Liquid Schedule Searching Strategies for the Optimization of Collective Network Communications

Emin Gabrielyan, Roger D. Hersch
École Polytechnique Fédérale de Lausanne
{Emin.Gabrielyan,RD.Hersch}@epfl.ch

Abstract

For collective communications, the upper limit of a network's capacity is its liquid throughput. The liquid throughput corresponds to the flow of a liquid in an equivalent network of pipes. However, in communication networks, the aggregate throughput of a collective communication pattern (traffic) scheduled according to straight forward topology unaware techniques may be several times lower than the maximal potential throughput of the network. In most of the cut-through, wormhole and wavelength division optical networks, there is a loss of performance due to congestions between simultaneous transfers sharing a common link resource. We propose to schedule the transfers of a traffic according to a schedule yielding the liquid throughput. Such a schedule, called liquid schedule, relies on the knowledge of the underlying network topology and ensures an optimal utilization of all bottleneck links. To build a liquid schedule, we partition the traffic into time frames comprising mutually non-congesting transfers keeping all bottleneck links occupied all the time. The search for mutually non-congesting transfers utilizing all bottleneck links is of exponential complexity. We present an efficient algorithm which non-redundantly traverses the search space and limits the search to only those sets of transfers, which are non-congesting and utilize all bottleneck links. We further propose a liquid schedule construction technique, which efficiently reduces the search space without affecting the solution space.

1. Introduction

Collective multicast communications are of increasing importance both in scientific and in commercial applications. The aggregate throughput of a collective communication pattern (traffic) depends on the underlying network topology. The amount of data that has to pass across the most loaded links of the network, called bottleneck links, gives their utilization time. The total size of a traffic divided by the utilization time of the bottleneck links gives an estimation of the *liquid throughput*, which corresponds to the flow capacity of a non-compressible fluid in a network of pipes [Melamed00]. Both in wormhole switching networks and in Wavelength Division Multiplexing (WDM) optical networks, due to possible link or wavelength allocation conflicts, not any combination of transfer requests may be carried out simultaneously. For a physical network modelled as a directed graph $G = (V(G), E(G))$, the objective is to minimize the number T of timeslots and/or wavelengths required to carry out a given set of transfer requests. Each transfer shall be allocated to one (and only one) time frame, such that no pair of transfers allocated to the same time frame use a common resource (link, wavelength).

The liquid scheduling problem is hard to solve. For the sizes of practical interest, solving the problem by Mixed Integer Linear Programming (MILP) [CPLEX02], [Fourer03] leads to very long computation times (see Appendix). Solving the problem by applying a heuristic graph colouring algorithm provides in short time a suboptimal solution. We propose an exact method for deriving the optimal solution, which is fast enough to allow real time scheduling of a traffic in congestion sensible transmission networks. Numerous applications require using network resources efficiently, for example parallel acquisition and distribution of multiple video streams [Chan01], [Sitaram00], switching of simultaneous voice communication sessions [H323], [EWSD04], [SIP04], and high energy physics, where particle collision events need to be transmitted from a large number of detectors and filters to clusters of processing nodes [CERN01].

We have tested liquid scheduling on a computer cluster interconnected by a high performance wormhole switch fabric [Kuonen99].

2. Applicable networks

This section introduces different types of electronic, optical and wireless networks having in common the problem of congestion avoidance. Liquid scheduling of collective communication patterns (multicast communications) is applicable to each of these networks.

2.1. Cut-through switching

In many high performance multicomputer communication networks, the links lying on the path of a message are kept occupied during the transmission of that message. Unlike packet switching (or store-and-forward switching) where each network packet is present at an intermediate router [Ayad97], wormhole switching (also called cut-through switching) [Liu01] transmits a message as a worm propagating itself across intermediate switches, i.e. a continuous stream of bits which make their way through the fabric spanning multiple switches. In a wormhole switching network [Duato99], [Shin96], [Rexford96], [Colajanni99], a message entering into the network is being broken up into small parts of equal size called flits (standing from flow-control digits). These flits are streamed across the network. All the flits of a packet follow the same path. As soon as a switch on the path of a message receives the head flit and processes the routing header, it triggers the flow of flits to the corresponding outgoing link. If the message encounters a busy outgoing link, the wormhole switch stalls the message in the network along the already established path until the link becomes available. Occupied channels are not released. A channel is released only when the last tail flit of the message has been transmitted. Thus each link laying on the path of the message is kept occupied during the whole transmission time of a message[1].

Compared with store and forward switches, wormhole switching considerably decreases the latency of message transmission across multiple routers. Wormhole switching makes the latency insensitive to the message distance. Most contemporary research and commercial multicomputers use some form of wormhole or advanced cut-through networks, e.g. Myrinet [Boden95], fat tree interconnections for clusters [Petrini03], [Quadrics] and Tnet [Horst95], [Brauss99].

Wormhole switching only pipelines messages and thus requires not more than a very small buffer. This enables a cost effective implementation of high performance switches on a single chip [Yocum97]. However, wormhole switching alone quickly saturates as load increases due to blocked message paths. Congestions occurring due to simultaneous transmission of messages sharing common network links result in an aggregate data throughput considerably lower than the liquid throughput offered by the network.

For the same set of collective communications the rate of network congestions may significantly vary depending in which order individual communications are carried out. Channel contentions can be avoided if the transfers are scheduled so that no congesting messages are transmitted at a time.

2.2. Lightpaths on demand

Lightpaths are end to end optical connections from a source node to a destination node over a wavelength within each intermediate link. Different lightpaths in a wavelength routing network can use the same wavelength as long as they do not share any common links. Fig. 1 shows an example of an optical network.

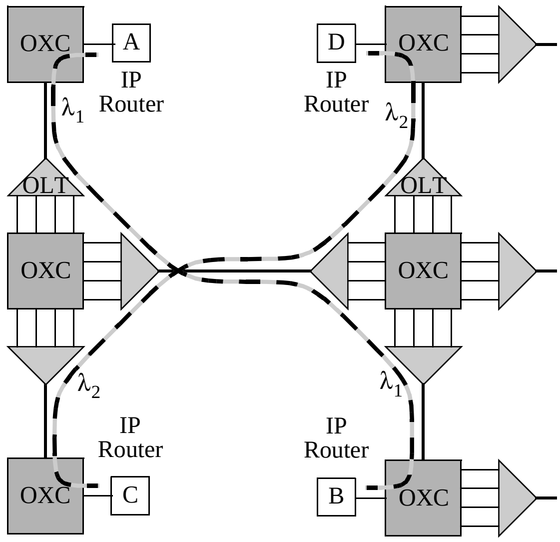


Fig. 1. Optical layer, wavelength-routing network.

The Optical Line Terminal (OLT) multiplexes multiple wavelengths into a single fiber and demultiplexes a set of wavelengths from a single fiber into separate fibers. The Optical Cross Connects (OXC) switches wavelengths from one port to another[2] [Ramaswami97], [Stern99].

Optical networks, such as the one shown in Fig. 1, are designed to provide lightpaths between the terminal edge nodes of a conventional network, for example IP routers. However from a network design point of view, it is essen-

1. However in virtual cut-through (VCT) networks, if the message encounters a busy outgoing link, the entire packet is buffered in the router and already allocated portions of the message path are released.

