Submitted to the - 31st Annual Conference - 2002 International Conference on Parallel Processing (ICPP-2002)

Network Topology-aware Traffic Scheduling

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

Abstract

We propose a method for the optimal scheduling of collective data exchanges relying on the knowledge of the underlying network topology. The method ensures a maximal utilization of bottleneck communication links and offers an aggregate throughput close to the flow capacity of a liquid in a network of pipes. On a 32 node K-ring T1 cluster we double the aggregate throughput. Thanks to the presented combinatorial search reduction techniques, the computational time required to find an optimal schedule takes less than 1/10 of a second for most of the cluster's topologies.

Keywords: Optimal network utilization, traffic scheduling, collective communications, collective data exchange, network topology, topology-aware scheduling.

1. Introduction

The interconnection topology is one of the key factors of a computing cluster. It determines the performance of the communications, which are often a limiting factor of parallel applications [1], [2], [3], [4]. Depending on the transfer block size, there are two opposite factors (among others) influencing the aggregate throughput. Due to the message overhead, communication cost increases with the decrease of the message size. However, smaller messages allow a more progressive utilization of network links. Intuitively, the data flow becomes liquid when the packet size tends to zero [5], [6]. In this paper we consider collective data exchanges between nodes where packet sizes are relatively large, i.e. the network latency is much smaller than the transfer time. The aggregate throughput of a collective data exchange depends on the underlying network topology and on the number of contributing processing nodes. The total amount of data together with the longest transfer time across the most loaded links (*bottlenecks*) gives an estimation of the aggregate throughput. This estimation is defined here as the *liquid* throughput of the network. It corresponds to the flow capacity of a non-compressible fluid in a network of pipes [6]. Due to the packeted behaviour of data transfers, congestions may occur in the network and thus the aggregate throughput of a collective data exchange may be lower than the liquid throughput. The rate of congestions for a given data exchange may vary depending on how the sequence of transfers forming the data exchange is scheduled by the application.

The present contribution presents a scheduling technique for obtaining the liquid throughput. In the present paper we limit ourselves to fixed packet sizes and we neglect network latencies. Switches are assumed to be full cross-bar, also with negligible latencies.

There are many other collective data exchange optimization techniques such as message splitting [7], [8], parallel forwarding [9], [10] and optimal mapping of an application-graph onto a processor graph [11], [12], [13]. Combining the above mentioned optimizations with the optimal scheduling technique described in the present article may be the subject of further research. Unlike flow control based congestion avoidance mechanisms [24], [25] we schedule the traffic without trying to regulate the sending processors' data rate.

There are numerous applications requiring highly efficient network resources: parallel acquisition of multiple video streams each one forwarded to a set of target nodes [14], [15], voice-over-data traffic switching [16], [17] and high energy physics data acquisition and transmission from numerous detectors to a cluster of processing nodes for filtering and event assembling [18], [19].

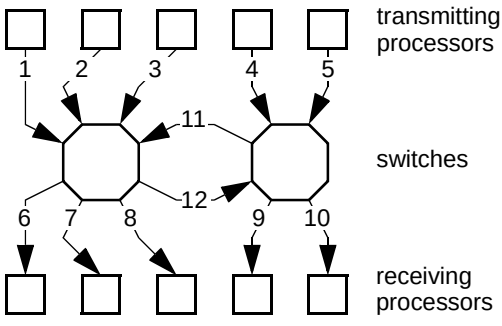


Fig. 1. Simple network topology.

Let us analyze an example of a collective data exchange on a simple topology (Fig. 1). We define an all-to-all data exchange as a collective transfer operation where each transmitting processor sends a packet to each receiving processor. As example we consider an *all-to-all* data exchange with 5 transmitting processors and 5 receiving processors. With a packet size 1*MB*, the data exchange operation transfers 25*MB* of data over the network.

During the collective data exchange, links 1 to 10 transfer 5*MB* of data each (Fig. 1). Links 11 and 12 are bottleneck links and need to transfer 6*MB* each. Suppose that the throughput of a link is 100*MB/s*. Since links 11 and 12 are bottleneck links, the longest transfer of the collective data exchange lasts $6MB/(100MB/s) = 0.06s$. Therefore the liquid throughput of the global operation is $25MB/0.06s = 416.67MB/s$. Let us now propose a schedule for successive data transfers and analyze its throughput.

