Network Topology Aware Scheduling of Collective Communications

Emin Gabrielyan, Roger D. Hersch
{Emin.Gabrielyan,RD.Hersch}@epfl.ch
École Polytechnique Fédérale de Lausanne, CH-1015 Switzerland

Abstract

We propose a method for the optimal scheduling of collective data exchanges relying on the knowledge of the underlying network topology. We introduce the concept of liquid schedules. Liquid schedules ensure the maximal utilization of a network's bottleneck links and offer an aggregate throughput as high as the flow capacity of a liquid in a network of pipes. The collective communication throughput offered by liquid schedules in highly loaded networks may be several times higher than the throughput of topology-unaware techniques. To create a liquid schedule we need to find the smallest partition of all transfers into subsets of mutually non-congesting transfers. The number of combinations of non-overlapping subsets of mutually non-congesting transfers grows exponentially with the number of transfers. We propose several methods to reduce the search space without affecting the solution space. On a real 32 node computer cluster, the measured throughputs of data exchanges scheduled according to our method are very close to the theoretical liquid throughputs.

Keywords: Optimal network utilization, collective data exchange, liquid schedules, network topology, topology-aware scheduling.

1. Introduction

Interconnection topology is one of the key elements determining the global communication throughput. In high-speed networks such as those used in cluster computing [1], [2] and in optical communication networks [3], the network is often formed by a set of full crossbar switches (called optical switches in optical networks). CDMA spread spectrum wireless networks [4] also can be viewed as interconnection topologies whose links represent orthogonal frequency spectra.

Full crossbar switches allow to dynamically route packets from their input ports to their output ports. The crossbar switches we consider are cut-through switches with full $k \times k$ connectivity allowing simultaneous transfers from any input port to any output port.

In the present contribution, we deal with the problem of collective communications through networks made of cut-through switches. We assume that end nodes do not perform store and forward operations. We also assume that the collective communication pattern is known in advance.

The aggregate throughput of a collective data exchange depends on the underlying network topology and on the number of receiving and emitting nodes (end nodes). The total amount of data together with the longest transfer time across the most loaded links gives an estimation of the aggregate throughput. We define this estimation as the liquid throughput of the network. It corresponds to the flow capacity of a non-compressible fluid in a network of pipes [5]. In most networks such as wormhole, cut-through, wavelength division and spread spectrum wireless networks, during a single message transmission, the corresponding link resources (communication circuits, wavelengths and frequency spectra) are kept occupied, causing therefore congestions between concurrent messages sharing a common link resource. Therefore, the aggregate throughput of a collective data exchange may be lower than the liquid throughput. The rate of congestions of a given data exchange may considerably vary according to the order in which the transfers are carried out.

This paper presents an algorithm proposing a schedule of transfers achieving the liquid throughput whenever such a schedule exists (henceforth called liquid schedule). The present approach assumes fixed packet sizes and neglects network latencies. These assumptions are acceptable in low latency networks (e.g. Myrinet) or for networks transmitting long messages (e.g. TDMA/CDMA networks).

We measured the performances of liquid schedules obtained by our scheduling algorithms on a set of real network topologies consisting of cut-through full cross-bar switches and obtained results close to the liquid throughputs.

In the context of wavelength routing in all-optical networks, it was shown that the computation of an optimal routing scheme can be formulated as a graph-coloring problem and that for general types of networks, the problem is NP-hard [6]. However, greedy algorithms exist which provide sub-optimal solutions in polynomial time.

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 [7], [8]. Specific classes of topologies were also analysed in the context of satellite-switch time division multiplexing networks [9].

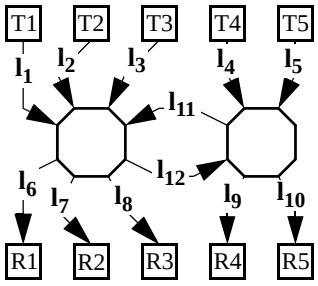
Unlike flow control based congestion avoidance mechanisms [10] [11], we establish schedules for the data transfers without trying to regulate the sending processors' data rate. We specifically address the problem of reaching the flow capacity of a fluid in a network by optimally scheduling the set of transfers of a collective data exchange.