2. End nodes, i.e. IP routers, SONET terminals or ATM switches are usually plugged into network via an Optical Add/Drop Multiplexer (OADM).

tial to keep transmissions in the optical domain as long as possible. For example a relatively inexpensive optical switch can be implemented by an array of microscopic mirrors, build with micro electromechanical systems (MEMS). Such an optical cross connect, called wavelength-selective cross-connect (WSXC), does not provide wavelength conversion features[1]. In networks build with WSXC switches, the basic optical transmission channel remains on a fixed wavelength from end to end (a condition called wavelength continuity constraint) and therefore any lightpath must be assigned the same wavelength on all the links it traverses. Two lightpaths traversing a common link must be assigned different wavelengths. Let us assume that the optical cross-connects of the example in the Fig. 1 do not carry out wavelength conversion. The lightpath between nodes *A* and *B* uses one single wavelength λ_1 all along its route and therefore λ_1 cannot be reused by another lightpath at any link of the route. Two transmissions from IP router *A* to *B* and from *C* to *D* must then either be carried out on two different wavelengths λ_1 and λ_2, or must be scheduled in different timeslots, i.e. by optical burst switching [Yoo99], [Turner99]. In order to apply liquid scheduling to optical networks we assume that the routing control is carried out at the edge nodes. We also assume that appropriate signalling is available between edge nodes enabling them to collaborate. Then, liquid scheduling, exploited by edge nodes ensures congestion avoidance and leads to an efficient usage of wavelengths and timeslots. For an incoming continuous traffic, edge nodes repeatedly apply liquid scheduling on the buffered communication requests. For a continuously evolving traffic demand, liquid scheduling will only accept the sub flow, which can be sustained by the optical cloud. This leads to a liquid schedule based admission control mechanism [Jagannathan02], [Mandjes02].

There has been research on theoretical considerations about the required number of wavelengths and the complexity of finding a solution to the wavelength assignment problem according to the network topology [Bermond96], [Caragiannis02]. Feed-back and flow control based congestion avoidance mechanisms are studied in [Maach04], [Chiu89], [Ozbay98], [Loh96]. In contrast to these approaches, we establish schedules for the data transfers without trying to regulate the sending nodes' data rate.

1. Lightpaths may be converted from one wavelength to another along their route, however this conversion is costly, since in most cases it requires O/E and E/O conversions. OXCs providing wavelength conversion are called wavelength-interchanging cross-connects (WIXC). WIXCs do both space switching and wavelength conversion.

2.3. Time division networks, CDMA and spread spectrum wireless networks.

Satellite-switch time division multiplexing networks [Jain91] and Code Division Multiple Access (CDMA) spread spectrum wireless networks [Battiti99], [Leelahakriengkrai03] can also be viewed as interconnection topologies. With CDMA, the bottleneck resources comprise also orthogonal frequency spectra. Congestion occurs if two transmissions simultaneously compete for a common frequency. One may therefore try to apply a variant of liquid scheduling to CDMA wireless networks so as to ensure an optimal utilization of the resources and to obtain the highest possible overall throughput.

2.4. Liquid scheduling for parallel applications

We have shown [Gabrielyan03] that liquid schedules may significantly increase the overall performance of a parallel application running on a high-speed low latency supercomputer network, e.g. Myrinet [Boden95] or Tnet [Horst95].

Liquid schedules do not always exists. When a liquid schedule does not exist, we may search for suboptimal solutions offered by heuristic graph colouring methods [Gabrielyan03]. In the present paper, we focus our attention only on algorithms for the construction of liquid schedules.

3. The liquid scheduling problem

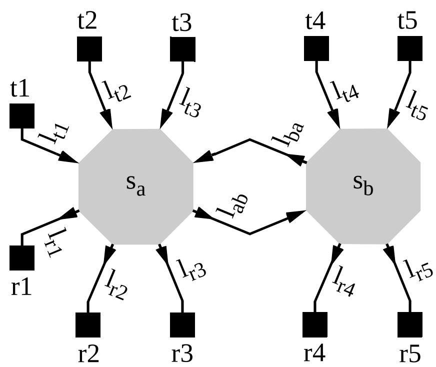
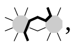
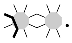
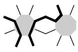
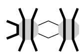
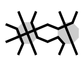
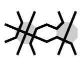
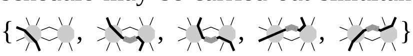
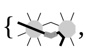
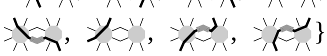
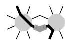
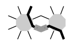


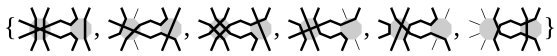
Fig. 2. Example of a network topology.

Let us consider a network topology (Fig. 2) consisting of ten end nodes $t1 \ldots t5, r1 \ldots r5$ (henceforth called processors), two wormhole cut-through switches s_a, s_b and twelve unidirectional links $l_{t1} \ldots l_{t5}, l_{r1} \ldots l_{r5}, l_{ab}, l_{ba}$ having identical throughputs. The processors $t1 \ldots t5$ only transmit data and $r1\ldots r5$ only receive data. It's easy to guess the routing, e.g. a message from *t4* to *r3* traverse links l_{t4}, l_{ba} and l_{r3}, and a message from *t1* to *r2* uses only links l_{t1} and l_{r2}.

We denote transfers symbolically to mark out the occupied network links. For example the transfer from *t4* to *r3* is symbolically represented as , the transfer from *t1* to *r2* as . We may also represent a set of transfers carried out simultaneously, e.g. a traffic transferring messages simultaneously from *t4* to *r3* and from *t1* to *r2* by .

Let each sending processor have messages to be transmitted to each receiving processor and let all messages have identical sizes [Naghshineh93]. Thus, in the present example, we have 25 transfers to carry out. Each of the ten links $l_{t1}\ldots l_{t5}, l_{r1}\ldots l_{r5}$ carries 5 transfers and the two links l_{ab}, l_{ba} must each carry 6 transfers. Therefore the links l_{ab}, l_{ba} are the network bottlenecks and have the longest active time. If the duration of the whole collective communication is as long as the active time of the bottleneck links, we say that the collective communication reaches its liquid throughput. In that case the bottleneck links are obviously kept busy all the time along the duration of the communication traffic. Assuming in this example a single link throughput $1Gbps$, the liquid throughput offered by the network is $(25/6)\times 1Gbps = 4.17Gbps$. Under identical packet size and link throughputs (kept all along this paper for the sake of simplicity) the *liquid throughput* of a traffic *X* is the ratio $\#(X)/\Lambda(X)$ multiplied by the single link throughput, where $\#(X)$ is the total number of transfers and $\Lambda(X)$ is the number of transfers carried out by one bottleneck link.

Now let us see if the order in which the transfers are carried out in this wormhole network has an impact on the collective communication performance. A straight forward schedule to carry out these 25 transfers is the round-robin schedule, according to which at first each transmitting processor sends the message to the receiving processor staying in front of it, then to the receiving processor staying at the next position, etc. Such a round robin schedule consists of 5 phases. The transfers of the first , second  and fifth  phase of the round-robin schedule may be carried out simultaneously, but the third  and fourth   phases contain congesting transfers, e.g. link l_{ab} (marked thick) can not be simultaneously used by the two transfers  and . None of these two phases can be carried out in less than two time frames and therefore the whole schedule lasts 7 time frames, instead of seemingly 5. Therefore the performance of our collective communication carried out according to the round-robin schedule corresponds to the throughput of $25/7 = 3.57$ messages per time frame or $(25/7)\times 1Gbps = 3.57Gbps$, which is less than the liquid throughput.

Nevertheless, a solution exists to schedule the 25 transfers within 6 time frames. The sequence of time frames  is an example of the liquid schedule for the 25-transfer collective communication request.

4. Definitions

The method we propose allows us to efficiently build liquid schedules for non-trivial network topologies. Thanks to liquid schedules we may considerably increase the collective data exchange throughputs, compared with traditional topology unaware schedules such as round-robin or random schedules. The present section introduces the definitions that will be further used for describing the liquid schedule construction method.

A single "point-to-point" transfer is represented by the set of communication links forming the network path between a transmitting and a receiving processor according to a given routing schema. A *transfer* is a set of links (i.e. the path between a sending processor and a receiving processor). A *traffic* is a set of transfers (i.e. a collective data exchange).