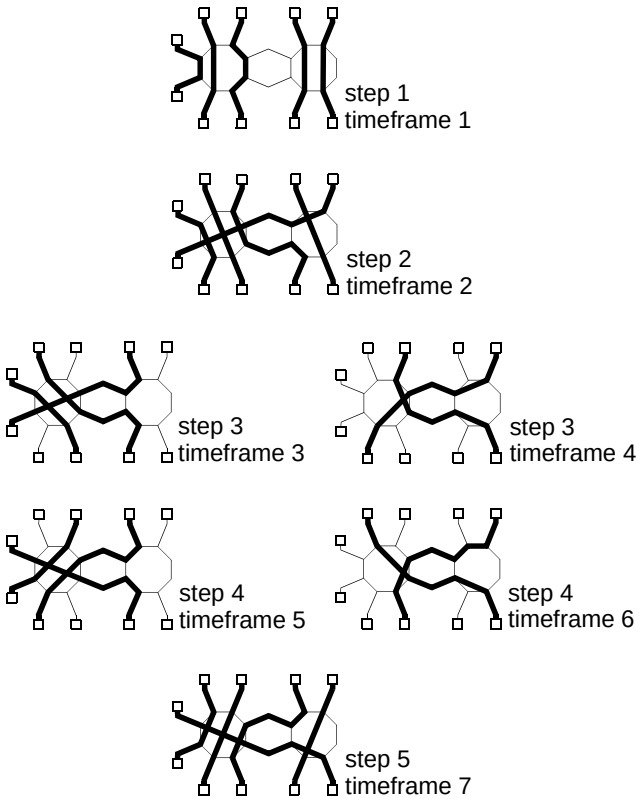


Fig. 2. Round-robin schedule of transfers.

Intuitively, a good schedule for an all-to-all exchange is a *round-robin* schedule where at each step each sender has a receiver shifted by one position. Let us now examine the round-robin schedule of an all-to-all data exchange on the network topology of figure 1. Fig. 2 shows that steps 1, 2 and 5 can be processed in the timeframe of a single transfer. But steps 3 and 4 can not be processed in a single timeframe, since there are two transfers trying to simultaneously use the same links 11 and 12, causing a congestion. Two conflicting transfers need to be scheduled in two single timeframe substeps. Thus the round-robin schedule takes 7 timeframes instead of the expected 5 and accordingly, the throughput of the round-robin all-to-all exchange is:

$25MB/\left(7\times\left(\frac{1MB}{100MB/s}\right)\right) = 357.14MB/s$. It is therefore less than the liquid throughput (416.67*MB/s*). Can we propose an improved schedule for the all-to-all exchange such that the liquid throughput is reached?

By ensuring that at each step the bottlenecks are always used, we create an improved schedule, having the performance of the network's liquid throughput (Fig. 3). According to this improved schedule only 6 steps are needed for the implementation of the collective operation, i.e. the throughput is:

$25MB/\left(6\times\left(\frac{1MB}{100MB/s}\right)\right) = 416.67MB/s$.

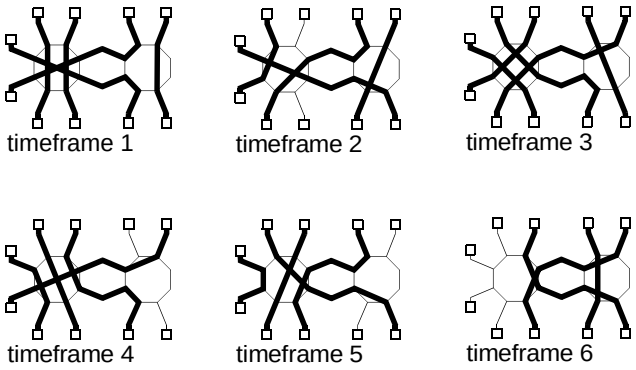


Fig. 3. An optimal schedule.

Section 2 shows how to describe the liquid throughput as a function of the number of contributing processing nodes and their underlying network topologies. An introduction to the theory of traffic scheduling is given in section 3. Section 4 presents measurements for the considered sub-topologies and draws the conclusions.

2. Throughput as a function of sub-topology

In this section we present a test-bed for performance measurements on a variety of topologies, in order to validate the theory presented in section 3.