There are numerous applications requiring highly efficient network resources: parallel acquisition and distribution of multiple video streams [12], [13]; switching of simultaneous voice communication sessions [14], [15]; 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 [16].

The solution we propose may also be helpful to schedule collective communications in switched optical networks with wavelength conversion [7], [8]. It may allow within single hop all-optical networks to compute for a given network and routing scheme, both the minimal number of wavelengths to be assigned to individual links and a schedule of a given collective data exchange.

1.1. The scheduling problem

For example, consider the all-to-all collective data exchange shown in Fig. 1.



$$\left\{\begin{array}{c}\{l_1, l_6\}, \{l_1, l_7\}, \{l_1, l_8\}, \{l_1, \mathbf{l_{12}}, l_9\}, \{l_1, \mathbf{l_{12}}, l_{10}\},\\ \{l_2, l_6\}, \{l_2, l_7\}, \{l_2, l_8\}, \{l_2, \mathbf{l_{12}}, l_9\}, \{l_2, \mathbf{l_{12}}, l_{10}\},\\ \{l_3, l_6\}, \{l_3, l_7\}, \{l_3, l_8\}, \{l_3, \mathbf{l_{12}}, l_9\}, \{l_3, \mathbf{l_{12}}, l_{10}\},\\ \{l_4, \mathbf{l_{11}}, l_6\}, \{l_4, \mathbf{l_{11}}, l_7\}, \{l_4, \mathbf{l_{11}}, l_8\}, \{l_4, l_9\}, \{l_4, l_{10}\},\\ \{l_5, \mathbf{l_{11}}, l_6\}, \{l_5, \mathbf{l_{11}}, l_7\}, \{l_5, \mathbf{l_{11}}, l_8\}, \{l_5, l_9\}, \{l_5, l_{10}\}\end{array}\right\}$$

Fig. 1. Example of a collective data exchange composed of 25 transfers.

There are 5 transmitting processors (T1,... T5), each of them sending a packet to each of the receiving processors (R1... R5). The network consists of 12 links. Links l_{11} and l_{12} are the most loaded links, since each of them will be used by 6 transfers. The most loaded links are the bottlenecks of the collective data exchange. They have the longest active time. In the best case, the duration of a collective data exchange is as long as the active time of the bottleneck links.

A *round-robin* schedule is carried out in 5 phases: *(1)* {T1→R1, T2→R2 ... T5→R5}, *(2)* {T1→R2, T2→R3 ... T5→R1}, etc. The round-robin schedule's throughput is lower than the liquid throughput, because bottleneck links l_{11} and l_{12} are idle in phase 1 (Fig. 2). Phases 3 and 4 need to be carried out in two time frames, since they contain congesting transfers.

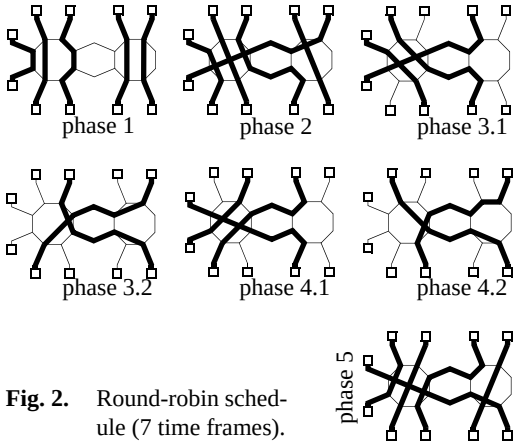


Fig. 2. Round-robin schedule (7 time frames).

Fig. 3 shows that a schedule achieving the liquid throughput for the considered collective data exchange exists.

