

Network's Liquid Throughput: Topology Aware Optimization of Collective Communication

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

Abstract

The upper limit of the network's capacity for a collective communication is its liquid throughput related to the flow capacity of a liquid in a network of pipes. However, in a communication network the aggregate throughput of a traffic carried out by 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, the loss of performance is caused by congestions between simultaneous transfers sharing a common link resource. We propose to carry out the transfers of the traffic according to special liquid schedules, which, relying on the knowledge of the underlying network topology and by ensuring successive utilization of bottleneck links, obtain the network's liquid throughput. To build a liquid schedule we need to partition and distribute the traffic into time-frames consisting of mutually non-congesting transfers and also keeping the bottleneck links occupied all the time. Time frames filled up until saturation by non-congesting transfers lead to an exponential explosion of many unsuccessful configurations, resulting in a very large search space. In this paper we present an efficient algorithm which non-redundantly traverses all possible subsets of simultaneous transfers for time-frames and, further on, a liquid schedule construction technique, which efficiently reduces the search space without affecting the solution space. There is a practical minority of collective communications yielding no liquid throughput for any resequencing of its transfers. The optimal or sub optimal schedules for such communications we propose to obtain by applying one of the heuristic graph coloring algorithms.

1. Introduction

Collective multicast communications are of increasingly high importance in multimedia scientific applications. The aggregate throughput of a collective communication

pattern depends on the application's underlying network topology. The amount of data that has to pass across the most loaded links of the network (bottleneck links) gives the utilization time of the bottleneck links. The total size of the traffic divided by the utilization time of the bottleneck links gives an estimation of the *liquid throughput* corresponding to the flow capacity of a non-compressible fluid in a network of pipes [Melamed]. In the wormhole electronic networks as well as in the WDM optical networks not any combination of transfer requests may be carried out simultaneously. For a physical network modelled as an undirected graph $G = (V(G), E(G))$ there is an objective to minimize the number T of timeslots and/or wavelengths required to carry out the given set of transfer requests. The proposed formulation is as follows:

Minimize: T

subject to:

$$\sum_{s,d} A_t^{s,d} \cdot R_e^{s,d} \leq 1 \quad \forall e \in E(G), \forall t \in \{1 \dots T\}$$

$$F_e^{s,d} = 0, 1 \quad A_t^{s,d} = 0, 1$$

Here $R_e^{s,d}$ denotes the routing, i.e. 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 .

It remains to express the partitioning constraint of the schedule, i.e. each transfer of traffic must be assigned to one and only one time slot:

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

The present problem is hard to solve. We will show that for the sizes of practical interest the application of the exact method MILP [CPLEX], [Fourer] (Mixed Integer Linear Programming) to the problem leads to very long solution

times. Application of heuristic method of graph colouring algorithms efficiently providing a sub optimal solution is possible. However the sub optimal results of heuristic graph colouring method, translated back to the original problem in term of performance results to a loss of up to 18%. The exact method proposed in this paper, is fast enough to allow real time scheduling of an evolving over time broadcast and multicast communication in congestion sensible transmission networks. There are numerous applications requiring highly efficient network resources: parallel acquisition and distribution of multiple video streams [Chan], [Sitaram]; switching of simultaneous voice communication sessions [H323], [Fritz]; and high energy physics data acquisition and transmission from a large number of detectors to a cluster of processing nodes for data filtering and event assembling [CERN].

An application of a particular interest is an optical network capable of the lightpath assignment on demand. Liquid scheduling is successfully tested on the cluster of parallel supercomputer interconnected by a high performance wormhole [Liu] switch fabric. A communication intensive application of a direct interest for applying networks's liquid scheduling optimization is a high performance distributed I/O for parallel applications, such as Striped File I/O¹ [SFIO], Petal Frangipani scalable virtual disk space [Thekkath] (e.g. for out of core parallel applications).

2. Applicable networks

This section introduces different types of electronic, optical, wireless networks having in common the problem of congestion avoidance in the traffic. Liquid scheduling of collective broadcast and multicast communication is applicable for each of these networks.

2.1. Cut-through switching

In many high performance multicomputer communication networks links lying on the path of a message are exclusively kept occupied during the transmission of that message. Unlike packet switching (or store-and-forward switching) with each network packet being solely present at intermediate router [Ayad], wormhole switching (or cut-through switching) 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 cut-through switching [Duato], [Shin], [Rexford],

[Colajanni] a message entering into a network is being broken up into small parts of equal size (e.g. one 32-bit word) called flits (standing from flow-control digit), which 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 available². Occupied channels are not released. A channel is released only when the last tail flit of the message is transmitted through it. Thus each link laying on the path of the message is kept occupied during the whole transmission time of a message.

Compared with store and forward switches, cut-through switching considerably decreases the latency of message transmission across multiple routers. Cut-through switching makes the latency insensitive to the message distance. Most contemporary research and commercial multicomputer switches use some form of cut-through switching, e.g. Myrinet [Boden], fat tree architecture of Quadrics [Petrini], [Quadrics], Tnet [Horst], [Brauss].

Wormhole switching only pipelines message and thus requires no more than a very small buffer. This makes easier realization of a high performance switch on a single chip considerably reducing the cost of a large scale network [Yocum]. 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 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.

1.2. 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 optical network.

1. Striped File I/O for parallel programs was our first application which indicated the importance of the knowledge of the underlying network topology by the parallel application.

2. If the message encounters a busy outgoing link virtual cut-through (VCT) switching buffers the entire packet in the router.

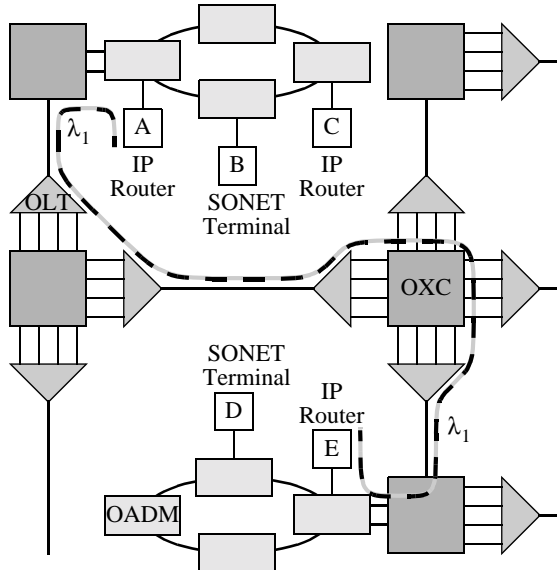


Fig. 1. Optical layer, wavelength-routing network.

OLT, Optical Line Terminal, multiplexes multiple wavelengths into a single fiber and demultiplexes a set of wavelengths on a single fiber into separate fibers; OADM is an Optical Add/Drop Multiplexer; and OXC, Optical Cross-connect, switches wavelengths from one port to another.

