On-line and Off-line Call-Scheduling in Stars and Trees

18 June 1997

Thomas Erlebach

TU München

Klaus Jansen

Universität Trier

- Introduction
- Results for Stars
- Results for Trees

Problem: Call-Scheduling

Given:

- undirected tree $T = (V, E)$ with edge capacities $c(e) = 1$
- list $L = (r_1, r_2, \ldots, r_k)$ of call requests

Each call request r is given by communication endpoints $u_r, v_r \in V$, a bandwidth requirement $b_r \in (0; 1]$, and a duration $d_r \in \mathbb{N}$.

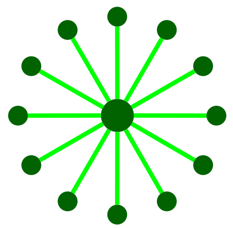
Goal:

Assign time steps to call requests, i.e., find $S : L \to \mathbb{N}_0, r \mapsto t_r$ (a schedule) such that the sum of bandwidth requirements of simultaneously active calls using an edge is at most 1.

Minimize the schedule length: $|S| = \max_{r \in L} t_r + d_r$

Background and Assumptions

- ATM networks, virtual circuits, guaranteed QoS
- Movie On Demand, Multimedia, Video Conferences
- single ATM switch $\triangleq$ star (i.e., tree of height one):



- no node capacities, precedence constraints, release times
- special case: unit durations
- off-line vs. on-line: call durations known/unknown
- off-line problem is NP-hard except for trees of bounded degree and calls with unit durations and unit bandwidths

Algorithm: List-Scheduling

Input: tree T and list L of call requests

$t \leftarrow 0$;

while not all calls have been scheduled **do**

  if $\exists$ call r in L such that bandwidth b_r is available

    along the undirected path from u_r to v_r in T at time t

  then schedule the first such call r at time t and remove it from L;

  else $t \leftarrow$ earliest time when one of the active calls finishes;

  fi

od

Remark: List-Scheduling is an on-line algorithm!

Observation: List-Scheduling Property

If a call r is scheduled at time t_r in a list-schedule, in every time step prior to t_r there exists at least one edge on the path from u_r to v_r that has less than b_r bandwidth available.

Definitions:

$\mathrm{LS} = \mathrm{LS}(L) \stackrel{\mathrm{def}}{=}$ length of the list-schedule for L

$\mathrm{OPT} = \mathrm{OPT}(L) \stackrel{\mathrm{def}}{=}$ length of an optimum schedule for L

$\ell = \ell(L) \stackrel{\mathrm{def}}{=}$ a call with minimum bandwidth requirement among all calls that finish last in the list-schedule for L

$$L(e) \stackrel{\mathrm{def}}{=} \sum_{r \in L:\ r \text{ uses e}} b_r \cdot d_r = \text{the load on edge } e$$

List-Scheduling in Stars

	unit durations	arbitrary durations	bandwidth of last call in LS
Upper Bounds	$\mathrm{LS} \leq 4.875 \cdot \mathrm{OPT}$	$\mathrm{LS} \leq 5 \cdot \mathrm{OPT}$	
	$\mathrm{LS} \leq 3.875 \cdot \mathrm{OPT}$	$\mathrm{LS} \leq 4 \cdot \mathrm{OPT}$	$b_\ell \leq \frac{1}{2}$
	$\mathrm{LS} \leq \frac{2k}{k-1} \cdot \mathrm{OPT}$	$\mathrm{LS} \leq 3 \cdot \mathrm{OPT}$	$b_\ell \leq \frac{1}{k}, k > 2$
Lower Bounds	$\mathrm{LS} \approx 3.7 \cdot \mathrm{OPT}$	$\mathrm{LS} \approx 3.7 \cdot \mathrm{OPT}$	$b_\ell \leq \frac{1}{2}$
	$\mathrm{LS} \approx \frac{2k}{k-1} \cdot \mathrm{OPT}$	$\mathrm{LS} \approx 3 \cdot \mathrm{OPT}$	$b_\ell < \frac{1}{k}, k > 2$

