ordered in a smallest last fashion if $v_i$ has the smallest degree in the subgraph induced by the vertices $v_1, \ldots, v_i$ (this way $v_n$ is the vertex with the smallest degree in $G$). In SLV, the requests are allocated in the smallest last order, i.e., the first request allocated is the one corresponding to vertex $v_1$. If $v_1$ cannot be allocated in none of the channels, it is then discarded, and the process is repeated for all other $v_i$’s.

The idea behind the SLV algorithm is that choosing the vertices with the largest degree, a smaller number of channels is used, however, SLV can generate great losses of requests [7].

![Fig. 1. Problem with heuristics.](image)
graph, a clique of size $M$ such that $k < M$, then necessarily $M - k$ requests are going to be discarded. The MCF heuristic computes the order in which the requests are going to be processed and also the requests that will be discarded. For that, the heuristic computes all maximal cliques of $G$ and then sorts them in an increasing order to time. Let $\{C_1, C_2, \ldots, C_m\}$ be the set of maximal cliques of $G$ ordered such that $C_i < C_j$ for $i < j$ (i.e., there is a request in $C_i$ that has smaller initial time than each one of the requests in $C_j$). The algorithm first process the requests of clique $C_1$, then $C_2$ and so on. If the size of any $C_j$ exceeds the number of channels $k$, then $|C_j| - k$ requests with smallest finishing time are discarded.

This heuristic can also produce poor results. Under MCF, the first clique to be processed (see Figure 1) is the clique with vertices “A” and “B”. Since there is only one channel, request “B” is discarded and “A” is allocated on the channel. All remaining requests are then discarded.

**Smallest Start-time First Ordering (SSF):** In this heuristic, requests are ordered according to their starting time, and then processed in this order. Poor results can also be produced as shown by the example of Figure 1. The first request “A” is processed and all other requests are discarded.

**Largest Interval First Ordering (LIF):** In this heuristics, the requests are ordered in non-increasing order of their lengths which is the difference between the finishing and start time of the request. The requests are then processed in this order. The example in Figure 1 also shows the poor behavior of this heuristic. When “A” is allocated, all other requests are discarded. It is interesting to note that even when the total length of the other requests is greater than the size of “A”, all other requests are discarded.

The main problem with these proposed approaches is that they are heuristics, since there is no guarantee that the best solutions will be computed. This way, these approaches depends on the structure of the generated interval graphs; for some of them good solutions are found, while for others the solutions are of poor quality.

**IV. THE BATCHOPT SCHEDULING ALGORITHM**

The formulation of the batch scheduling problem in OBS networks provided in [7] assumes that if a request intersects with another request previously allocated to a channel, then the new request cannot be allocated to that channel. This assumption leads to the formulation of the problem as a job scheduling with non-identical machines, which is a NP-hard problem. Several heuristics are then proposed to solve this NP-hard problem [7].

In this work we solve the batch scheduling problem as a job scheduling problem with identical machines. For this, the proposed algorithm considers the set $S$, of previously allocated requests, together with a new batch of requests $I$, in such a way that the requests in $S$ remain allocated. This leads to an optimal algorithm called BATCHOPT which has complexity time $O(n^2 \log n)$, where $n$ is the number of requests being processed (the ones in $I \cup S$).

We present an algorithm for the following problem: Let $I = \{J_1 = (s_1, e_1, w_1), \ldots, J_n = (s_n, e_n, w_n)\}$ be a list of jobs, where $(s_i, e_i)$ is the start and end time of job $J_i$, and $w_i$ its weight. The algorithm also receives $k$ identical machines and a list $S$ of jobs already allocated in the machines. The problem is to find a sublist $I' \subseteq I$ of jobs of maximum total weight that can be allocated on the $k$ machines in a feasible schedule, i.e., no two different jobs in $I' \cup S$ that were allocated to the same machine can intersect. In [9], an optimal algorithm (denoted by AS) for the job scheduling problem with identical machines was introduced and it will be described next. Then we present our algorithm which extends the algorithm AS to include the restriction that there are some jobs $S$ that are already allocated.

Let $I = \{J_1, \ldots, J_n\}$ be a list of jobs, each $J_i$ with weight $w_i$ and interval $(s_i, e_i)$. The algorithm AS constructs an interval graph $G$ considering these jobs. The algorithm then computes all maximal cliques of this graph and then orders them in an increasing order of starting time. Figure 2 shows an example of this first construction.

**Fig. 2.** Requests are on the left. On the right is the corresponding interval graph and its maximal cliques numbered according to the ordering of time.