End nodes are IP routers and SONET terminals. The network is designed to provide permanent lightpaths between terminal nodes, e.g. between the node A and E. Both OADMs and OXCs in the network may incorporate wavelength conversion capabilities. OXCs that provides no wavelength conversion features are called wavelength-selective cross-connects (WSXC). In networks uniquely based on WSXC optical switches, it is assumed that 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 [Ramaswami]). Assuming that the OXCs of the example in the Fig. 1 do not carry out wavelength conversion, the lightpath between the nodes A and E 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 route¹.

In most cases we are already given the routing, in which case, we are concerned only with the Wavelength Assignment (WA) problem. WA is a problem of optimal

1. In some cases, lightpaths may be converted from one wavelength to another wavelength along their route. OXCs providing the wavelength conversion are called wavelength-interchanging cross-connects (WIXC). WIXCs do both space switching and wavelength conversion.

design in the optical layer. There has been work on theoretical considerations about the required number of wavelengths and the complexity of finding a solution according to the network topology [Bermond], [Caragiannis].

Under wavelength continuity condition, when the network is based on WSXC switches, the problem of wavelength assignment to the set of lightpaths and the problem of the time slot assignment to the set of transmissions are very similar.

Additional interest for liquid scheduling is found in optical transmission systems deploying the capability of dynamic switching of lightpath circuits. The functionality providing the ability to set up and take down lightpaths across the network in a dynamic fashion is currently evolving in the optical layer very fast [Stern]. An issue in such a network providing lightpaths on demand is a fast and efficient scheduling of multicast communications within minimal number of timeframes.

1.3. Time division networks and spread spectrum wireless networks (CDMA).

Satellite-switch time division multiplexing networks [Jain] and CDMA spread spectrum wireless networks [Battiti] also can be viewed as interconnection topologies. The links of the interconnection topology reflecting CDMA networks represent orthogonal frequency spectra. Congestion occurs if there is a common frequency for two simultaneous transmissions. The liquid throughput may be obtained for a traffic over a CDMA wireless network by applying our algorithms to the same radio-communication traffic, but carried out over a virtual interconnection topology reflecting the configuration of the wireless network. The obtained distribution of radio transmissions over successive time frames will yield the liquid throughput of the wireless network.

We propose a dynamic scheduling of multicast communications, where the application is capable of determining its underlying topology and apply liquid schedules to its data intensive collective communication requests. Such a large research subject as the search of a suitable topology for the application is not covered in our work [Puente]. We assume that the application has an access to information on the interconnection network, such that based on the actual run-time node allocation it may obtain the underlying topology knowledge. Unlike flow control based congestion avoidance mechanisms [Chiu], [Ozbay], [Loh] we establish schedules for the data transfers without trying to regulate the sending nodes' data rate. Liquid schedules do not always exists, in that cases we turn to optimal or sub optimal solutions offered by one of

efficient heuristic algorithms of the graph colouring problem.

We integrated implementations of the presented algorithm as a library with a communication intensive MPI (Message Passing Interface) application running on a cut-through high performance network [Pacheco], [Snir], [Gropp98], [Gropp99], [MPI]. The obtained measurement results are very close to the expected theoretical values.

2. Examples and demonstrations

The example of network topology shown in Fig. 2 consists of ten end nodes $t1...t5, r1...r5$ (henceforth called processors), two wormhole cut-through switches s_a, s_b and twelve unidirectional links $l_{t1}...l_{t5}, l_{r1}...l_{r5}, l_{ab}, l_{ba}$ having identical throughputs. In this example the processors $t1...t5$ only transmit data and $r1...r5$ only receive. It's easy to guess the routing, e.g. a message from $t4$ to $r3$ traverse links l_{t4}, l_{ba}, l_{r3} and l_{r3} , a message from $t2$ to $r3$ uses only links l_{t2} and l_{r3} , etc.

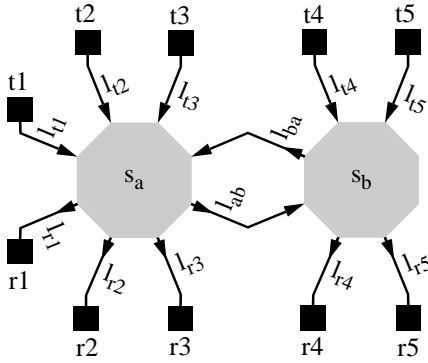













Fig. 2. Example of a network topology.

We denote transfers symbolically to mark out the network links occupied by the transfer. For example the transfer from $t4$ to $r3$ is symbolically represented as , the transfer from $t1$ to $r2$ as , etc. Extending the graphical representation a set of transfers carried out simultaneously we also denote symbolically, e.g.  corresponds to a traffic simultaneously transferring messages from $t4$ to $r3$ and from $t1$ to $r2$.

Let each sending processor have a messages destined to each receiving processor and let all messages have identical sizes [Naghshineh]. Thus we have 25 transfers to carry out. Each of the ten links $l_{t1}...l_{t5}, l_{r1}...l_{r5}$ carries 5 transfers and the two links l_{ab}, l_{ba} must carry 6 transfers each. Therefore the links l_{ab}, l_{ba} are the network

bottlenecks and have the longest active time. If the duration of 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 the identical transfer size and link throughput constraints (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 single link throughput, where $\#(X)$ is the total number of transfers and $\Lambda(X)$ is the number of transfers carried by one bottleneck link.

Now let us see if the order in which the transfers are carried out in this wormhole network can have any impact to 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, then to the receiving processor staying in the next position, etc. (the order turns from the last fifth processor to the first one). This round robin schedule consists of 5 phases. The transfers of the first  second  and fifth  phases of the round-robin schedule may be carried out simultaneously, but the third  and fourth  phases contain congesting transfers, e.g. at the third phase the transfer  stays blocked until the transmission  is accomplished (or vice a versa). 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. Consecutively 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 liquid throughput.

Nevertheless, the 25 transfers may be carried out within 6 time-frames. We call a *liquid schedule* the schedule yielding the liquid throughput of the collective communication. The following sequence of time-frames  is an example of the liquid schedule for the 25-transfer collective communication request.

3. Definitions

The method we propose allows efficient construction of liquid schedules for complex network topologies resulting in considerable increase of collective data exchange throughputs, compared with traditional topology-unaware techniques such as round-robin or random schedules. This section introduces the definitions that will be further used in this paper for presentation of the underlying algorithms of the 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. 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. the collective data exchange). Fig. 3 shows the traffic for a data exchange carried out on a network topology shown in the Fig. 2. In the figure the bottleneck links of the network are marked in bold.

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

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

This half-all-to-all data exchange is a particular case of a traffic, any collective exchange comprising of transfers between possibly overlapping sets of sending and receiving processors is a traffic. 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. If they don't use a common link they are *simultaneous*. 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 is obviously can be carried out in one timeframe required by a single transfer. $\lambda(l, X)$, the *load* of link l in the traffic X , is the number of transfers in X using link l . Now comes the formal definition of $\Lambda(X)$ already introduced before. The *duration* $\Lambda(X)$ of a traffic X is the

maximal value of the load among all links involved in the traffic.

$$\Lambda(X) = \max_{\left\{ l \in \bigcup X \right\}} \lambda(l, X)$$

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 .