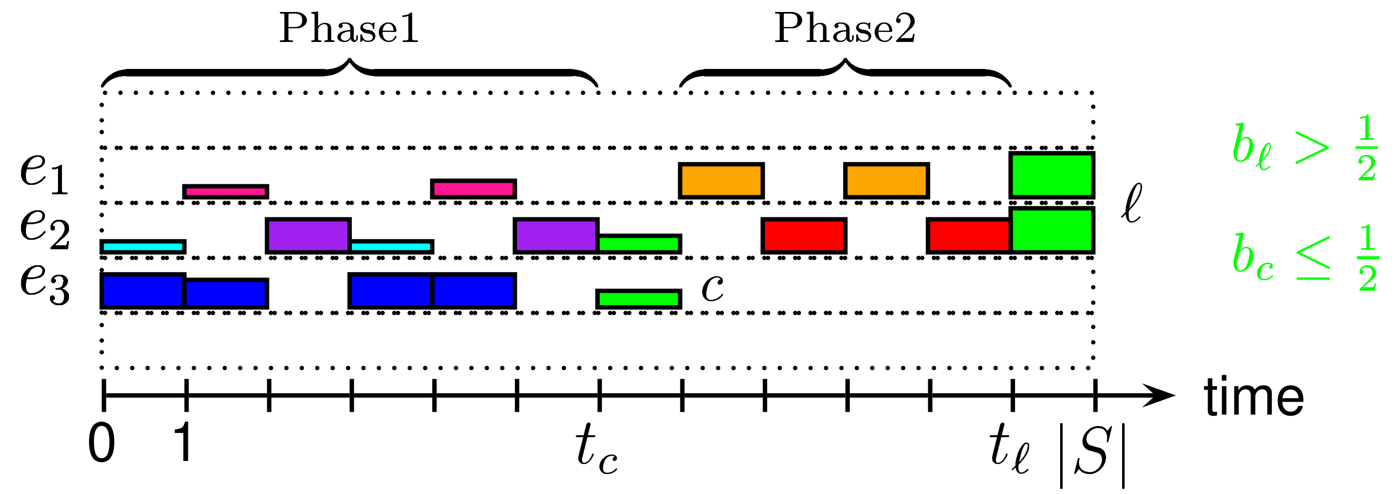
Proof Technique for Upper Bounds

1. Partition the list-schedule into phases according to the minimum bandwidth requirements of active calls on certain edges, and derive a set of inequalities.
2. Combine inequalities to obtain upper bounds.

Examples for Inequalities:

- If r is a call using edges e_1 and e_2, the load on e_1 or e_2 is greater than $\lceil t_r/2 \rceil \cdot (1 - b_r) + d_r \cdot b_r$.
- If calls with bandwidth requirements $> \frac{1}{2}$ use the same edge, OPT is at least as large as the sum of their durations.
- $\mathrm{OPT} \geq L(e)$ for all e.

A Simple Example for Phases (unit durations):



A = steps in Phase 1 during which ℓ is blocked only on e_1

B_1 = steps in Phase 1 during which c and ℓ are blocked on e_2

B_2 = steps in Phase 1 during which only ℓ is blocked on e_2

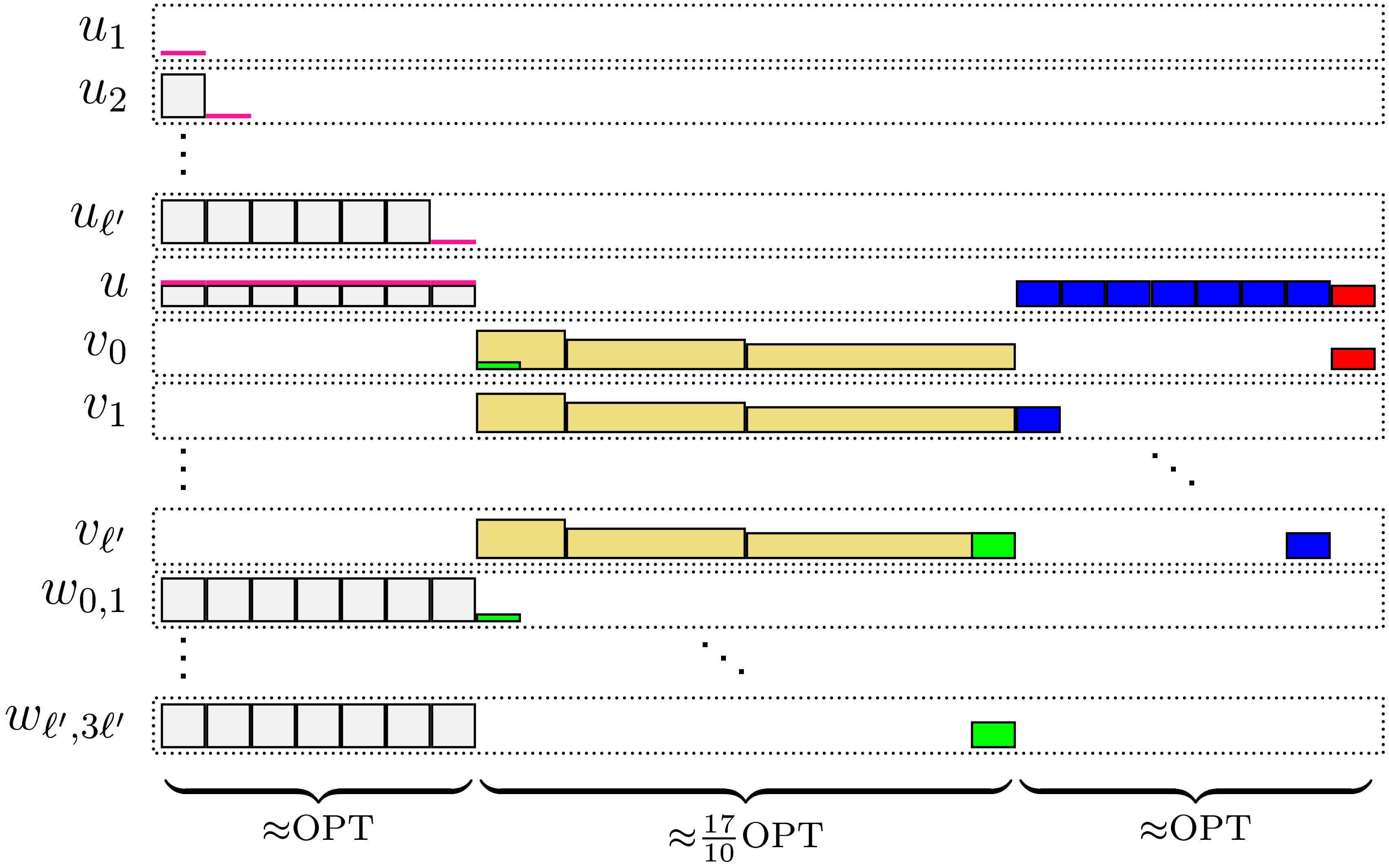
C = steps in Phase 2 during which ℓ is blocked only on e_1