Fig. 3 shows a traffic for a collective data exchange carried out on the network of Fig. 2. The bottleneck links of the network are marked in bold. The exchange shown in Fig. 3 is a particular case of a traffic. Any collective exchange comprising transfers between possibly overlapping sets of sending and receiving processors is a traffic.

$$\left\{\begin{array}{c}
\{l_{t1}, l_{r1}\}, \{l_{t1}, l_{r2}\}, \{l_{t1}, l_{r3}\}, \{l_{t1}, \boldsymbol{l_{ab}}, l_{r4}\}, \{l_{t1}, \boldsymbol{l_{ab}}, l_{r5}\} \\
\{l_{t2}, l_{r1}\}, \{l_{t2}, l_{r2}\}, \{l_{t2}, l_{r3}\}, \{l_{t2}, \boldsymbol{l_{ab}}, l_{r4}\}, \{l_{t2}, \boldsymbol{l_{ab}}, l_{r5}\} \\
\{l_{t3}, l_{r1}\}, \{l_{t3}, l_{r2}\}, \{l_{t3}, l_{r3}\}, \{l_{t3}, \boldsymbol{l_{ab}}, l_{r4}\}, \{l_{t3}, \boldsymbol{l_{ab}}, l_{r5}\} \\
\{l_{t4}, \boldsymbol{l_{ba}}, l_{r1}\}, \{l_{t4}, \boldsymbol{l_{ba}}, l_{r2}\}, \{l_{t4}, \boldsymbol{l_{ba}}, l_{r3}\}, \{l_{t4}, l_{r4}\}, \{l_{t4}, l_{r5}\} \\
\{l_{t5}, \boldsymbol{l_{ba}}, l_{r1}\}, \{l_{t5}, \boldsymbol{l_{ba}}, l_{r2}\}, \{l_{t5}, \boldsymbol{l_{ba}}, l_{r3}\}, \{l_{t5}, l_{r4}\}, \{l_{t5}, l_{r5}\}
\end{array}\right\}$$

Fig. 3. Example of a traffic composed of 25 transfers carried out over the network shown on Fig. 2.

A link l is *utilized* by a transfer x if $l \in x$. A link l is utilized by a traffic X if l is utilized by a transfer of X. Two transfers are in *congestion* if they share a common link. Note that we will be limiting ourselves to data exchanges consisting of identical packet sizes.

A *simultaneity* of a traffic X is a subset of X consisting of mutually non-congesting transfers. A transfer is in congestion with a simultaneity if the transfer is in congestion with at least one member of the simultaneity. A simultaneity of a traffic is *full* if all transfers in the complement of the simultaneity in the traffic are in congestion with that simultaneity. A simultaneity of a traffic obviously can be carried out within one time frame (the time to carry out a single transfer). The *load* $\lambda(l, X)$ of link l in a traffic X is the number of transfers in X using link l. The *duration* $\Lambda(X)$ of a traffic X is the maximal value of the load among all links involved in the traffic.

$$\Lambda(X) = max\{\lambda(l_i, X)\} \qquad \forall l_i$$

The links having maximal load values, i.e. $\lambda(l, X) = \Lambda(X)$, are called *bottlenecks.* The *liquid throughput* of a traffic X is the ratio $\#(X)/\Lambda(X)$ multiplied by the single link throughput, where $\#(X)$ is the number of transfers in the traffic X.

We define a simultaneity of X as a *team* of X if it uses all bottlenecks of X. A team of X is *full* if it is a full simultaneity of X. Let $\Re(X)$ and $\Im(X)$ be respectively the sets of all full simultaneities and all full teams of X.

In order to form liquid schedules, we try to schedule transfers in such a way that all bottleneck links are always kept busy. Therefore we search for a liquid schedule by trying to assemble non-overlapping teams carrying out all transfers of the given traffic, i.e. we partition the traffic into teams. To cover the whole solution space we need to generate all possible teams of a given traffic. This is an exponentially complex problem. It is therefore important that the team traversing technique be non-redundant and efficient, i.e. each configuration is evaluated once and only once, without repetitions.

5. Obtaining full simultaneities

To obtain all full teams, we first optimize the retrieval of all simultaneities and then use that algorithm to retrieve all full teams.

Recall that in a traffic X, any mutually non-congesting combination of transfers is a simultaneity. A full simultaneity is a combination of non-congesting transfers taken from X, such that its complement in X contains only transfers congesting with that simultaneity.

We can categorize full simultaneities according to the presence or absence of a given transfer x. A full simultaneity is x-positive if it contains transfer x. If it does not contain transfer x, it is x-negative. Thus the set of full simultaneities $\Re(X)$ is partitioned into two non-overlapping subsets: an x-positive and x-negative subset of $\Re(X)$. For example, if y is another transfer, the set of x-positive full simultaneities may be further partitioned into y-positive and y-negative subsets. Iteration of this concept allows us to recursively traverse the whole set of all full simultaneities $\Re(X)$, one by one, without repetitions.

Let us define a *category* of full simultaneities of X as an ordered triplet (*excluder, depot, includer*), where the includer is a simultaneity of X (not necessarily full), the excluder contains some transfers of X non-congesting with the includer and the depot contains all the remaining transfers non-congesting with the includer.

A category, defined by the transfers of its includer and excluder, constrains a subset of full simultaneities. We therefore say that a full simultaneity is *covered* by a category R, if the full simultaneity contains all the transfers of the category's includer and does not contain any transfer of the category's excluder. Consequently, any full simultaneity covered by a category is the category's includer together with some transfers taken from the category's depot. The collection of all full simultaneities of X covered by a category R is defined as the *coverage* of R. We denote the coverage of R as $\phi(R)$.

Transfers of a category's includer form a simultaneity (not full). By adding different variations of transfers from the depot, we may obtain all possible full simultaneities covered by the category.

The category $(\varnothing, X, \varnothing)$ is a *prim-category* since it covers all full simultaneities of X, i.e. $\phi(\varnothing, X, \varnothing) = \Re(X)$.

By taking an arbitrary transfer x from the depot of a category R, we partition the coverage of R into x-positive and x-negative subsets. The respective x-positive and x-negative subsets of a coverage of R are coverages of two categories derived from R: a positive subcategory and a negative subcategory of R.

The positive subcategory R_{+x} is formed from the category R by adding transfer x to its includer, and by removing from its depot and excluder[1] all transfers congesting with x. The negative subcategory R_{-x} is formed from the category R by moving transfer x from its depot to its excluder. The replacement of a category R by its two sub

1. Since transfers congesting with x are naturally excluded from a full simultaneity covered by R_{+x}, we may safely remove them from the excluder (and avoid redundancy in the exclusion constraint)

categories R_{+x} and R_{-x} is defined as a *fission* of the category. Fig. 4 and Fig. 5 show an example of fission of a category into positive and negative sub categories.

$$R = \left\{ \begin{array}{cc} \{\Theta_1\} & includer \\ \{\Xi_1, \boldsymbol{x}, \Xi_2, \Theta_2\} & depot \\ \{\Xi_3, \Theta_3\} & excluder \end{array} \right\}$$

Θ - denotes any transfer non-congesting with $\boldsymbol{x}$

Ξ - denotes any transfer congesting with $\boldsymbol{x}$

Fig. 4. An initial category before fission, where symbol Ξ, represents any transfer that is in congestion with *x* and symbol Θ represents any transfer which is simultaneous with *x*.

Fig. 4 shows an example of a category *R* and Fig. 5 shows the resulting two sub categories obtained from the initial category by a fission relatively to a transfer *x* taken from the depot.

$$R \rightarrow \left\{ \begin{array}{l} R_{+x} = \left\{ \begin{array}{cc} \{\Theta_1, \boldsymbol{x}\} & includer \\ \{\Theta_2\} & depot \\ \{\Theta_3\} & excluder \end{array} \right\} \\ R_{-x} = \left\{ \begin{array}{cc} \{\Theta_1\} & includer \\ \{\Xi_1, \Xi_2, \Theta_2\} & depot \\ \{\Xi_3, \Theta_3, \boldsymbol{x}\} & excluder \end{array} \right\} \end{array} \right.$$

Fig. 5. Fission of the category of Fig. 4 into its positive and negative sub categories.

The coverage of *R* is partitioned by the coverages of its sub categories R_{+x} and R_{-x}, i.e. the coverage of a category is the union of coverages of its sub categories: $\phi(R_{+x}) \cup \phi(R_{-x}) = \phi(R)$, and the coverages of the sub categories have no common transfers, $\phi(R_{+x}) \cap \phi(R_{-x}) = \varnothing$.

A *singular* category is a category that covers only one full simultaneity. That full simultaneity is equal to the includer of the singular category. The depot and excluder of a singular category are empty.