Let us 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 $\mathfrak{R}(X)$ and $\mathfrak{S}(X)$ be respectively the sets of all full simultaneousities 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. partitioning of the traffic into teams [Halmos]). To cover the whole solution space we need means of generating 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.

4. Obtaining full simultaneousities

The construction of liquid schedules requires the ability of traversing the set of all full teams of an arbitrary traffic. To limit redundant search steps, each full team should be constructed once and only once. We first optimize the retrieval of all simultaneousities 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 simultaneousities 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 simultaneousities $\mathfrak{R}(X)$ is partitioned into two non-overlapping subsets: an x -positive and x -negative subset of $\mathfrak{R}(X)$. For example, if y is another transfer, the set of x -positive full simultaneousities may be further partitioned into y -positive and y -negative subsets. Iteration of this concept allows us to design a recursive technique traversing whole set of all full simultaneousities $\mathfrak{R}(X)$ 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) and the transfers of X non-congesting with the includer are either in the depot or in the excluder.

We 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 $\phi(R)$.

The category $(\emptyset, X, \emptyset)$ is a *prim-category* since it covers all full simultaneities of X , i.e. $\phi(\emptyset, X, \emptyset) = \mathfrak{R}(X)$.

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 x -positive and x -negative subsets of a coverage of R respectively are coverages of two categories derived from R : a positive sub category and a negative sub category of R .

The positive sub category R_{+x} is formed from the category R by adding transfer x to its includer, and removing from its depot and excluder¹ all transfers congesting with x . The negative sub category R_{-x} is formed from the category R by moving transfer x from its depot to its excluder. Fig. 4 and Fig. 5 show an example of fission of a category into positive and negative sub categories.

$$R = \left\{ \begin{array}{ll} \{\Theta\} & \text{includer} \\ \{\Xi, x, \Xi, \Theta\} & \text{depot} \\ \{\Xi, \Theta\} & \text{excluder} \end{array} \right\} \rightarrow \left\{ \begin{array}{l} R_{+x} \\ R_{-x} \end{array} \right.$$

Θ - denotes any transfer non-congesting with x

Ξ - denotes any transfer congesting with x

Fig. 4. Fission of a an initial category into two sub categories. Symbol Ξ Xi, represents any transfer that is in congestion with x and symbol Θ Theta signify 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 some transfer x taken from the depot.

$$R \rightarrow \left\{ \begin{array}{l} R_{+x} = \left\{ \begin{array}{ll} \{\Theta, x\} & \text{includer} \\ \{\Theta\} & \text{depot} \\ \{\Theta\} & \text{excluder} \end{array} \right\} \\ R_{-x} = \left\{ \begin{array}{ll} \{\Theta\} & \text{includer} \\ \{\Xi, \Xi, \Theta\} & \text{depot} \\ \{\Xi, \Theta, x\} & \text{excluder} \end{array} \right\} \end{array} \right.$$

Fig. 5. Fission of a 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: $\phi(R_{+x}) \cup \phi(R_{-x}) = \phi(R)$, where the coverages of the sub categories have no common transfers, $\phi(R_{+x}) \cap \phi(R_{-x}) = \emptyset$. The replacement of a category R by its two sub categories R_{+x} and R_{-x} is defined as a *binary fission* of a category.

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. Finally, 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 carries out recursively the fission of categories and yields all full simultaneities without repetitions.

There is a further optimization to be considered. Full simultaneities covered by a category have no transfer from the category's excluder. Therefore each full simultaneity covered by a category must contain a congesting transfer for each member of the excluder. Since we keep in the excluder transfers which do not congest with the includer, congesting transfers must be taken from the depot. A category whose depot doesn't have a congesting transfer

1. Since transfers congesting with x anyway can not be in a full simultaneity covered by R_{+x} , we may safely remove them from the excluder.

for at least one of the excluder's transfers is *blank*. The coverage of a blank category is empty and there is therefore no need to pursue its fission.

Let a category within X be *idle* if its includer and its depot together don't use all bottlenecks of X . The coverage of an idle category does therefore not contain a team.

An algorithm that is carrying out successive fissions, starting from the prim-ancestor and contiguously removing all the blank and idle categories ultimately leads to all full teams.

5. Speeding up full team formation

This section presents a further method for speeding up the search for all full teams $\mathfrak{S}(X)$ 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 *skeleton* of X . We denote the skeleton of X as $\zeta(X)$. Obviously, $\zeta(X) \subset X$.

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.

Let us obtain all full teams of the traffic's skeleton by applying the fission algorithm 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

$\mathfrak{S}(X)$, by carrying out the following steps:

1. Obtain the set of the skeleton's full teams $\mathfrak{S}(\zeta(X))$ 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 required 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 32 node cluster computer (see Fig. 10) which will be introduced further in section 7. The search space is given

by the number of nodes that are being explored within the recursion tree. Fig. 6 shows the obtained search space reductions compared with a naive algorithm that would build full teams without any of the proposed optimizations. The skeleton algorithm reduces on average the search space to 12.48%, i.e. full teams are computed 8 times faster than without search space reduction techniques. Note that in the above comparison even the naive algorithm has implemented the most important coverage partitioning strategy, i.e. all presented algorithms, including the naive algorithm, are smart enough to avoid repetitions of full simultaneities.

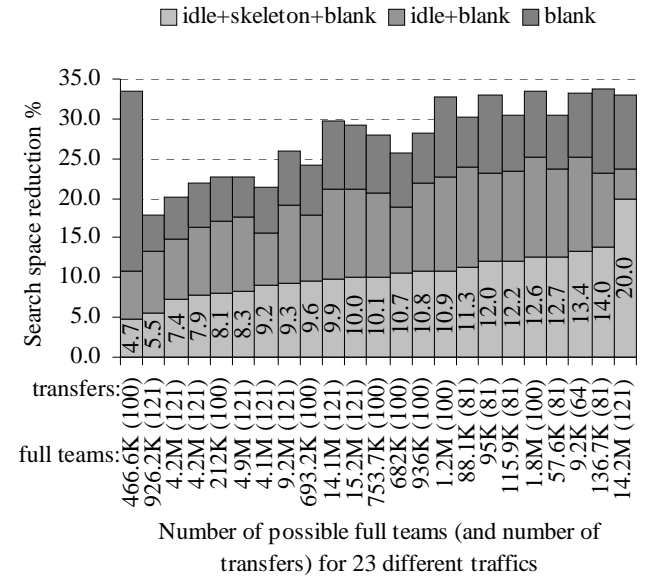


Fig. 6. Search space reduction obtained by idle+skeleton+blank optimization.

6. Liquid schedules

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

DEFINITIONS. A *partition* of X is a disjoint collection of non-empty subsets of X whose union is X [Halmos]. A *schedule* α of a traffic X is a collection of simultaneities of X partitioning the traffic X . A *time frame* of a schedule α is an element of the schedule α . $\#(\alpha)$, the *length* 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

of the traffic then the schedule is *liquid*, i.e. α is liquid if $\#(\alpha) = \Lambda(\bigcup \alpha)$.