Let $C_1, \ldots, C_r$ be the computed maximal cliques ordered according to an increasing time criterion. The next step is to construct a flow-graph $G'$ as follows: first create a vertex $v_0$ and for each clique $C_j$ ($j = 1, \ldots, r$) create a vertex $v_j$, then, create directed arcs $(v_j, v_{j+1})$ for each $C_j$, with cost 0 and infinity capacity. Let $M$ be the maximum size of a clique among the cliques $C_1, \ldots, C_r$. For each clique $C_j$, create a directed arc $(v_{j-1}, v_j)$ with cost 0 and capacity equal to $M - |C_j|$ that represents a dummy job. For each job $J_i$ that belongs to cliques $C_j, \ldots, C_{j+t}$, create a directed arc $(v_{j-1}, v_{j+t})$ with capacity 1 and cost $w_i$. This arc represents all cliques for which $J_i$ belongs (from $C_j$ to $C_{j+t}$). The aim is to find a flow from $v_0$ to $v_r$ in $G'$ of $(M - k)$ units and with minimum cost. Given a solution to this minimum cost flow problem, the used arcs in the flow corresponds to jobs that have to be discarded. All arcs with zero flow in the solution corresponds to jobs that must be allocated.

Figure 3 provides an example of the construction of the flow-graph $G'$ for the interval graph $G$ of Figure 2. The arcs going through $v_0$ in the direction of $v_r$ represent the requests.

**Theorem 4.1:** The algorithm AS [9] solves optimally the job scheduling problem with identical machines.

**Proof:** Please, see reference [9] for a proof.

The set $S$ of previously allocated requests can be easily
introduced in the problem detailed above by assuming for each request \( J_i \in S \) that its cost is infinity. In this way, it is guaranteed that the already allocated requests will remain allocated in the final solution. The proposed algorithm called BATCHOPT considers both requests to be allocated and already allocated in its formulation and provides an optimal solution as in [9]. Moreover, it has complexity time \( O(n^2 \log n) \), where \( n = |S \cup I| \) which depends only on the number of requests being processed. The pseudo-code of the BATCHOPT algorithm is presented next (Algorithm 1).

**Algorithm 1 BATCHOPT**

**Input**

\( k \) channels of a node \( i \), a set \( I \) of requests and a set \( S \) of previously allocated requests that intersects with some of the requests in \( I \).

**Output**

A subset \( I' \subseteq I \) of maximum weight and with a feasible schedule.

**BATCHOPT**

1. Set infinity weights to each request in \( S \).
2. Construct an interval graph \( G \) representing \( I \cup S \).
3. Order the maximal cliques of \( G \) according to time.
4. Construct a flow graph \( G' \) as shown in section IV.
5. Compute a flow of size \( M - k \) and minimum cost.
6. Allocate all requests that corresponds to arcs in \( G' \) that have flow equal to zero.

**Theorem 4.2:** The BATCHOPT algorithm optimally solves the OBS scheduling problem. Moreover, each request that is already being transmitted by the node remains allocated to its previously allocated channel.

**Proof:** First we prove the optimality of the algorithm. Since we give a weight equal to infinity to each request in \( S \), and the algorithm computes a minimum cost flow, the algorithm will not use any of the arcs corresponding to requests in \( S \). This way, the requests in \( S \) will necessarily remain allocated. Besides that, from the result of Theorem 4.1, we know that among the requests in \( I \), the algorithm allocates a sublist \( I' \) of maximum weight.

Now we show that we do not need to change the channels assigned to requests that are already being transmitted. Suppose the algorithm generates the scheduled starting in some time \( t \), with a set \( S \) of requests already allocated. Let \( S' \subseteq S \) be the set of requests with starting time less than \( t \). This means that each request in \( S' \) is already being transmitted, and we can’t change its allocated channel. But notice that for the requests in \( S \setminus S' \) we can change their channel without problems. The BATCHOPT algorithm selected a set \( I' \) of new requests such for any time \( t \leq t' \) there are at most \( k \) requests from \( I' \cup S \) that intersects with \( t' \). We can sort the requests in \( I' \cup S \) by their starting times and allocate each request in this order to an available channel. This generates a feasible schedule since at any time \( t' \) there are at most \( k \) requests intersecting it, and so there is an available channel each time we assign a request. Notice that the requests in \( S \) where previously allocated in a feasible schedule. Since the requests in \( S' \) have the smallest starting times among the requests in \( S \cup I' \), they are going to be processed first and so we can allocate each request in \( S' \) to its previously allocated channel.

**V. MODIFIED JET PROTOCOL**

Previously proposed batch scheduling algorithms collect requests during a time window of fixed duration, and then employ different heuristics to schedule the requests. However, such approach leads to unnecessary loss of bursts. Figure 4 illustrates this type of loss. In this figure, two control packets arrive during the fixed size time window. Burst \( A \) arrive at time \( t_a \) and the burst \( B \) arrive at time \( t_B \). While burst \( A \) will be processed by the batch scheduling policy and will be potentially transmitted, burst \( B \) won’t be considered by the scheduling policy and will be lost because it arrived during the fixed time window. Such situation exists because the batch scheduling policy ignores the arrival time of the bursts by fixing the duration of the window to collect the requests, called acceptance window (\( \Delta \)).