We apply the binary fission to the prim-category and split it into two categories. Then, we apply the fission to each of these categories. Repeated fission increases the number of categories and narrows the coverage of each category. Eventually, the fission will lead to singular categories only, i.e. categories whose coverage consists of a single full simultaneity. Since at each stage we have been partitioning the set of full simultaneities, at the final stage we know that each full simultaneity is covered by one and only one singular category.

The algorithm recursively carries out the fission of categories and yields all full simultaneities without repetitions.

There is a further optimization to be considered. Take a category. A full simultaneity must contain no transfer from that category's excluder in order to be covered by that category. In addition, since the full simultaneity is full, it is in congestion with all transfers that it does not contain. Obviously any full simultaneity covered by some category must congest with each member of that category's excluder. Therefore, transfers congesting with the transfers of the excluder must be available in the depot of the category[1]. If the excluder contains at least one transfer, for which the depot has no congesting transfer, then this category is *blank*. The includer of a blank category, cannot be further extended by the transfers of the depot to a simultaneity which is full (and congests with every remaining transfer of the excluder). The coverage of a blank category is therefore empty and there is no need to pursue its fission.

Let us now instead of retrieving all full simultaneities retrieve all full teams (i.e. those full simultaneities, which ensure the utilization of all bottleneck links).

A category within *X* is *idle* if its includer and its depot together don't use all bottlenecks of *X*. This mean that we can not grow the current simultaneity (i.e. the includer of the category) into a full simultaneity, which will use all bottlenecks. The coverage of an idle category does therefore not contain a full simultaneity, which is a team. Idle categories allow us to prune the search tree.

Carrying out successive fissions, starting from the prim-category and continuously removing all the blank and idle categories ultimately leads to all full teams.

6. Speeding up the search for full teams

This section presents an additional method for speeding up the search for all full teams $\Im(X)$ of an arbitrary traffic *X*.

Let us consider from the original traffic *X* only those transfers that use bottlenecks of *X* and call this set of transfers the *skeleton* of *X*. We denote the skeleton of *X* as $\varsigma(X)$. Obviously, $\varsigma(X) \subset X$.

1. The category's excluder, according to the fission algorithm, keeps no transfer congesting with the includer.

Fig. 6 shows the relative size of skeletons compared with the size of the corresponding traffic, for 362 different traffic patterns within the T1 32 node cluster computer (see Fig. 11, in section 8). The skeleton sizes are on average 31.5% of the corresponding traffic sizes.

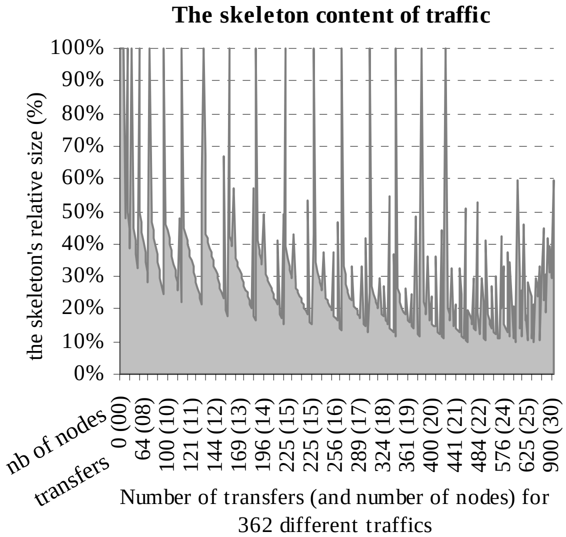


Fig. 6. Proportion of the number of transfers within a skeleton, compared with the number of transfers of the corresponding traffic.

When considering the skeleton of a traffic *X* as another traffic, the bottlenecks of the skeleton of a traffic are the same as the bottlenecks of the traffic. Consequently, a team of a skeleton is also a team of the original traffic.

We may first obtain all full teams of the traffic's skeleton by iteratively applying the fission algorithm and by eliminating the idle categories.

Then, a full team of the original traffic may be obtained by adding a combination of non-congesting transfers to a team of the traffic's skeleton.

We therefore obtain the set of a traffic's full teams $\Im(X)$ by carrying out the following steps:

1. Obtain the set of the skeleton's full teams $\Im(\varsigma(X))$ by applying the fission algorithm.
2. Create for each skeleton's full team a category by:
   - 2.1. Initializing the includer with the transfers of the skeleton's full team;
   - 2.3. Initializing the excluder as empty;
   - 2.2. And putting into the depot all transfers of *X* non-congesting with the includer.
3. Apply the fission to each category, discarding the check for idle categories, since the includer is already a team, i.e. it uses all bottlenecks.

By first applying the fission to the skeleton and then expanding the skeleton's full teams to the traffic's full teams, we strongly reduce the processing time and at the same time we obtain all full teams of the original traffic without repetitions.

We measured the reduction in search space according to the different search space reduction methods we propose. We consider 23 different traffic patterns within the T1 cluster computer (see section 8). The search space is given by the number of categories that are being iteratively traversed by the fission algorithm. Fig. 7 shows the obtained search space reductions compared with a naive algorithm that would build full teams according to a coverage partitioning strategy, i.e. by constructing categories thanks to the fission algorithm, but without any of the proposed optimizations.

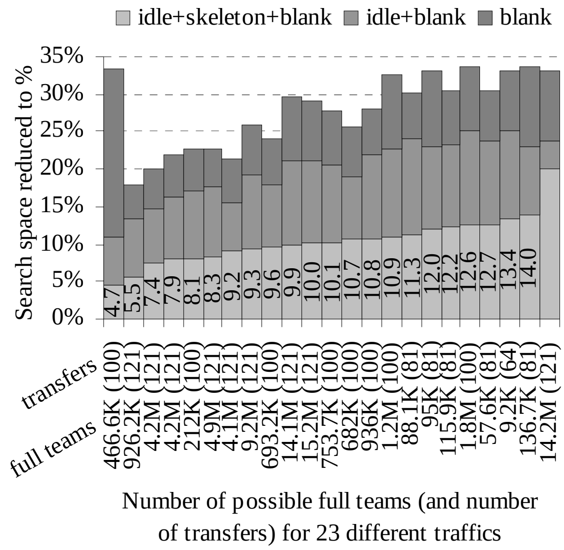


Fig. 7. Search space reduction obtained by idle+skeleton+blank optimization steps.

The skeleton algorithm together with the idle and blank optimizations reduces on average the search space to 10.6%, i.e. full teams are computed 9.43 times faster than without search space reduction techniques. Note that in the above comparison even the naive algorithm is smart enough to avoid repeatedly exploring the full simultaneities.

7. Construction of liquid schedules

Having the capability of building full teams, this section presents the general method for building liquid schedules on irregular topologies for any collective communication pattern. Note that we neglect network latencies, consider a constant packet size and assume static routing.

Let us introduce the definition of a schedule. By defining a *partition* of *X* as a disjoint collection of non-empty subsets of *X* whose union is *X* [Halmos74], a *schedule* α of a traffic *X* is a collection of simultaneities of *X* partitioning the traffic *X*. An elements of a schedule α is called *time frame*. The *length* $\#(\alpha)$ of a schedule α is the number of time frames in α. A schedule of a traffic is *optimal* if the traffic does not have any shorter schedule. If the length of a schedule is equal to the duration[1] of the traffic, then the schedule is *liquid*, i.e. a schedule α of a traffic *X* is liquid if $\#(\alpha) = \Lambda(X)$.