A liquid schedule is optimal, but the inverse is not always true, meaning that a traffic may not have a liquid schedule. Fig. 7. demonstrates a simple traffic with three bottleneck links. However this traffic have no team and therefore no liquid schedule.

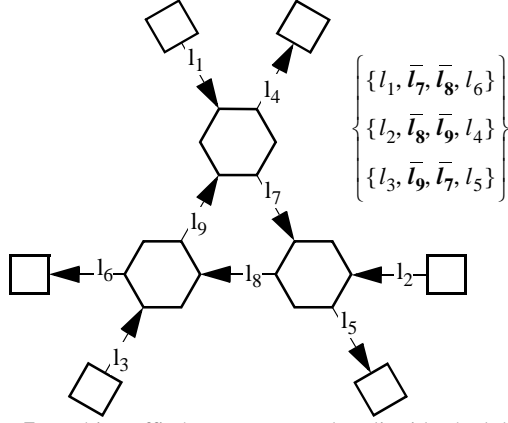


Fig. 7. This traffic has no team and no liquid schedule.

In larger scale networks of practical importance any dead locking in the attempts of simultaneous occupation of bottleneck links arisen at some final stage of the schedule construction should be avoided by rearrangement of the previously scheduled transfers.

In Annex A, the problem of finding a sub optimal schedule is represented as a graph colouring problem [Beaquier]. Annex B, overviews the formulation of the problem in MILP [CPLEX], [Fourer] and the performance results are shown.

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

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

Fig. 8. A liquid schedule of the collective traffic shown in Fig. 2.

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

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 . Therefore we have:

THEOREM 1. The necessary and sufficient condition for the liquidity of a schedule α on X is that each time frame of α be a team of X .

6.1. Liquid schedule naive search algorithm

First we propose a simple technique for construction of a liquid schedule and then in the next subsection we introduce an optimization considerably improving the efficiency of the algorithm.

Conducted by Theorem 1, 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 , its all possible teams A_1, A_2, \dots, A_n which could be taken as the schedule's first time frame. $X - A_1, X - A_2, \dots$ is the variety of possible sub traffics remaining after the choice of the first time. Each of the possible sub traffics, X_i , remaining after the choice of the first time frame has its own set of possibilities to the second time frame $\aleph(X_i) = \{A_{i,1}, A_{i,2}, A_{i,3}, \dots\}$. The choice of the second candidate yield yet another time reduced sub traffic (see Fig. 9).

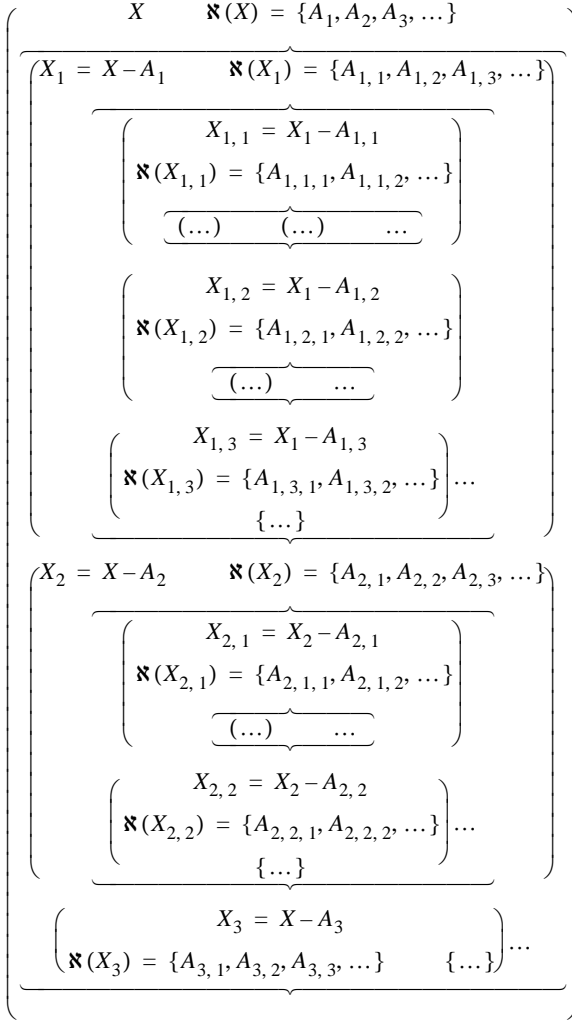


Fig. 9. Liquid schedule search tree. $X_{i_1 i_2 \dots i_n}$ denotes a reduced sub traffic at the layer $n + 1$ of the three and $A_{i_1 i_2 \dots i_n i_{n+1}}$ denotes a prospective candidate for the time frame $n + 1$. The operator \mathfrak{N} applied to a sub traffic X_{sub} represents the set of all possible candidates for a time frame at the present stage.

Dead ends are possible if there are no choice for the successive 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 sub traffic appear to be alike the one shown in Fig. 7. Once a dead end is faced, construction is backtracked to change one of the preceding choices. The construction recursively advances and backtracks until a valid liquid schedule is built¹.

In the naive algorithm for obtaining the liquid schedule, we iteratively explore the full solution space conducted by

the necessary condition of the choice of teams. The choice of possible time frames at any remaining sub traffic X_{sub} is any team of the original traffic X that can be built using the remaining transfers. Therefore the present algorithm is graphically represented in the Fig. 9 with an assumption that $\mathfrak{N}(X_{sub}) = \{A \in \mathfrak{Z}(X) \mid A \subset X_{sub}\}$, where operator \mathfrak{Z} denotes the set of all teams (not necessarily full).

TO CONCLUDE. Since, in order to be liquid it is necessary that each time frame of a schedule α be a team of X , we apply recursive construction algorithm of Fig. 9 assuming that $\mathfrak{N}(X_{sub}) = \{A \in \mathfrak{Z}(X) \mid A \subset X_{sub}\}$ and therefore covering whole solution space and thus ultimately leading to a liquid schedule if it exists.

6.2. Liquid schedule search space reduction

In Fig. 8. remove few first time frames of the liquid schedule and look at the reduced sub traffic. Load of bottleneck link decreases by the number of removed time frames, however additional bottleneck links are emerging. Processing of each time frame of a liquid schedule may result for the reduced traffic in additional bottleneck links (bared and written in bold in Fig. 8), e.g. from time frame 3 on, links $\overline{l_{t3}}$ and $\overline{l_{r3}}$ 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 sub traffic. Bellow we prove that for the liquidity of a schedule, it is necessary for a time frames to be not only a team of the original traffic but also a team of the reduced sub traffic.

THEOREM 2. Let α be a liquid schedule on X and A be a time frame of α . Then $\alpha - \{A\}$ is a liquid schedule on $X - A$.