D = steps in Phase 2 during which ℓ is blocked on e_2

E = steps in Phase 1 during which c is blocked only on e_3

$$\Rightarrow |S| = A + B_1 + B_2 + 1 + C + D + 1 \le \cdots \le 5 \cdot \mathrm{OPT}$$

Worst-Case Example with $\mathrm{LS}(L) \approx 3.7 \cdot \mathrm{OPT}(L)$



An Improvement for Stars and Unit Durations

Algorithm: Decreasing-Bandwidth-List-Scheduling (DBLS)

(1) sort L according to non-increasing bandwidth requirements

(2) use List-Scheduling to schedule L

Bounds on the performance of DBLS:

$$\mathrm{DBLS}(L) \leq \frac{8}{3} \cdot \mathrm{OPT}(L) = 2.\overline{6} \cdot \mathrm{OPT}(L)$$

There are infinitely many inputs L for which:

$$\mathrm{DBLS}(L) \approx \frac{22}{9} \cdot \mathrm{OPT}(L) = 2.\overline{4} \cdot \mathrm{OPT}(L)$$

Proof Idea: Modify proofs of upper and lower bounds for First-Fit-Decreasing (FFD) bin-packing.

Trees and Unit Durations

Note: $\frac{\mathrm{LS}(L)}{\mathrm{OPT}(L)} = \Omega(n)$ for worst-case inputs in a chain with n nodes.

Algorithm Level-List-Scheduling (LLS)

(1) assign levels to the nodes of the tree (distance from root)

(2) assign levels to calls (smallest node level on corresponding path)

(3) sort L according to non-decreasing levels

(4) use List-Scheduling to schedule L

$\Rightarrow$ **Upper Bound:** $\mathrm{LLS}(L) \leq 6 \cdot \mathrm{OPT}(L)$

Trees and Arbitrary Durations

Algorithm List-Scheduling by Levels (LSL)

(1) assign levels to the n nodes of the tree using a separator tree

(2) assign levels to calls (smallest node level on corresponding path)

(3) $L_i \leftarrow$ calls in L with level i, $0 \leq i < \log_2 n$

(4) use List-Scheduling to schedule $L_0, L_1, \ldots, L_{\log n - 1}$ in turn

$\Rightarrow$ **Upper Bound:** $\mathrm{LSL}(L) \leq 5 \log_2 n \cdot \mathrm{OPT}(L)$

Note: LSL is an on-line algorithm.

Summary of Results

Algorithm	Upper Bound	Lower Bound
Star, $d_r = 1$, LS	4.875	3.7
Star, $d_r = 1$, DBLS	$2.\overline{6}$	$2.\overline{4}$
Star, $d_r \in \mathbb{N}$, LS	5	3.7
Tree, $d_r = 1$, LLS	6	3.7
Tree, $d_r \in \mathbb{N}$, LSL	$5 \log n$	$3.7 \log n$

To do:
- tighten the upper and lower bounds
- find better algorithms
- consider different network topologies