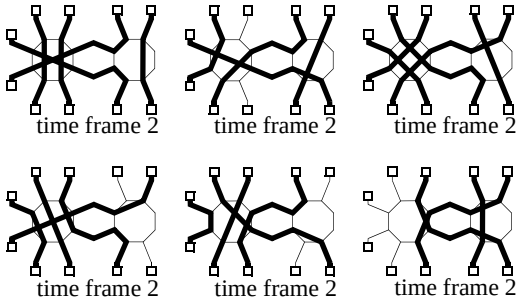


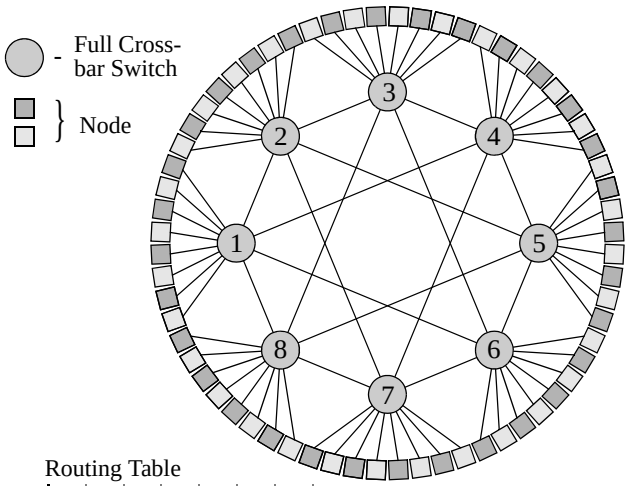
Fig. 3. An optimal schedule (6 time frames).

Section 2 introduces a testbed consisting of 363 different network sub-topologies and represents the liquid throughput as a function of the number of contributing processing nodes and their underlying network topologies. The algorithms for the construction of liquid schedules are presented in section 3. In section 4, we compare for many different sub-topologies the theoretical values of liquid throughputs with the throughputs measured for collective communications scheduled according to our method. In section 5 we draw the conclusions.

2. Throughput as a function of sub-topology

We would like to compare theoretically expected aggregate throughputs with measured throughputs. For benchmarking purposes we need to consider a large variety of network topologies.

Let us form as many distinct network topologies as possible from the set of all sub-topologies of a Swiss-T1 parallel computer cluster (called henceforth T1, see Fig. 4). The network of the T1 forms a K-ring [17] and has a static routing scheme. The throughputs of all links are identical and equal to 86*MB/s*. The cluster consists of 64 processors paired into 32 nodes [18]. For the sake of simplicity, we assume that each node incorporates one transmitting and one receiving processor (Fig. 4).



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. 4. Architecture of the T1 cluster computer.

Since there may be between 0 and 4 allocated nodes in front of each of 8 switches, we have 5^8 = 390625 possible computing node allocations (i.e. 390625 possible sub-topologies). Knowing the routing information, we can compute the liquid throughput of each sub-topology for a collective all-to-all communication pattern.

Because of symmetries, many of these sub-topologies yield an identical liquid throughput. We extract a set of 363 different sub-topologies, which represent all possible liquid throughput values. Fig. 5. shows these 363 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.

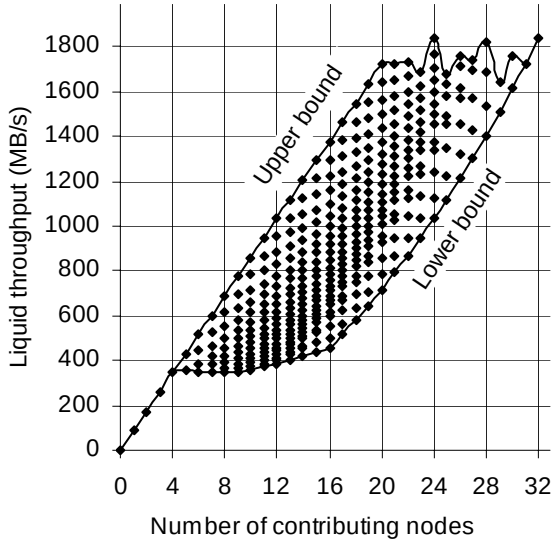


Fig 5. Each of 363 sub-topologies is characterized by its liquid throughput and the number of contributing nodes.

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. 6 shows the theoretical liquid throughput and the throughput measured with a topology-unaware round-robin schedule.