PROOF. Clearly A is a team of X . Removal of team A from X forms a new traffic $X - A$. Transfers of A used all bottleneck links of X . Therefore the removal of A obligatorily decreased the load of all bottleneck links of X by 1 and the load of the other links either by 0 or by 1. Even with decreased load, the links which were bottlenecks for X , are still the most loaded links for $X - A$, since their load was decreased by only one and the load of other links is not increased. Recall that the duration of X , $\Lambda(X)$ is defined to be the load of its bottlenecks. Therefore for any

1. Solution is found when the transfers remaining in the reduced traffic all form one single team, the last time frame of the liquid schedule.

team A of X , $\Lambda(X-A) = \Lambda(X) - 1$. The schedule α without time frame A , $\alpha - \{A\}$, is a schedule for $X-A$, such that $\#(\alpha - \{A\}) = \Lambda(X-A)$, i.e. the length of the schedule is equal to the duration of the traffic. Therefore $\alpha - \{A\}$ is a liquid schedule on $X-A$.

In other words, if the traffic has a liquid schedule, then a schedule reduced by one team is a liquid schedule on the reduced traffic. The repeated application of Theorem 2 implies that any non-empty subset of a liquid schedule is a liquid schedule, i.e. if $\beta \subset \alpha$ and α is liquid schedule on $\bigcup \alpha$, then β is also liquid on $\bigcup \beta$.

If a time frame of a schedule is a team of original traffic but is not a team of a sub traffic obtained by removal of some of the other time frames, then this schedule may not be liquid. Then 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 sub traffic. Since the reduced sub traffic contains extra bottleneck links, there are less teams of the reduced sub traffic than teams of the original traffic included in the reduced sub traffic. The liquid schedule of Fig. 8. is built as explained above. Each its successive time frame being a team of reduced traffic incorporates all bottlenecks of that reduced traffic.

TO CONCLUDE. We apply recursive construction algorithm of Fig. 9 assuming that $\aleph(X_{sub}) = \mathfrak{Z}(X_{sub})$ instead of $\{A \in \mathfrak{Z}(X) \mid A \subset X_{sub}\}$. By doing so, we considerably reduce the search space without affecting the solution space.

6.3. Liquid schedule construction speed up

In this subsection we show that a liquid schedule can be successfully built by limiting the choice of teams of the reduced sub traffic only by its full teams.

Let us modify a liquid schedule so as to convert one of its teams into a full team. Let a traffic X have 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 not full team A of X_{sub} at any stage of construction, then necessarily the liquid schedule could have been also built by a choice of a full team A of X_{sub} , such that $A \subset A$. 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.

TO CONCLUDE. We apply recursive construction algorithm of Fig. 9 assuming that $\aleph(X_{sub}) = \mathfrak{Z}(X_{sub})$

instead of $\mathfrak{Z}(X_{sub})$.

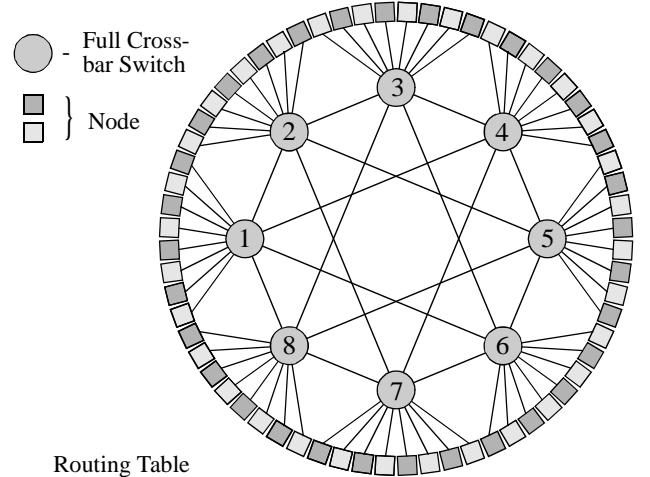
The expression bellow summarizes the search space reduction mechanisms presented in this section.

$$\{A \in \mathfrak{Z}(X) \mid A \subset X_{sub}\} \supset \mathfrak{Z}(X_{sub}) \supset \mathfrak{Z}(X_{sub})$$

Thanks to this speed up for traffic patterns consisting of thousands transmissions the liquid schedule is found in fraction of seconds.

7. Testbed and measurements

In this section we present a testbed consisting of sample traffic patterns for various topologies. Measurements of collective data exchange throughputs will help us to validate the efficiency of our scheduling strategy.



Routing Table								
	1	2	3	4	5	6	7	8
1	x	✓	2	✓	4	✓	8	✓
2	✓	x	✓	7	✓	3	✓	5
3	2	✓	x	✓	4	✓	8	✓
4	✓	7	✓	x	✓	7	✓	3
5	4	✓	4	✓	x	✓	6	✓
6	✓	3	✓	7	✓	x	✓	1
7	8	✓	8	✓	6	✓	x	✓
8	✓	5	✓	3	✓	1	✓	x

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

Fig. 10. 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 henceforth T1, see Fig. 10). The network of the T1 forms a K-ring [Kuonen], [Sayoud] and

has a static routing scheme. The throughputs of all links are identical and equal to 86MB/s . The cluster consists of 32 nodes, each one comprising 2 processors [SwissT1], [Gruber] (i.e. 64 processors).

The sample traffic patterns are selected from different configurations of half-all-to-all collective data exchanges between a set of sending and receiving processors, where each sending processor effect a transmission to each receiving processor (similar to a bi-partite graph). Within each node we allocate one of the processors for transmission and the other one for receiving such that any given allocation of nodes gives an equal number of sending and receiving processors.

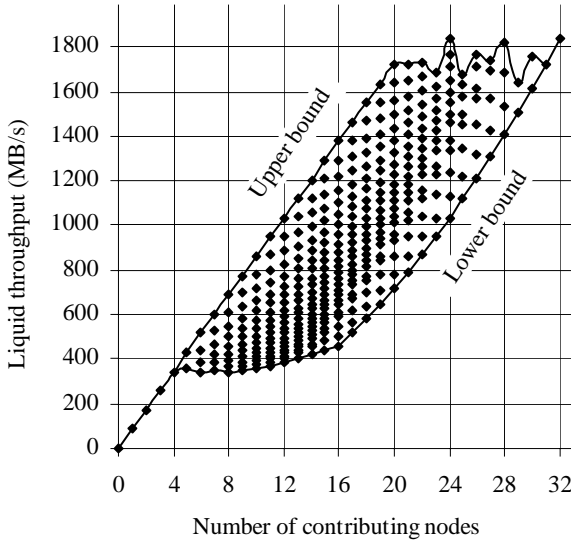


Fig. 11. Liquid throughput in relation to the number of nodes with variations according to sub topologies.

Since the T1 cluster incorporates 32 nodes, there exist $2^{32} = 4294967296$ possible allocations of nodes to an application. Considering only the number of nodes in front of each switch, there are only $5^8 = 390625$ different node allocations, since there are 8 switches, each having from 0 to 4 nodes allocated (i.e. 5 different possibilities). To limit our choice to really different topologies, we've computed the liquid throughputs for each of 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 different liquid throughput values were obtained. We have extracted a set of 362 different topologies one for each value of the liquid throughput. Fig. 11. shows these 362 sub topologies, each one being characterized by the

number of contributing nodes and by its liquid throughput. Depending on the sub topology, the liquid throughput for a given number of nodes may considerably vary.