Fig. 8 shows a liquid schedule for the collective traffic shown in Fig 2.

time frame (1) time frame (2) time frame (3)

$$\left\{\begin{matrix}\{l_{t1}, \mathbf{l_{ab}}, l_{r4}\}\\ \{l_{t2}, l_{r2}\}\\ \{l_{t3}, l_{r3}\}\\ \{l_{t4}, \mathbf{l_{ba}}, l_{r1}\}\\ \{l_{t5}, l_{r5}\}\end{matrix}\right\} \left\{\begin{matrix}\{l_{t1}, \mathbf{l_{ab}}, l_{r5}\}\\ \{l_{t2}, l_{r1}\}\\ \{l_{t4}, \mathbf{l_{ba}}, l_{r2}\}\\ \{l_{t5}, l_{r4}\}\end{matrix}\right\} \left\{\begin{matrix}\{l_{t1}, \mathbf{l_{r3}}\}\\ \{l_{t2}, \mathbf{l_{ab}}, l_{r4}\}\\ \{\mathbf{l_{t3}}, l_{r1}\}\\ \{l_{t4}, l_{r5}\}\\ \{l_{t5}, \mathbf{l_{ba}}, l_{r2}\}\end{matrix}\right\}$$

time frame (4) time frame (5) time frame (6)

$$\left\{\begin{matrix}\{l_{t1}, l_{r2}\}\\ \{l_{t2}, \mathbf{l_{r3}}\}\\ \{\mathbf{l_{t3}}, \mathbf{l_{ab}}, l_{r4}\}\\ \{l_{t5}, \mathbf{l_{ba}}, l_{r1}\}\end{matrix}\right\} \left\{\begin{matrix}\{l_{t1}, l_{r1}\}\\ \{l_{t2}, \mathbf{l_{ab}}, \mathbf{l_{r5}}\}\\ \{\mathbf{l_{t3}}, l_{r2}\}\\ \{\mathbf{l_{t4}}, \mathbf{l_{ba}}, \mathbf{l_{r3}}\}\end{matrix}\right\} \left\{\begin{matrix}\{\mathbf{l_{t3}}, \mathbf{l_{ab}}, \mathbf{l_{r5}}\}\\ \{\mathbf{l_{t4}}, \mathbf{l_{r4}}\}\\ \{l_{t5}, \mathbf{l_{ba}}, \mathbf{l_{r3}}\}\end{matrix}\right\}$$

Fig. 8. The time frames of a liquid schedule of the collective traffic shown in Fig. 2.

If a schedule is liquid, then each of its time frames must use all bottlenecks. Inversely, if all time frames of a schedule use all bottlenecks, the schedule is liquid.

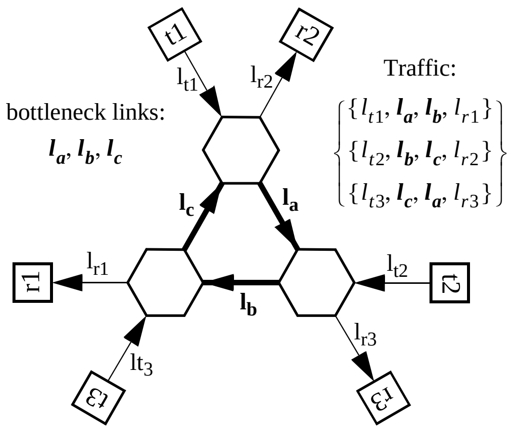


Fig. 9. This traffic has no team and no liquid schedule.

The necessary and sufficient condition for the liquidity of a schedule is that all bottlenecks be used by each time frame of the schedule. Since a simultaneity of *X* is defined as a *team* of *X*, if it uses all bottlenecks of *X*, an equivalent condition for the liquidity of a schedule α on *X* is that each time frame of α be a team of *X*.

A liquid schedule is optimal, but the inverse is not always true, meaning that a traffic may not have a liquid schedule. Fig. 9 shows a simple traffic with three bottleneck links. Since there is no schedule whose time frames keep all bottleneck links all the time busy, this traffic has no team and therefore no liquid schedule.

In the Appendix, we formulate the problem of searching for a liquid schedule with Mixed Integer Linear Programming (MILP), [CPLEX02], [Fourer03]. We compare the performances of the liquid schedule search approach presented here with that of MILP.

7.1. Liquid schedule naive search algorithm

We first propose a simple technique for the construction of a liquid schedule and then introduce an optimization improving the efficiency of liquid schedule construction.

Our strategy for finding a liquid schedule relies on partitioning the traffic into a set of teams forming the sequence of time frames. Associate to the traffic *X* all its possible teams $A_1, A_2, \ldots, A_n$ which could be selected as the schedule's first time frame. $X - A_1, X - A_2, \ldots$ is the variety of possible subtraffics remaining after the choice of the first time frame. Each of the possible subtraffics X_i remaining after the selection of the first time frame has its own set of possibilities for the second time frame $\aleph(X_i) = \{A_{i,1}, A_{i,2}, A_{i,3}, \ldots\}$. The choice of the second team for the second time frame yields a further reduced subtraffic (see Fig. 10).

Dead ends are possible if there are no choice for the next time frame, i.e. no team of the original traffic may be formed from the transfers of the reduced traffic. A dead end situation may occur, for example, when the remaining subtraffic appears to be like the one shown in Fig. 9. Once a dead end is faced, backtracking occurs.

The construction recursively advances and backtracks until a valid liquid schedule is formed. A valid liquid schedule is obtained, when the transfers remaining in the reduced traffic form one single team for the last time frame of the liquid schedule.

We use the search tree shown in Fig. 10 and assume that at any stage the choice $\aleph(X_{sub})$ for the next time frame is among the set of the original trafic's teams $\tilde{\mathfrak{I}}(X)$, i.e. $\aleph(X_{sub}) = \left\{A \in \tilde{\mathfrak{I}}(X) \middle| A \subset X_{sub}\right\}$. In the next sub sec-

1. The duration of a traffic *X* is the load of its bottlenecks.

tions we reduce the search space by considering newly emerging bottlenecks at successive time frames.

$$X \qquad \aleph(X) = \{A_1, A_2, A_3, \ldots\}$$

$$X_1 = X - A_1 \qquad \aleph(X_1) = \{A_{1,1}, A_{1,2}, A_{1,3}, \ldots\}$$

$$X_{1,1} = X_1 - A_{1,1} \qquad \aleph(X_{1,1}) = \{A_{1,1,1}, A_{1,1,2}, \ldots\} \qquad (\ldots) \quad (\ldots) \quad \ldots$$

$$X_{1,2} = X_1 - A_{1,2} \qquad \aleph(X_{1,2}) = \{A_{1,2,1}, A_{1,2,2}, \ldots\} \qquad (\ldots) \quad \ldots$$

$$X_{1,3} = X_1 - A_{1,3} \qquad \aleph(X_{1,3}) = \{A_{1,3,1}, A_{1,3,2}, \ldots\} \qquad \{\ldots\} \quad \ldots$$

$$X_2 = X - A_2 \qquad \aleph(X_2) = \{A_{2,1}, A_{2,2}, A_{2,3}, \ldots\}$$

$$X_{2,1} = X_2 - A_{2,1} \qquad \aleph(X_{2,1}) = \{A_{2,1,1}, A_{2,1,2}, \ldots\} \qquad (\ldots) \quad \ldots$$

$$X_{2,2} = X_2 - A_{2,2} \qquad \aleph(X_{2,2}) = \{A_{2,2,1}, A_{2,2,2}, \ldots\} \qquad \{\ldots\} \quad \ldots$$

$$X_3 = X - A_3 \qquad \aleph(X_3) = \{A_{3,1}, A_{3,2}, A_{3,3}, \ldots\} \qquad \{\ldots\} \quad \ldots$$

Fig. 10. Liquid schedule search tree. $X_{i_1 i_2 \ldots i_n}$ denotes a reduced subtraffic at the layer $n+1$ of the three and $A_{i_1 i_2 \ldots i_n i_{n+1}}$ denotes a candidate for the time frame $n+1$. The operator $\aleph$ applied to a subtraffic X_{sub} represents the set of all possible candidates for a time frame at the present stage.

7.2. Search space reduction by considering newly emerging bottlenecks

We observe in Fig. 8 that when we step from one time frame to the next, additional new bottleneck links emerge, e.g. from time frame 3 on, links $\boldsymbol{l}_{t3}$ and $\boldsymbol{l}_{r3}$ appear as new bottlenecks.