To eliminate such kind of loss, it is necessary to adopt acceptance windows with variable durations as well as a threshold to delimit the time the processing of the batch of requests should start. This threshold called processing threshold should be determined so that no data burst arrived in the acceptance window should be lost. In the proposed scheme, the acceptance window should be added to the offset time, as illustrated in Figure 5 and this new interval accounted for by the reservation protocol. Such extension of the JET protocol was named JET-\( \Delta \). This extension can be used by batch scheduling algorithms as well as by greedy scheduling [2]–[6] (by making \( \Delta = 0 \)).

![Fig. 3. An example of construction of a flow graph.](image)

![Fig. 4. Example of data burst loss due to the use of fixed size acceptance window.](image)
batch processing time, and $t_r$ the arrival time of control packet $r$ with offset time. $T_j^i$ at node $i$ in its way to node $j$. $\delta$ is of the order of microseconds whereas the acceptance window of the order of miliseconds [10].

When the processing threshold is reached, the batch is processed and all bursts going to the same destination are grouped and the information about the new batch go into the same control packet to decrease the overhead imposed [11].

$$\Delta = D - T_s^j / H$$  \hspace{1cm} (1)

An alternative way to reduce the introduction of delay is to control the burst size at the assembly time as in [12] and [13].

VI. PERFORMANCE EVALUATION

The schemes proposed in this paper were evaluated via simulation using the OB2S tool [14]. In each simulation, 10,000 reservation requests were generated. The replication method was used to generated confidence intervals with 95% confidence level; twenty replications were pursued to generate mean values. The NSFNet and the Abiliene topologies were used (Figure 6). However, due to space limitation, only results for the NSF topology are shown. Similar conclusions can be derived by results found when using the Abiliene topology. We consider that each fiber has 16 wavelength channels of 2.5 Gbps. The time taken to process a control packet is $50 \mu s$.

Both source and destinations of a request are taken from a uniform distribution involving all nodes. The burst generation rate is Poissonic and the size of the burst are exponentially distributed. The load varied from 200 to 2000 Erlangs using increments of 200 Erlangs.

In order to produce a fair comparison, all the algorithms [7] used the JET-\(\Delta\) protocol. First, the impact of the duration of the acceptance window (\(\Delta\)) on the algorithms was evaluated. For that, all the requests have the same weight. Results shown in Figure 7 were derived using a load of 1,000 Erlangs. This figure displays the blocking probability as a function of the duration of $\Delta$. It can be seen that the algorithms LIF, MCF, SSF and SLV are not sensitive to the window duration. Regardless of the increase of the window size, these policies were not able to reduce the blocking probability. Moreover, BATCHOPT takes advantage of the knowledge about the arrival time of the bursts to produce the lowest blocking probability. Besides that, the almost constant value produced by BATCHOPT evinces that it adjusts the size of the acceptance window considering only the pattern of arrival of bursts.

Figure 8 shows the blocking probability as a function of the load when requests have the same weight. As the load increases, so does the number of bursts lost due to the increase of the size of the maximum cliques of the interval graphs. However, the loss produced by the BATCHOPT is the lowest one. As requests have the same weight, the set of requests selected by BATCHOPT has always the highest cardinality and, thus, the number of bursts lost is the lowest one.

Simulation with requests associated to different Classes of
Blocking probability as a function of the load when burst have the same weight

Quality of Service were also conducted. In these simulations, five classes were considered; the class associated to a request was randomly picked from a uniform distribution. The weight of the classes involved follow the following pattern: $w_1 = 1, w_2 = 2, w_3 = 4, w_4 = 8, w_5 = 16$, where $w_i$ is the weight of class $i$. Figure 9 shows the distribution of loss per class. Note that BATCHOPT provides the lowest blocking probabilities for the high priority classes, concentrating most of the losses in the classes with lower priority. In other words, BATCHOPT is capable to provide differentiated services to distinct classes of service. This does not happen with the other policies which produce an "almost uniform" distribution of losses.

Fig. 8. Overall blocking probability as a function of the load when burst have the same weight

Fig. 9. Blocking probability per class as a function of the load

VII. CONCLUSIONS

This paper introduced an optimal algorithm that considered not only the unprocessed requests but also the allocation already made in the problem formulation. This led to an optimal algorithm with polynomial time which differ from previous others based on heuristics. Moreover, the paper extended the JET reservation protocol to consider the scheduling of bursts and yet being compatible to the original JET. Results derived via simulation showed that the proposed scheduling algorithm outperforms previously proposed ones. Results using trace driven simulation have been derived and preliminary results reinforce the findings shown in this paper.

REFERENCES