These 363 sub-topologies may be placed on one axis, sorted first by the number of nodes and then according to their liquid throughput. For each sub-topology, Fig. 12 shows the theoretical liquid throughput and the throughput measured with a topology-unaware round-robin schedule.

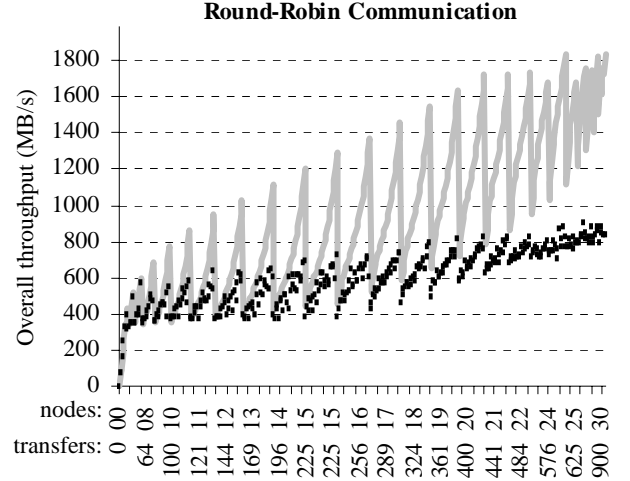


Fig. 12. Theoretical liquid throughput and measured round-robin schedule throughput for 362 network sub topologies. Black dots “ \blacksquare ” represent the measured throughput of collective communication carried out by a round-robin schedule, the gray line “—” shows the upper limit of network capacity, i.e. the network's liquid throughput.

For each measurement, the amount of data transferred from a transmitting processor to a receiving processor is equal to 2MB . For each topology, 20 measurements were made. The black dots represent the median of the collected results. For many sub-topologies, the theoretical liquid throughput is twice as high as the round-robin throughput. This clearly shows that topology-unaware scheduling techniques do not utilize efficiently the potential throughput capabilities offered by the communication network. Throughputs of collective exchanges carried out according to a random schedule do not perform better.

The 362 sample traffic patterns were then scheduled by our liquid scheduling algorithms. The traffic for each pattern was effected over the network according to the computed schedules. Overall throughput results are measured and presented in Fig. 13. The curve of the theoretical value is given for comparison.

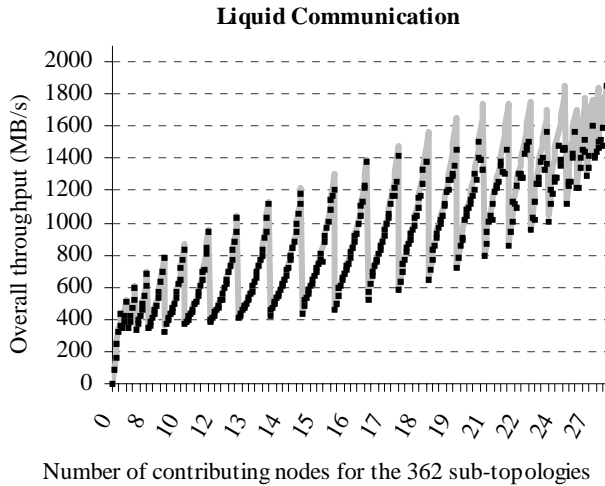


Fig. 13. Predicted liquid throughput and measured throughput according to the computed liquid schedule. The gray line “—” shows the theoretical limit of the network capacity, black dots “■” represent the measured throughput of collective communication scheduled by our method.

Each black dot represents the median of 7 measurements. Processor to processor transfers have a size of 5MB. The measured aggregate throughputs (black dots) are very close to the theoretically expected values of the liquid throughput (gray curve). For many sub topologies, the proposed scheduling technique allows to increase the aggregate throughput by a factor of two compared with topology-unaware round-robin scheduling (Fig. 12).

Thanks to the presented search space reduction algorithms, the computation time of a liquid schedule takes for more than 97% of the considered sub topologies of the T1 cluster less than 1/10 of a second on one Compaq Alpha 500MHz computer.

For applications having relatively long communication patterns such as exchanges of continuous media streams, the gain in the utilization of network resources may be significant compared with the resources required to compute a liquid schedule.

8. Conclusion

If transmissions of a traffic are not properly scheduled, in many high performance cluster networks based on cut-through wormhole switch fabrics and wavelength division multiplexing optical networks a significant performance drop is observed due to congestions between transfers sharing common communication links. We propose a method for fast scheduling of collective communication yielding an aggregate throughput equal to the network's

theoretical capacity's upper limit: liquid throughput of the network. Fast processing of the collective communication is achieved by an algorithm having the time required for the overall transmission broken into time frames, over which, all the transfers of the traffic are to be allocated. In order to process one time frame in the transmission period of a single transfers, time frames must not contain congesting transfers sharing a common link. To ensure the shortest possible processing time frames must envelope as many transfers as possible and further all bottleneck links of the network must be kept busy by each time frame. We proposed a technique non-redundantly and efficiently traversing all the subsets of simultaneous transfers pre-examined to be suitable for obtaining the minimal number of time frames.

Measurements on the traffic carried out on the various sample sub topologies of the Swiss T1 cluster computer has shown that for the most sub topologies our technique allows to increase the collective communication throughput by a factor between 1.5 and 2.

As an alternative to the search for the liquid schedule, we can colour the conflict graph associated to the set of collective transfers by applying a greedy algorithm (see Annex A). Sub optimal scheduling solutions may be obtained with polynomial computation effort, however for many topologies the loss of the performance may overpass 10%. In a concrete application one may choose to carry out simultaneously the search for a liquid schedule and the colouring of the conflict graph. In the case no liquid schedule is found within a certain limit of time, the sub optimal solution may be adopted.

In the present problem we have been always assuming a given in advance routing. Currently our research is being continued on the problem where the routing constraint is released and the search of the routing leading to the best liquid schedule is also included in the objective. Further, we are particularly interested in methods, leading to multipath routing solutions adjusting additionally the routing in order to increase the traffic's fault-tolerance against link failure events, note, without sacrificing the value of once obtained best liquid throughput.

References

- [Melamed] 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, <http://www.darpa.mil/ito/research/nms/meetings/nms2001apr/Rutgers-SD.pdf>
- [CPLEX] ILOG CPLEX 8.0. User's Manual. ILOG SA, Gentilly, France, 2002.