In the construction strategy presented in the previous subsection we considered as a possible time frame any team of the original traffic X that can be built from the transfers of the reduced subtraffic. We have shown [Gabrielyan03] that for the liquidity of a schedule, it is necessary for each time frame to be not only a team of the original traffic but also a team of the reduced subtraffic. If α is a liquid schedule on X and A is a *time frame* of α, then $\alpha - \{A\}$ is a liquid schedule on $X - A$.

Thus a liquid schedule may not contain a time frame which is a team of the original traffic but is not a team of a subtraffic obtained by removing some of the other time frames. Therefore we can limit at each iteration our choice to the collection of only those teams of the original traffic which are also teams of the current reduced subtraffic. Since the reduced subtraffic contains additional bottleneck links, there are less teams in the reduced subtraffic than teams remaining from the original traffic.

By considering in each time frame all occurring bottlenecks, we considerably reduce the search space without affecting the solution space, i.e. $\aleph(X_{sub}) = \tilde{\mathfrak{I}}(X_{sub})$.

7.3. Liquid schedule construction optimization by considering only full teams

We can build a liquid schedule by limiting the choice of teams of the reduced subtraffic to its full teams.

Let us modify a liquid schedule so as to convert one of its teams into a full team. We assume that a traffic X has a liquid schedule α. Let A be a time frame of α. If A is not a full team of X, then, by moving the necessary transfers from other time frames of α, we can convert the team A to a full team. Evidently, the properties of liquidity (partitioning, simultaneousness and length) of α will not be affected.

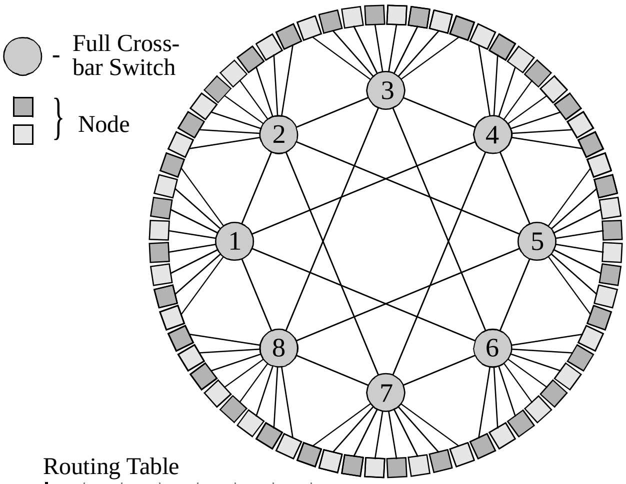
Therefore if a liquid schedule is built by a choice of a non full team $\tilde{A}$ of X_{sub} at any stage of construction, then the liquid schedule could have also been built by a choice of a full team A of X_{sub}, such that $\tilde{A} \subset A$. Therefore the choice of the teams in the construction may be narrowed from the set of all teams to the set of full teams only, i.e. $\aleph(X_{sub}) = \mathfrak{I}(X_{sub})$.

The expression bellow summarizes the search space reduction by building liquid schedule using full teams of the reduced traffic.

$$\left\{A \in \tilde{\mathfrak{I}}(X) \middle| A \subset X_{sub}\right\} \supset \tilde{\mathfrak{I}}(X_{sub}) \supset \mathfrak{I}(X_{sub})$$

8. Testbed and measurements

In this section we present a testbed consisting of sample traffic patterns for various topologies. Measurements of collective data exchange throughputs enable us to validate the efficiency of our full team search and liquid schedule construction strategy.



Routing Table

	1	2	3	4	5	6	7	8
1	×	✓	2	✓	4	✓	8	✓
2	✓	×	✓	7	✓	3	✓	5
3	2	✓	×	✓	4	✓	8	✓
4	✓	7	✓	×	✓	7	✓	3
5	4	✓	4	✓	×	✓	6	✓
6	✓	3	✓	7	✓	×	✓	1
7	8	✓	8	✓	6	✓	×	✓
8	✓	5	✓	3	✓	1	✓	×

Transmissions from switch *i* to switch *j* are routed through the switch with the number located in the table's *i*th raw and *j*th column. Symbol "✓" indicates a direct link between two switches.

Fig. 11. Architecture of the T1 cluster computer interconnected by a high performance wormhole switch fabric.

As basic network topology for our testbed, we use the Swiss-T1 cluster (called T1, see Fig. 11). The network of the T1 forms a K-ring [Kuonen99] and has a static routing scheme. The throughputs of all links are identical and equal to 86*MB/s*. The cluster consists of 32 nodes, each one comprising 2 processors, i.e. 64 processors, [SWISSTX99], [Gruber00].

The sample traffic patterns are selected from different configurations of half-to-half collective data exchanges between a set of sending and a set of receiving processors, where each sending processor carries out a transmission to each receiving processor. Within each node we allocate one of the processors for transmission and the other one for reception such that for any given allocation of nodes, we obtain an equal number of sending and receiving processors.

With a T1 cluster incorporating 32 nodes, 8 switches and with 4 nodes per switch we have 5 possibilities of allocating nodes to switches. This yields $5^8 = 390625$ different node allocation patterns. To limit our choice to really different topologies, we have computed the liquid throughputs for each of the 390625 topologies, taking into account the routing information of the cluster. Because of various symmetries within the network, many of these topologies yield an identical liquid throughput and only 362 topologies yielding different liquid throughput values were obtained.

Fig. 12. shows these 362 topologies, each one being characterized by the number of contributing nodes and by its liquid throughput. Depending on the specific topology, the liquid throughput for a given number of nodes may considerably vary.

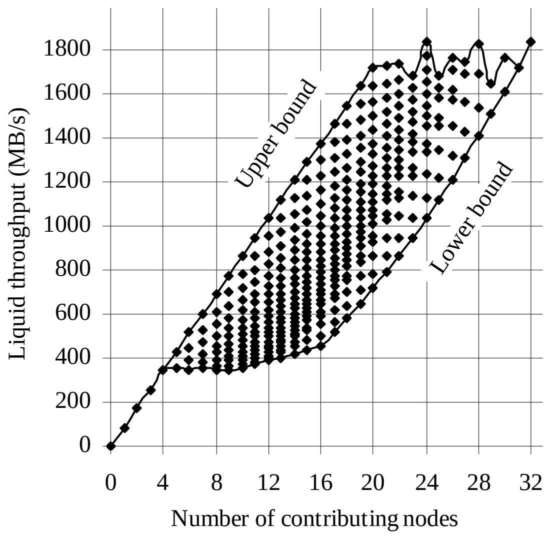


Fig. 12. Liquid throughput in relation to the number of nodes with variations according to different topologies.

These 362 topologies may be placed on one axis, sorted first by the number of nodes and then according to their liquid throughput. For each topology, Fig. 13 shows the theoretical liquid throughput and the throughput measured according to a topology-unaware round-robin schedule.

For each measurement, the amount of data transferred from a transmitting processor to a receiving processor is equal to 2*MB*. For each topology, 20 measurements were made. The black dots represent the median of the collected results. The throughput difference shows that topology-unaware scheduling techniques do not utilize efficiently the potential throughput capabilities offered by the communication network.

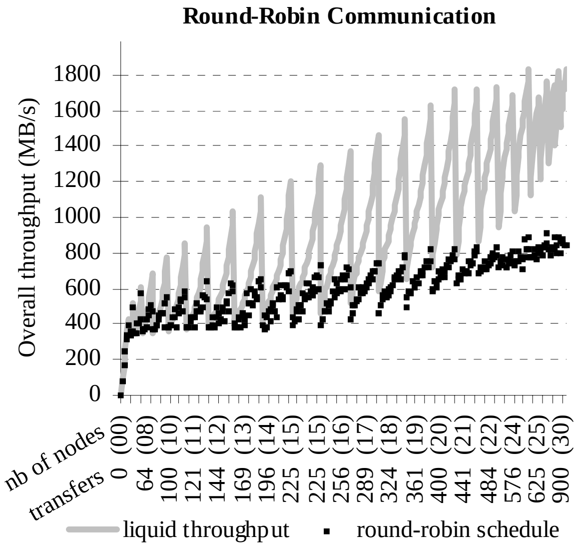


Fig. 13. Theoretical liquid throughput and measured round-robin schedule throughput for 362 network sub topologies.