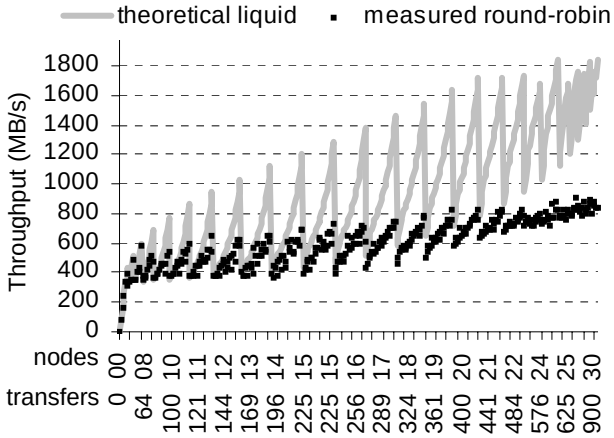


Fig. 6. Theoretical liquid throughput and measured round-robin schedule throughput for 363 network sub-topologies.

For many sub-topologies, the theoretical liquid throughput is twice as large 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.

3. Liquid schedules

This section presents a general method for building on irregular topologies liquid schedules for any collective communication pattern. We neglect network latencies, consider a constant packet size and assume a static routing scheme.

The model of a collective data exchange is introduced by the following formal definitions.

DEFINITIONS. A *transfer* is a set of links (i.e. the links forming the path from a sending processor to a receiving processor). A *traffic* is a set of transfers (i.e. the transfers forming the collective exchange, see Fig. 1). 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 of a traffic X *congest* if they use a common link. A sub-traffic of X (a subset of X) is *simultaneous* if it consists of non-congesting transfers.

A simultaneous subset of a traffic is processed in the time frame of a single transfer. The *load* of a link l in the traffic X is the number of transfers in X using l. The maximally loaded links are called *bottlenecks*. The *duration* $\Lambda(X)$ of a traffic X is the load of its bottlenecks. The size of the traffic $\#(X)$ is the number of its transfers. The *liquid throughput* of a traffic X is the ratio $\#(X)/\Lambda(X)$ multiplied by the single link throughput.

For example, the traffic X shown in Fig. 1 has a number of transfers $\#(X) = 25$ and the duration of the traffic is $\Lambda(X) = 6$. Therefore the aggregate liquid throughput is the ratio $25/6$ of a single link throughput, i.e.

$$(25/6) \times 100MB/s = 416.67MB/s,$$

assuming a single link throughput of 100*MB/s*.

3.1. Partitioning

A *partition* of X is a disjoint collection of non-empty subsets of X whose union is X [19]. A *schedule* α of a traffic X is a collection of simultaneous sub-traffics of X partitioning the traffic X. A *time frame* of a schedule α is an element of the schedule α (i.e. is a simultaneity). The *length* $\#(\alpha)$ of a schedule gives 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*. A liquid schedule is optimal, but the inverse is not always true, meaning that a traffic may not have a liquid schedule. Fig. 7 shows a liquid schedule for the collective traffic shown in Fig 1.

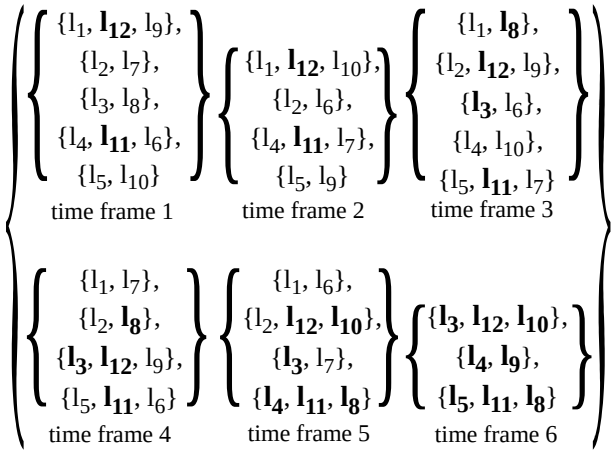


Fig. 7. A liquid schedule for the collective traffic shown in Fig. 1. Bold links in a time frame indicate bottlenecks in the reduced traffic.

In the annex, the problem of finding an optimal schedule is formulated as the problem of coloring a conflict graph [6].