- [Fourer] R. Fourer, D. M. Gay, B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, Thomson Learning Brooks/Cole, ISBN: 0-534-38809-4, 2003.
- [Chan] 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.
- [Sitaram] Dinkar Sitaram, Asit Dan, *Multimedia Servers*, Morgan Kaufmann Publishers, San Francisco California, ISBN 1-55860-430-8, 2000, 69-73.
- [H323] H.323 Standards, <http://www.openh323.org/standards.html>
- [Fritz] D.A. Fritz, D.W. Moy, R.A. Nichols, "Modeling and simulation of Advanced EHF efficiency enhancements", *Proc. of Military Communications Conference, IEEE MILCOM 1999*, Vol. 1, 354-358.
- [CERN] Large Hadron Collider, Computer Grid project, CERN, 20.09.2001, <http://press.web.cern.ch/Press/Releases01/PR10.01EGoaheadGrid>.
- [Liu] 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.
- [SFIO] E. Gabrielyan, R. Hersch, "SFIO a striped file I/O library for MPI", *Large Scale Storage in the Web*, 18-th IEEE Symposium on Mass Storage Systems and Technologies, April 17-20, 2001, pp. 135-144, ISBN: 0-7695-0849-9
- [Thekkath] C. Thekkath, T. Mann, and E. Lee, "Frangipani: A Scalable Distributed File System", 16th SOSP, ACM, Dec 1997, pp. 224-237.
- [Ayad] 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.
- [Duato] 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
- [Shin] K.G. Shin, S.W. Daniel "Analysis and implementation of hybrid switching", *IEEE Transactions on Computers*, Vol. 45 Issue 6, June 1996, 684-692
- [Rexford] Jenifer Rexford, Kang G. Shin, "Analytical Modeling of Routing Algorithms in Virtual Cut-Through Networks"
- [Colajanni] M. Colajanni, B. Ciciani, F. Quaglia, "Performance Analysis of Wormhole Switching with Adaptive Routing in a Two-Dimensional Torus"
- [Boden] N.J. Boden, et al., "Myrinet - A gigabit per second local area network," *IEEE Micro*, pp. 29-36, February 1995.
- [Petrini] F. Petrini, E. Frachtenberg, A. Hoisie, S. Coll, Performance Evaluation of the Quadrics Interconnection NetworkCluster Computing 6, 125-142, 2003
- [Quadrics] www.quadrics.com
- [Horst] R. Horst, "TNet: A Reliable System Area Network", *IEEE Micro*, vol. 15, no. 1, February 1995, pp. 37-45.
- [Brauss] Stephan Brauss, Communication Libraries for the Swiss-Tx Machines. EPFL Supercomputing Review, Nov 99, pp. 12-15. <http://sawwww.epfl.ch/SIC/SA/publications/SCR99/scr11-page12.html>
- [Yocum] 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.
- [Ramaswami] R. Ramaswami, G. Sasaki, "Multiwavelength optical networks with limited wavelength conversion", *Proc. of IEEE Infocom*, 1997.
- [Bermond] 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*, 574-585, 1996.
- [Caragiannis] 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*, 2002, Vol. 76, 104-112.
- [Stern] Thomas E. Stern, Krishna Bala, *Multiwavelength Optical Networks: A Layered Approach*, Addison-Wesley, ISBN: 020130967X, May 1999
- [Jain] 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.
- [Battiti] Roberto Battiti, Alan A. Bertossi, Maurizio A. Bonuccelli, "Assigning Codes in Wireless Networks: Bounds and Scaling Properties.", *ACM/Baltzer Wireless Networks*, Vol. 5, 1999, 195-209.
- [Puate] V. Puente, C. Izu, J. A. Gregorio, R. Beivide, J. M. Prellero, F. Vallejo, "Improving parallel system performance by changing the arrangement of the network links", *Proc. of the International Conference on Supercomputing*, May 2000, 44-53.
- [Chiu] Dah-Ming Chiu, Raj Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks", *Computer Networks and ISDN Systems*, 1989, Vol. 17, 1-14.
- [Ozbay] 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.
- [Loh] P.K.K. Loh, Wen Jing Hsu, Cai Wentong, N. Srisathan, "How network topology affects dynamic loading balancing", *IEEE Parallel & Distributed Technology: Systems & Applications*, Vol. 4, No. 3, Fall 1996, 25-35.
- [Pacheco] Peter S. Pacheco, *Parallel Programming with MPI*, by Morgan Kaufmann Publishers, pages 137-178, 1997

- [Snir] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, Jack Dongarra, MPI - The Complete Reference, Volume 1, The MPI Core, MIT Press, pages 123-189, 1996
- [Gropp98] William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, Marc Snir, MPI - The Complete Reference, Volume 2, The MPI Extensions, MIT Press, pages 185-274, 1998
- [Gropp99] William Gropp, Ewing Lusk, Rajeev Thakur, Using MPI-2 Advanced Features of the Message-Passing Interface, MIT Press, pages 51-118, 1999
- [MPI] Message Passing Interface Forum, MPI-2 Extensions to the Message-Passing Interface, University of Tennessee, pages 209-300, 1997
- [Naghshineh] 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.
- [Halmos] Paul R. Halmos, *Naive Set Theory*, Springer-Verlag New York Inc, ISBN 0-387-90092-6, 1974, 26-29.
- [Beauquier] 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.
- [Kuonen] 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, April 1999, 381-385.
- [Sayoud] H. Sayoud, K. Takahashi, B. Vaillant, "Designing communication network topologies using steady-state genetic algorithms", IEEE Communications Letters, Vol. 5, No. 3, March 2001, 113-115.
- [SwissT1] Pierre Kuonen, Ralf Gruber, "Parallel computer architectures for commodity computing and the Swiss-T1 machine", EPFL Supercomputing Review, Nov 99, pp. 3-11, <http://sawwww.epfl.ch/SIC/SA/publications/SCR99/scr11-page3.html>
- [Gruber] Ralf Gruber, "Commodity computing results from the Swiss-Tx project Swiss-Tx Team", http://www.grid-computing.net/documents/Commodity_computing.pdf
- [Campers] G. Campers and O. Henkes and J. P. Leclercq "Graph Coloring Heuristics: A Survey, Some New Propositions and Computational Experiences on Random and '{L}eighton's' Graphs" Proc. Operational Research, 917-932, 1988.
- [Hertz] A. Hertz and D. de Werra "Using Tabu Search Techniques for Graph Coloring" in Computing(39) 345-351, 1987.
- [Brelaz] Daniel Brelaz, "New Methods to Color the Vertices of a Graph", CACM(22), 1979, 251-256.

Annex A. Performance loss induced by a graph colouring heuristic algorithm

The search for a liquid schedule requires to partition the traffic [Halmos] into a set of non-overlapping mutually non-congesting transfers. The problem can also be formulated as an intersection graph colouring problem [Campers], [Hertz]. Application of conflict graph colouring is also studied in [Beauquier]. Vertices of the conflict intersection graph represent the transfers of the traffic. Edges between vertices represent congestions between transfers. Two vertices of the conflict graph are joined by an edge if the two corresponding transfers are congesting.