The 362 sample traffic patterns were scheduled by our liquid scheduling algorithms. The traffic for each pattern was launched onto the network according to the computed liquid schedules. Overall throughput results are measured and presented in Fig. 14. The curve of the theoretical value is given for comparison.

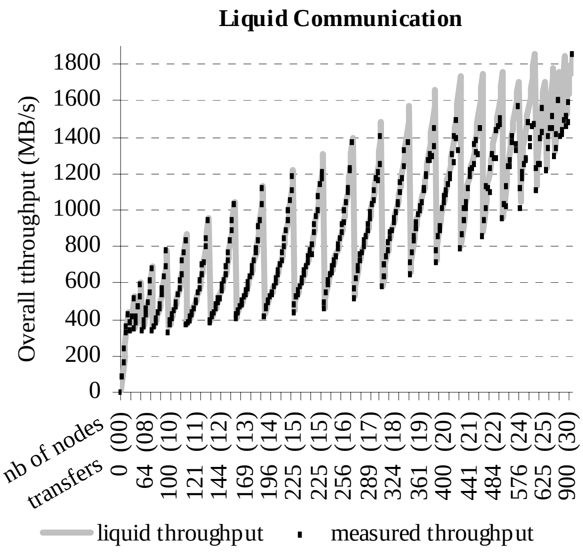


Fig. 14. Predicted liquid throughput and measured throughput according to the computed liquid schedule.

Each black dot represents the median of 7 measurements. Processor to processor transfers have a size of 5*MB*. The measured aggregate throughputs (black dots) are very close to the theoretically expected values of the liquid throughput (gray curve). For many topologies, liquid scheduling allows to increase the aggregate throughput by a factor of two compared with topology-unaware round-robin scheduling (Fig. 13).

Thanks to the presented search space reduction algorithms, the computation time of a liquid schedule takes for more than 97% of the considered topologies less than $1/10$ of a second on one Compaq Alpha 500MHz computer.

9. Conclusion

In high performance networks based on cut-through wormhole switch fabrics or on wavelength division multiplexing optical networks, significant performance drops may be observed due to congestions between transfers sharing common resources. We propose a method for scheduling collective communications which avoids congestions. The proposed scheduling method yields an aggregate throughput equal to the network's theoretical upper limit, i.e. its liquid throughput. Efficient computation of the liquid schedule is achieved by breaking the overall traffic request into time frames within which all the transfers of the traffic are allocated. To ensure a liquid schedule, the time frames must incorporate as many transfers as possible and utilize all bottleneck links. In order to compute the liquid schedule we propose a method for traversing efficiently and without redundancy all candidate subsets of simultaneous transfers.

We obtained a considerable speed up in the construction of liquid schedules by carrying out the following optimizations, which do not affect the solution space.

1. Full teams are enumerated by partitioning the solution space using inclusion and exclusion constraints.
   1.1. The *blank* optimization identifies empty partitions, which do not need to be further evaluated.
   1.2. The *idle* optimization identifies partitions containing no full teams, which do not need to be further evaluated.
   1.3. The *skeleton* optimization speeds up the retrieval of full teams, first by considering only the transfers necessary to keep all bottleneck links busy and then by adding up non-congesting transfers.
2. We construct liquid schedules by partitioning the traffic into teams.
   2.1. The construction of the liquid schedule is accelerated by limiting at each time frame the choice to teams, which use also the newly

emerging bottleneck links, i.e. teams of the reduced traffic.

2.2. Choosing only full teams of the reduced traffic further speeds up the construction of the liquid schedule.

Measurements on the traffic carried out on various sub-topologies of the Swiss T1 cluster computer have shown that for most of the sub-topologies we are able to increase the collective communication throughput by a factor between 1.5 and 2. In congestion sensible coarse-grain transmission networks, i.e. wireless networks, wormhole or lightpath switching networks, liquid scheduling may considerably improve the utilization of transmission resources such as communication links, wavelengths and orthogonal frequency spectra. Liquid schedules avoid congestions and minimize the overall transmission time for collective communications.

In the future we also intend to develop multipath routing solutions, which increase the traffic's fault-tolerance against link failures and at the same time keep the throughput liquid.

References

[Ayad97] N.M.A. Ayad, F.A. Mohamed, "Performance analysis of a cut-through vs. packet-switching techniques", Proc. Second IEEE Symposium on Computers and Communications, 1997, 230-234.

[Battiti99] Roberto Battiti, Alan A. Bertossi, Maurizzio A. Bonuccelli, "Assigning Codes in Wireless Networks: Bounds and Scaling Properties.", ACM/Baltzer Wireless Networks, Vol. 5, 1999, 195-209.

[Beauquier97] B. Beauquier, J.C. Bermond, L. Gargano, P. Hell, S. Pérennes, U. Vaccaro, "Graph Problems Arising from Wavelength-Routing in All-Optical Networks", 2nd IEEE Workshop on Optics and Computer Science (WOCS, part of IPPS'97), IEEE Press, April 1997.

[Bermond96] J.-C. Bermond, L. Gargano, S. Perennes, A. A. Rescigno, and U. Vaccaro, "Efficient collective communication in optical networks", Proc. of ICALP'96. Lecture Notes in Computer Science, 1996, 574-585

[Boden95] N.J. Boden, et al., "Myrinet - A gigabit per second local area network," *IEEE Micro*, February 1995, pp. 29-36.

[Brauss99] Stephan Brauss, Communication Libraries for the Swiss-Tx Machines, EPFL Supercomputing Review, Nov 1999, pp. 12-15. *http://sawww.epfl.ch/SIC/SA/publications/SCR99/scr11-page12.html*

[Brelaz79] Daniel Brelaz, "New Methods to Color the Vertices of a Graph", *CACM(22)*, 1979, 251-256.

[CERN01] Large Hadron Collider, Computer Grid project, CERN, Sept. 2001, *http://press.web.cern.ch/Press/Releases01/PR10.01EGoaheadGrid.*

[CPLEX02] *ILOG CPLEX 8.0. User's Manual*. ILOG SA, Gentilly, France, 2002.

[Campers88] G. Campers and O. Henkes and J. P. Leclerq "Graph Coloring Heuristics: A Survey, Some New Propositions and Computational Experiences on Random and Leighton's Graphs" Proc. Operational Research, 1988, 917-932

[Caragiannis02] I. Caragiannis and Ch. Kaklamanis and P. Persiano, "Wavelength Routing in All-Optical Tree Networks: A Survey", *Bulletin of the European Association for Theoretical Computer Science*, Vol. 76, 2002, 104-112.

[Chan01] S.-H.G. Chan, "Operation and cost optimization of a distributed server architecture for on-demand video services", IEEE Communications Letters, Vol. 5, No. 9, Sept. 2001, 384-386.

[Chiu89] Dah-Ming Chiu, Raj Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks", Computer Networks and ISDN Systems, Vol. 17, 1989, 1-14.

[Colajanni99] M. Colajanni, B. Ciciani, F. Quaglia, "Performance Analysis of Wormhole Switching with Adaptive Routing in a Two-Dimensional Torus", Proc. Euro-Par'99, Toulose, France, Spinger-Verlang, August-September 1999.

[Duato99] J. Duato, A. Robles, F. Silla, R. Beivide, "A comparison of router architectures for virtual cut-through and wormhole switching in a NOW environment", IEEE Symposium on Parallel and Distributed Processing SPDP, 1999, 240 -247

[EWSD04] Siemens Carrier Networks, EWSD Digital Switching System, April 2004, http://www.icn.siemens.com/carrier/products/switching/ewsdsw.html.

[Fourer03] R. Fourer, D. M. Gay, B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, Thomson Learning Brooks/Cole, 2003, ISBN: 0-534-38809-4

[Gabrielyan03] Emin Gabrielyan, Roger D. Hersch, Network Topology Aware Scheduling of Collective Communications, ICT03 - 10th International Conference on Telecommunications, February 23 - March 1, 2003, pp. 1051-1058, ISBN: 0-7803-7661-7

[Gruber00] Ralf Gruber, "Commodity computing results from the Swiss-Tx project Swiss-Tx Team", Grid Computing Meeting, 2000

[H323] H.323 Standards, *http://www.openh323.org/standards.html*

[Halmos74] Paul R. Halmos, *Naive Set Theory*, Springer-Verlag New York Inc, ISBN 0-387-90092-6, 1974, 26-29.