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. Let us define a *team* of a traffic X as a simultaneous subset of X using all its bottlenecks. Therefore 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.

Our strategy for finding a liquid schedule therefore relies on searching for teams of a traffic. Hence, we need to partition the traffic into a set of teams forming the sequence of time frames.

The traffic can be recursively partitioned by an algorithm traversing the search tree in a depth-wise order (Fig. 8). The root of the tree is the original traffic X. Associated to the traffic X is the collection of all possible teams for the first time frame $\{A_1, A_2, \ldots, A_n\}$. The choices of candidate teams $A_1, A_2, \ldots$ yield respectively the remaining sub-traffics $X - A_1, X - A_2, \ldots$. Each of these sub-traffics has its own collection of candidate teams for the next time

frame. As before, members of this collection produce successor sub-traffics at the next level of the tree (Fig. 8).

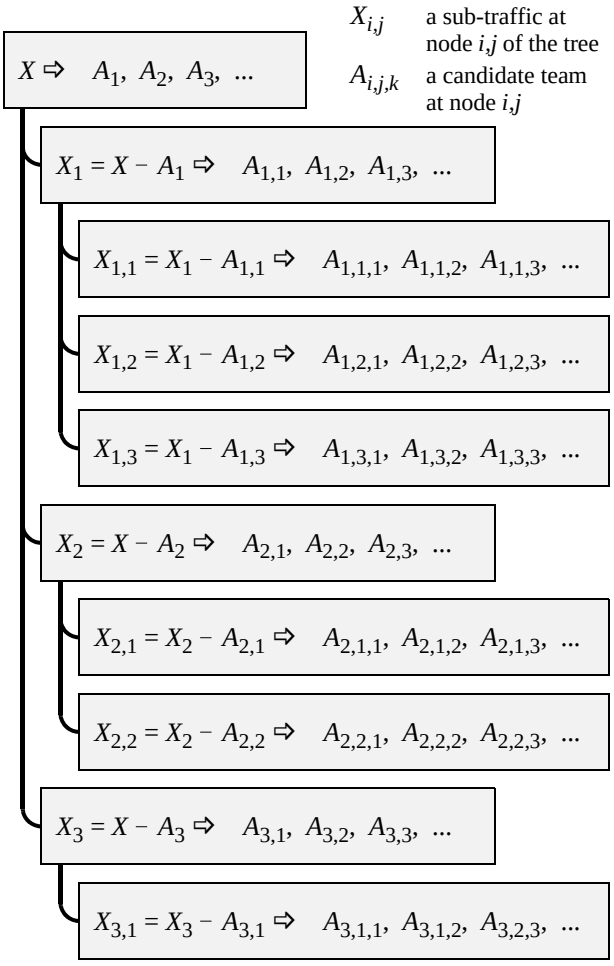


Fig. 8. Liquid schedule search tree. The symbol "⇨" points to all possible time frames for the current reduced traffic.

A possible time frame for each sub-traffic X_{sub} is any team of X formed by not yet carried out transfers $\{A \in \Im(X) | A \subset X_{sub}\}$, where operator $\Im$ associates to a traffic the set of all its teams.

3.2. Reducing the search space

We would like to reduce the search space. For this purpose we introduce two theorems proving that for computing successive time frames, instead of forming teams from the original traffic, we can form teams from the reduced traffic.

Let us show that by removing a time frame (i.e. a team) from a liquid schedule, we form a new liquid schedule on the remaining traffic. Note that the remaining traffic may have additional bottlenecks. For example, in Fig. 7, from time frame 3 on, links $\boldsymbol{l_3}$ and $\boldsymbol{l_8}$ appear as additional bottlenecks. Emerging additional bottlenecks allow us to reduce the search space when creating a liquid schedule.

THEOREM 1. 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. Remove the team A from X so as to form a new traffic $X - A$. The duration of the new traffic $X - A$ is the load of the bottlenecks in $X - A$. Bottlenecks of $X - A$ include the bottlenecks of X. The load of a bottleneck of X is decreased by one in the new traffic $X - A$ and therefore the duration of $X - A$ is the duration of X decreased by one, i.e. $\Lambda(X - A) = \Lambda(X) - 1$. The schedule α without the element A is a schedule for $X - A$ with the previous length decreased by one. Therefore the new schedule $\alpha - \{A\}$ has as many time frames as the duration of the new traffic $X - A$. Hence $\alpha - \{A\}$ 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 1 implies that any non-empty subset of a liquid schedule is a liquid schedule on the correspondingly reduced traffic.