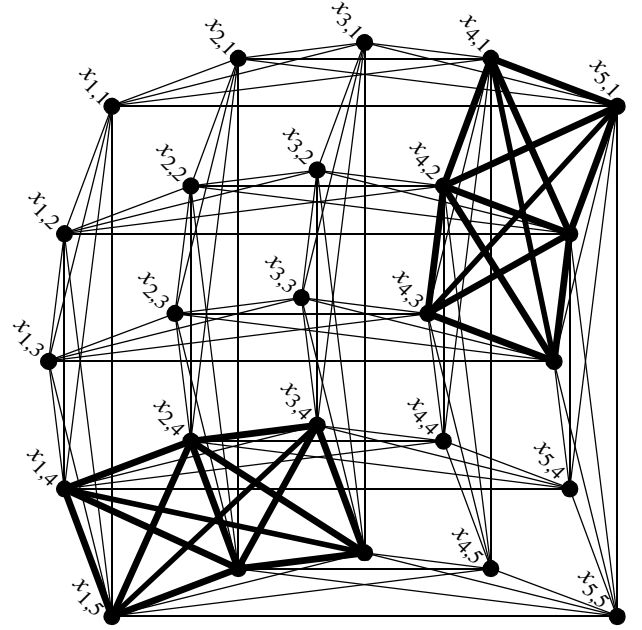


Fig. 14. Conflict graph corresponding to the 25-transfer the traffic of Fig. 2.

Fig. 14 shows the graph corresponding to the data exchange shown in the Fig. 2. The 25 vertices of the graph represent the transfers. The edges represent congestion relation between transfers. Each edge represents one or more communication links shared by two transfers. Bold edges of the figure represent all congestions due to bottleneck links l_{ab} and l_{ba} .

The objective is to colour the vertices of the graph with as few colours as possible, such that no two adjoint vertices have the same colour. Vertex $x_{i,j}$ corresponds to a transfer from the sending processor t_i to the receiving processor r_j . For example vertex $x_{4,1}$ represents the transfer $t4 \rightarrow r1 = \{l_{t4}, l_{ba}, l_{r1}\}$.

Whenever a liquid schedule exists, an optimal solution of the graph colouring problem is a liquid schedule. The chromatic number of the graph's optimal colouring is the

length of the liquid schedule. Vertices having the same colour represent a time frame of the liquid schedule.

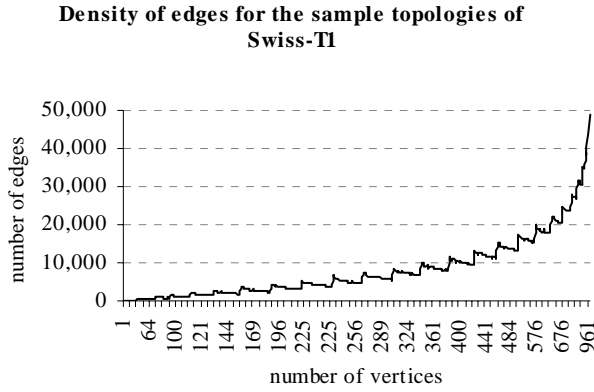


Fig. 15. Characteristics of the graphs corresponding to 362 sample traffic exchanges on the testbed shown in the Fig. 10.

The graph to be coloured is characterized by relatively low density of its edges. Fig. 15 shows the ratio of the number of vertices of a graph to the density of its edges. We can label each edge of the graph by the link(s) causing the congestion. A half-all-to-all data exchange on the Swiss T1 cluster with 32 transmitting and 32 receiving processors forms a graph with $32 \times 32 = 1024$ vertices and 48704 edges.

We compared our method of finding a liquid schedule with the results obtained by applying a greedy high-speed graph colouring algorithm *Dsatur* [Brelaz]. We compared the network performance corresponding to the sub optimal solution of the greedy algorithm with the performance delivered by liquid schedule algorithm. Fig. 16 shows the loss in overall throughput on the Swiss-T1 sample sub topologies due to the redundant unnecessary colours induced by the greedy graph colouring algorithm.

Loss in performance due to suboptimal scheduling

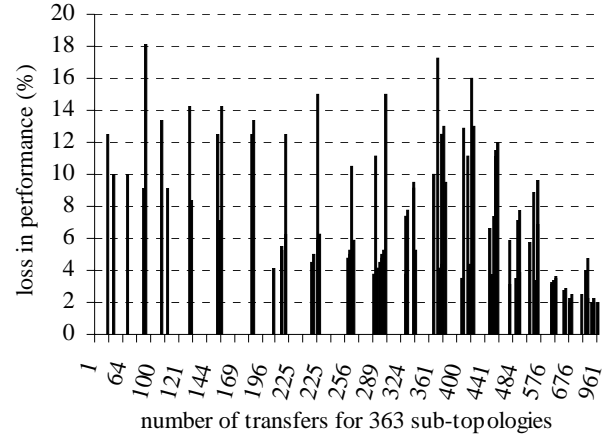


Fig. 16. Loss in performance induced by schedules computed with the *Dsatur* heuristic algorithm.

For 74% of the topologies there is no loss of performance. For 18% of the topologies, the performance loss is below 10% and for 8% of the topologies, the loss of performance is between 10% and 20%. However the computation time of the greedy algorithm is polynomial and compares therefore favourably with the liquid schedule construction algorithm.

Annex B. Comparison of efficiency of liquid scheduling algorithm with MILP

For 362 test bed topologies introduced in the section 7 we applied MILP [CPLEX], [Fourer]. The efficiency of the MILP for this problem was far bellow compared with our liquid scheduling algorithm. The median of the gain factor of our algorithm is about 4000. Fig. 17 shows the computation times required for the optimal scheduling by both MILP and liquid scheduling methods. For the sake of precision it has to be noted that the computation of the optimal scheduling is carried on different processors: liquid scheduling algorithm is benchmarked on Compaq 500MHz Alpha processor and the MILP algorithm on Intel 1400MHz Pentium 4 processor. This difference obviously may not result in a change of any interest in the gain factor.

Comparison of two exact algorithms

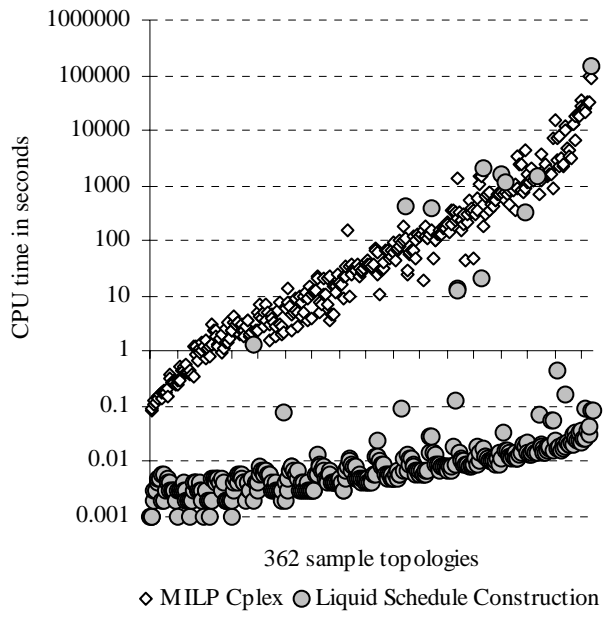


Fig. 17. Comparison of efficiency of two algorithms, MILP method and liquid scheduling method.