[Horst95] R. Horst, "TNet: A Reliable System Area Network", IEEE Micro, vol. 15, no. 1, February 1995, pp. 37-45.

[Jagannathan02] S. Jagannathan, A. Tohmaz, A Chronopoulos, H.G Cheung, "Adaptive admission control of multimedia traffic in high-speed networks", Proc. of IEEE International Symposium on Intelligent Control, Oct. 27-30, 2002, 728-733

[Jain91] R. Jain, G. Sasaki, "Scheduling packet transfers in a class of TDM hierarchical switching systems", IEEE International Conference on Communications ICC'91, Vol. 3, 1991, 1559-1563.

[Kuonen99] P. Kuonen, "The K-Ring: a versatile model for the design of MIMD computer topology", Proc. of the High-Performance Computing Conference (HPC'99), San Diego, USA, 381-385, April 1999

[Leelahakriengkrai03] R. Leelahakriengkrai, R. Agrawal, "Scheduling in multimedia CDMA wireless networks", IEEE Transactions on Vehicular Technology, Vol. 52, Issue 1, Jan. 2003, 226-239

[Liu01] Pangfeng Liu, Jan-Jan Wu, Yi-Fang Lin, Shih-Hsien Yeh, "A simple incremental network topology for wormhole switch-based networks", Proc. 15th International Parallel and Distributed Processing Symposium, 2001, 6-12.

[Loh96] P.K.K. Loh, Wen Jing Hsu, Cai Wentong, N. Sriskanthan, "How network topology affects dynamic loading balancing", IEEE Parallel & Distributed Technology: Systems & Applications, Vol. 4, No. 3, Fall 1996, 25-35.

[Maach04] Abdelilah Maach, Gregor v. Bochmann, Hussein Mouftah, "Contention avoidance in optical burst switching", International Conference on Networking, 2004, 1-7

[Mandjes02] M. Mandjes, D. Mitra, W. Scheinhardt, "Simple models of network access, with applications to the design of joint rate and admission control", Proc. of IEEE INFOCOM 2002, Vol. 1, June 23-27, 2002, 3-12

[Melamed00] Benjamin Melamed, Khosrow Sohraby, Yorai Wardi, "Measurement-Based Hybrid Fluid-Flow Models for Fast Multi-Scale Simulation", DARPA/NMS BAA 00-18 AGREEMENT No. F30602-00-2-0556, Sept. 2000, *http://204.194.72.101/pub/nms2000sep/UMissouri-KC.pdf*

[Naghshineh93] M. Naghshineh, R. Guerin, "Fixed versus variable packet sizes in fast packet-switched networks", Proc.Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM '93., Networking: Foundation for the Future, IEEE Press, Vol. 1, 1993, 217-226.

[Ozbay98] H. Ozbay, S. Kalyanaraman, A. Iftar, "On rate-based congestion control in high-speed networks: Design of an H-infinity based flow controller for single bottleneck", Proc. of the American Control Conference, June 1998, 2376-2380.

[Petrini03] F. Petrini, E. Frachtenberg, A. Hoisie, S. Coll, "Performance Evaluation of the Quadrics Interconnection Network", Cluster Computing 6, 2003, 125-142.

[Quadrics] www.quadrics.com

[Ramaswami97] R. Ramaswami, G. Sasaki, "Multiwavelength optical networks with limited wavelength conversion", Proc. of IEEE Infocom, 1997

[Rexford96] Jennifer Rexford, Kang G. Shin, "Analytical Modeling of Routing Algorithms in Virtual Cut-Through Networks", U. Michigan, 1996.

[SWISSTX99] Pierre Kuonen, Ralf Gruber, "Parallel computer architectures for commodity computing and the Swiss-T1 machine", EPFL Supercomputing Review, Nov 1999, pp. 3-11, *http://sawww.epfl.ch/SIC/SA/publications/SCR99/scr11-page3.html*

[Shin96] K.G. Shin, S.W. Daniel "Analysis and implementation of hybrid switching", IEEE Transactions on Computers, Vol. 45 Issue 6, June 1996, 684-692

[SIP04] SIP Forum, http://www.sipforum.org/

[Sitaram00] Dinkar Sitaram, Asit Dan, *Multimedia Servers*, Morgan Kaufmann Publishers, San Francisco California, 2000, 69-73, ISBN 1-55860-430-8

[Stern99] Thomas E. Stern, Krishna Bala, *Multiwavelength Optical Networks: A Layered Approach*, Addison-Wesley, May 1999, ISBN: 020130967X

[Turner99] Jonathan Turner, "Terabit Burst Switching", Journal of High Speed Networks, vol. 8, no. 1, 1999, 3-16

[Yocum97] K.G. Yocum, J.S. Chase, A.J. Gallatin, A.R. Lebeck, "Cut-through delivery in Trapeze: An Exercise in Low-Latency Messaging", 6th IEEE International Symposium on High Performance Distributed Computing, 1997, 243-252.

[Yoo99] M. Yoo, C.Qio, "Optical Burst Switching [OBS] - A New Paradigm for an Optical Internet", Journal of High Speed Networks, vol. 8, no. 1, 1999, 68-84

Appendix. Comparison of efficiency of liquid scheduling algorithm with Mixed Integer Linear Programming

The problem of liquid scheduling can be formulated with Mixed Integer Linear Programming (MILP), see [CPLEX02], [Fourer03]. The network is represented as a directed graph $G = (V(G), E(G))$. The objective is to minimize the number T of timeslots and/or wavelengths required to carry out a given set of transfer requests. We may formulate this as follows:

Minimize: T

subject to:

$$\sum_{s,d} A_t^{s,d} \cdot R_e^{s,d} \le 1 \qquad \forall e \in E(G), \forall t \in \{1 \ldots T\}$$

$$R_e^{s,d} = 0, 1 \qquad A_t^{s,d} = 0, 1$$

Here $R_e^{s,d}$ denotes the routing, i.e. it indicates if the transfer (flit stream flow for wormhole switching or lightpaths for optical networks) from source *s* to destination *d* traverses the link *e*. $A_t^{s,d}$ indicates if the transfer from source *s* to destination *d* is assigned to the time slot *t*.

Since a traffic is partitioned into time frames of a schedule, i.e. each transfer of a traffic must be assigned to one and only one time slot, then

$$\sum_{t=1}^{T} A_t^{s,d} = 1 \qquad \forall s, d$$

The present problem is hard to solve with MILP. For the 362 test bed topologies introduced in section 8, we compared Mixed Integer Linear Programming (MILP) with liquid scheduling. The computation speed of MILP was far below that of our liquid scheduling algorithm (Fig. 15). Our algorithm is on average about 4000 times faster than MILP.

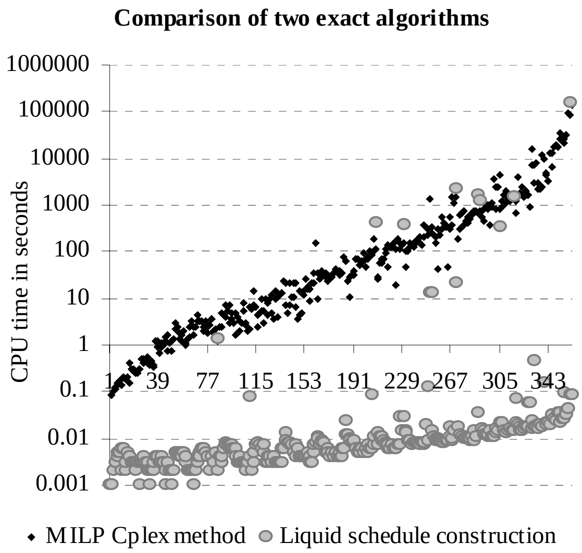


Fig. 15. Running times for computing liquid schedules by MILP and by our optimized liquid schedule computation method.