THEOREM 2. If, by traversing each team A of a traffic X none of the sub-traffics $X - A$ has a liquid schedule, then the traffic X does not have a liquid schedule either.

PROOF. Let us suppose that X has a liquid schedule α. Then a time frame A of α shall be a team of X. Further, according to Theorem 1 the schedule $\alpha - \{A\}$ shall be a liquid schedule for $X - A$. Therefore for at least one team A of X the sub-traffic $X - A$ has a liquid schedule. This proves the theorem by contraposition.

Theorem 2 implies that if X has a liquid schedule at least one team A of X will be found, such that the sub-traffic $X - A$ has a liquid schedule β. Obviously $\beta \cup \{A\}$ will be a liquid schedule for X.

Therefore, instead of forming the set of possible time frames by considering teams of the original traffic included in the current sub-traffic X_{sub}, i.e. $\{A \in \Im(X) | A \subset X_{sub}\}$, we propose to form the set of all possible time frames at the current node using all teams of the current sub-traffic, i.e. $\Im(X_{sub})$. Since the teams of the current sub-traffic X_{sub} together with the bottlenecks of the original traffic X must also use the additional bottlenecks of X_{sub}, the number of teams of the current subtraffic $\Im(X_{sub})$ is smaller or equal

to the number of teams of the original traffic whose transfers belong to the current subtraffic, i.e.

$$\#(\Im(X_{sub})) \leq \#(\{A \in \Im(X) | A \subset X_{sub}\}).$$

Therefore less possible teams need to be considered when building the schedule. The solution space is not affected, since theorem 2 is valid at any level of the search tree.

By traversing the tree in depth-wise order, we cover the full solution space. A solution is found when the current node (sub-traffic) forms a single team. The path from the root to that leaf node forms the set of teams yielding the liquid schedule. A node is a dead end if it is not possible to create a team out of that sub-traffic. In that case we have to backtrack to evaluate other choices. Evaluation of all choices ultimately leads to a solution if it exists.

If a solution for X (i.e. a liquid schedule on X) exists, then the algorithm will find it. If the algorithm does not find a solution for X, and since we explored the full solution space, we conclude that X does not have a liquid schedule.

Let us describe a further simple and efficient search space reducing technique.

DEFINITIONS. A simultaneous subset A of a traffic X is *full* with respect to X if each transfer of $X - A$ is in congestion with a transfer of A. A team of X is called *full* team if it is a full simultaneous subset of X.

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 X has a solution then it has also a solution when one of its time frames is full, hence the choice of the teams in the construction may be narrowed from the set of all teams to the set of full teams only. Fig. 7 shows a liquid schedule constructed with full teams.

In each step of the liquid schedule search tree (Fig. 8), we need to traverse the set of all full teams of the current sub-traffic.

For this purpose, we partition the collection of all full teams into two sub-collections, one consisting of full teams including a given transfer and the other one consisting of full teams excluding that transfer. At the next step, each of these two sub-collections is again partitioned to include, respectively exclude another selected transfer. By recursively applying this approach we steadily increase the quantity of sub-collections and decrease their sizes. At any stage, each sub-collection is characterised by the history of previously selected transfers. Nodes which do not represent full teams are eliminated. Ultimately, each leaf node of the partitioning process represents one single full team. All histories of selected transfers recursively obtained by traversing the partitioning tree lead to the enumeration of all possible full teams[1].

In the annex we compare our liquid schedule construction algorithm with a heuristic solution for coloring the conflict graph.

4. Results

In this section, we present the measured throughput values for collective communications carried out on a real network according to the liquid schedules computed with our algorithms. We then compare the measured throughput values with the theoretical liquid throughput values.