In order to evaluate the throughput of collective data exchanges we need to specify along an independent axis the

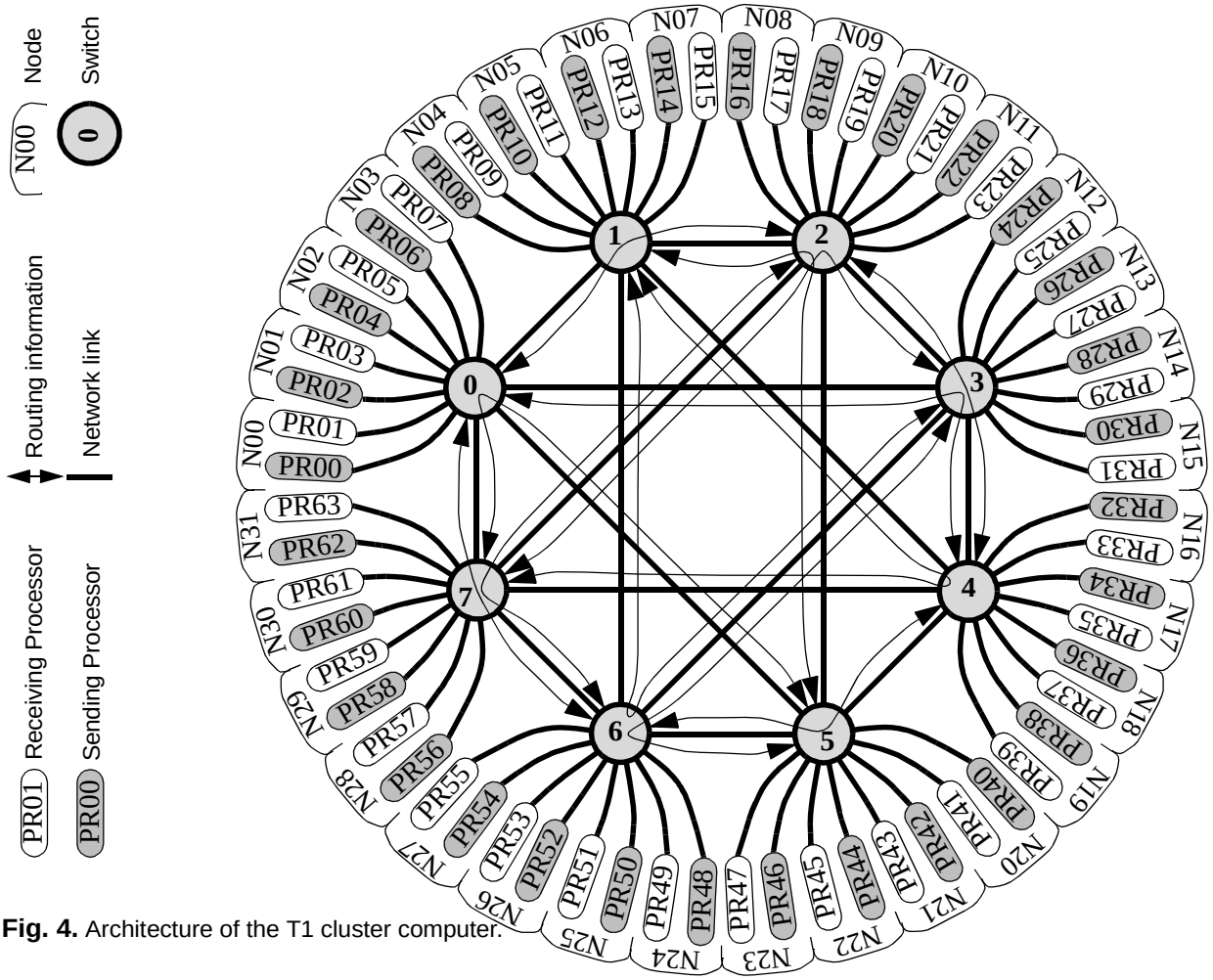


Fig. 4. Architecture of the T1 cluster computer.

number of contributing processing nodes as well as significant variations of their underlying network topologies. To simplify the model let us limit the configuration to an identical number of receiving and transmitting processors forming successions of node pairs. The applications perform all-to-all data exchanges over the allocated nodes (each transmitting processor sends one packet to each receiving processor).

Let us create variations of processing node allocations by considering the specific network of the Swiss-T1 cluster (called henceforth T1, see Fig. 4). The network of the T1 forms a K-ring [20] and has a static routing scheme. The throughputs of all links are identical and are equal to 86*MB/s*. The cluster consists of 64 processors paired into 32 nodes [21], [22].