Fig. 9 shows the measured aggregate throughputs of collective data exchanges performed according to the computed liquid schedules on the T1 cluster. Each black dot represents the median of 7 measurements. The horizontal axis represents the 363 sub-topologies as well as the number of contributing nodes.

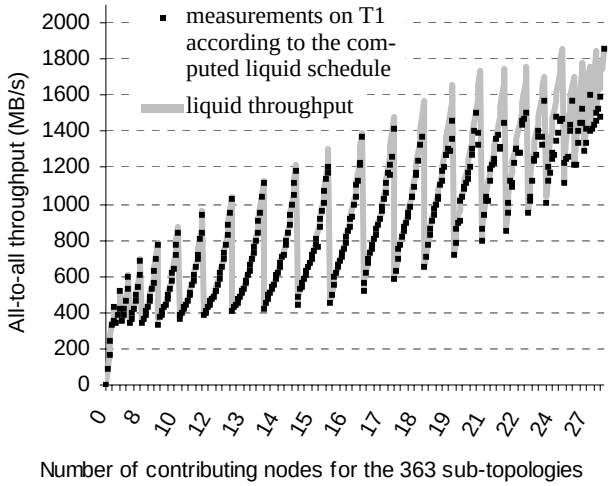


Fig. 9. Predicted liquid throughput and measured throughput according to the computed liquid schedule.

Processor to processor transfers have a size of 5*MB*, transferred as a single message of 5*MB*. The measured all-to-all aggregate throughputs (black dots) are close to the theoretically computed liquid throughput (gray line). For many sub-topologies, the proposed scheduling technique allows to increase the aggregate throughput by a factor of two, compared with the topology-unaware round-robin schedule (Fig. 6).

1. The detailed algorithm, together with further search space reduction techniques will be presented in a subsequent publication.

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 a single Compaq 500MHz Alpha processor.

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.

5. Conclusions

We propose a method for scheduling collective data exchanges in order to obtain an aggregate throughput equal to the network's liquid throughput. This is achieved by building at each time frame of the schedule mutually non-congesting sets of transfers using all bottleneck links. Exploration of the full solution space yields a liquid schedule if it exists. The proposed search space reduction techniques make the approach practical for networks having in the order of ten interconnected crossbar switches.

Experiments carried out on the Swiss T1 cluster computer, show that for most sub-topologies the proposed scheduling technique allows to increase the collective data exchange throughput by a factor between 1.5 and 2.

As an alternative to the search for the liquid schedule, we can color the conflict graph associated to the set of collective transfers by applying a greedy algorithm (see Annex). Slightly sub-optimal scheduling solutions may be obtained without much computation effort. In a concrete application one may choose to carry out simultaneously the search for a liquid schedule and the coloring of the conflict graph. In the case that no liquid schedule is found after a certain limit of time, the sub optimal solution offered by the greedy graph coloring algorithm may be adopted.

In the future, we intend to explore how to extend the presented scheduling techniques in order to dynamically reschedule collective data exchanges when the set of planned exchanges evolves over time.

Annex

The search for a liquid schedule requires to partition the traffic into sets of mutually non-congesting transfers. The problem can be formulated as the problem of coloring the conflict graph [6]. Vertices of the conflict graph are formed by transfers. Edges between vertices represent congestions between transfers.

Fig. 10 shows the graph whose vertices are to be colored for the collective data exchange of Fig. 1. Vertex $x_{n,m}$ corresponds to a transfer from an emitting processor n to a receiving processor m. For example vertex $x_{4,1}$ represents the transfer T4→R1={l_4, $\mathbf{l_{11}}$, l_6}. The bold edges of the graph show congestions of transfers due to specific bottleneck links.

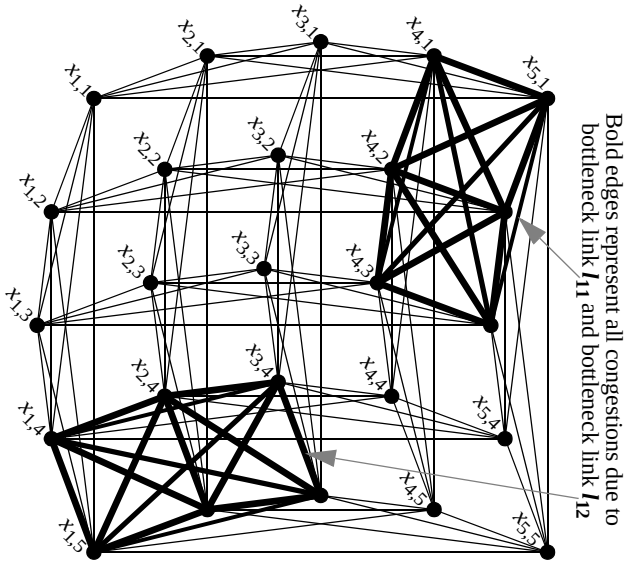


Fig. 10. Graph corresponding to the data exchange shown in Fig. 1. The 25 vertices of the graph represent the transfers. The edges represent congestion relations between transfers, i.e. each edge represents one or more communication links shared by two transfers.

Whenever a liquid schedule exists, an optimal solution of the graph coloring problem is a liquid schedule. The chromatic number of the graph's optimal coloring is the length of the liquid schedule. Vertices having the same color represent a time frame of the liquid schedule.

The graph to be colored is characterised by the relatively low density of its edges. We can label each edge of the graph by the link(s) causing the congestion. An 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 coloring algorithm *Dsatur* [20], which carries out the following operations:

1. Arrange the vertices by decreasing order of degrees.
2. Color a vertex of maximal degree with color 1.
3. Choose a vertex with a maximal saturation degree (defined as the number of different colours to which it is adjacent). If there is an equality, priority is given to the vertex having the maximal degree in the uncoloured subgraph.

4. Color the chosen vertex with the least possible (lowest numbered) color.
5. If all the vertices are colored, stop. Otherwise, return to 3.

Fig. 11 shows the loss of performance on the T1 sub-topologies due to the additional unnecessary colours induced by the greedy graph coloring algorithm, compared with the liquid schedule algorithm.

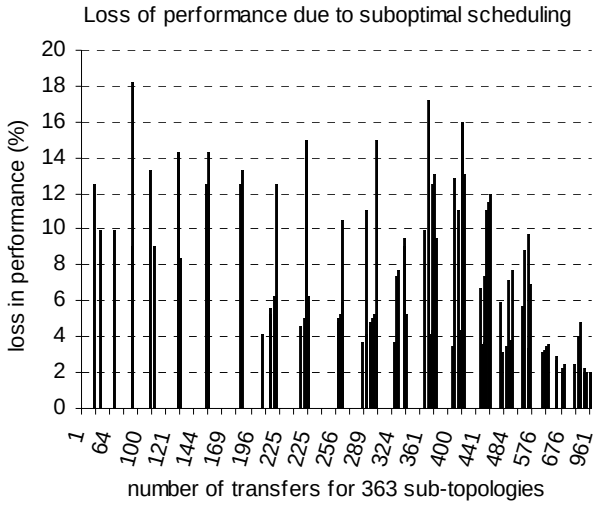


Fig. 11. Loss in performance induced by schedules computed with the Dsatur heuristic algorithm.

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%.

The computation time of the greedy algorithm is polynomial and compares therefore favourably with the algorithm searching for the liquid schedule.

References

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

[2] J. Duato, A. Robles, F. Silla, R. Beivide, "A Comparison of Router Architectures for Virtual Cut-Through and Wormhole Switching in a NOW Environment", *ACM Journal of Parallel and Distributed Computing*, February 2001, Vol. 61, Issue 2, 224-253.

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

[4] 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.

[5] 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*, April 2001.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

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

[15] 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.

[16] Large Hadron Collider, Computer Grid project, CERN, 20.09.2001, *http://press.web.cern.ch/Press/Releases01/PR10.01EGoaheadGrid.html*

[17] 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.

[18] Pierre Kuonen, Ralf Gruber, "Parallel computer architectures for commodity computing and the Swiss-T1 machine", EPFL Supercomputing Review, Nov 99, pp. 3-11, *http://sawww.epfl.ch/SIC/SA/publications/SCR99/scr11-page3.html*

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

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