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 processing node allocations, since there are 8 switches having each n used nodes ($0 \le n \le 4$). Each allocation may be represented by a vector $\left[n_0\ n_1\ n_2\ n_3\ n_4\ n_5\ n_6\ n_7\right]$. With a model incorporating the given network topology and routing tables, we can compute the liquid throughput of an all-to-all traffic for any allocation. The full set of 390625 allocation vectors is given as input to the model and the liquid throughput is computed for each input vector.

For the T1's network, only 363 different values of liquid throughput are formed and thus the set of 390625 is partitioned into 363 different subsets. Each of the obtained 363 key sub-topologies is characterized by its liquid throughput and the number of allocated nodes (see Fig. 5). The figure demonstrates that depending on the sub-topology, the liquid throughput for a given number of nodes may considerably vary.

For the purpose of enumerating the 363 sub-topologies we sort these sub-topologies according to the number of nodes and within the same number of nodes according to the

value of the liquid throughput. Figure 6 demonstrates the liquid throughput of the network together with the throughput of an imaginary full crossbar network. The horizontal axis represents the collection of the 363 sub-topologies together with the number of contributing processing nodes (in parentheses).

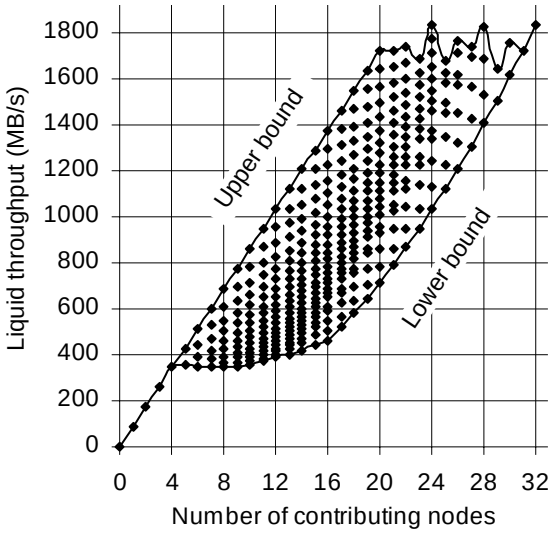


Fig 5. Liquid throughput in relation to the number of nodes with variations according to sub-topologies.

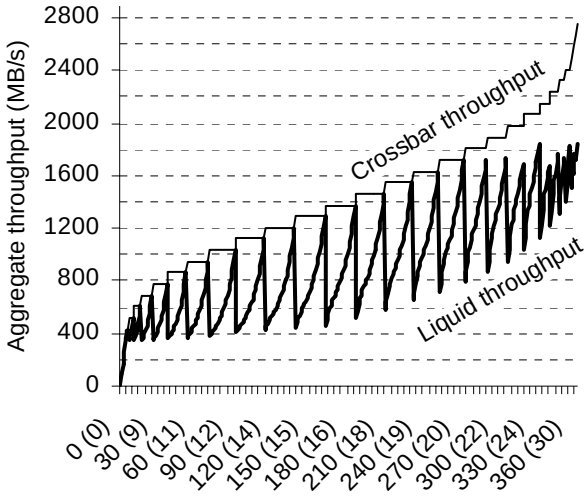


Fig. 6. Liquid and crossbar throughputs on T1.

3. Liquid schedules

This section presents a theory for building liquid schedules on any topology. As in most real computational networks we assume a static routing scheme. The presented theory is valid for any combination of transmitting and receiving processors performing any type of collective exchange (not limited to all-to-all exchanges). We neglect network latencies and assume a constant packet size for all data exchanges.

Let us describe a formal model of a collective data exchange. In this model, a single point-to-point transfer is represented by the set of communication links forming the path between a transmitting and a receiving processor. The collective data exchange comprises a set of transfers having identical packet sizes. A sending processor may transfer a packet to a given receiving processor not more than once.

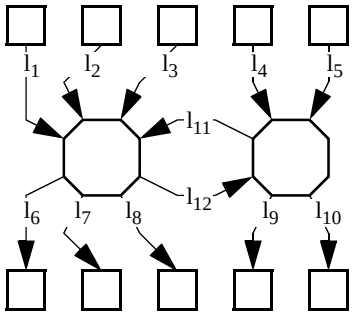
DEFINITIONS. A *transfer* is a set of links (i.e. the links forming the path from a sending processor to a receiving processor). A *traffic* is a set of transfers (i.e. the transfers forming the collective exchange, see Fig. 7). A link l is *utilized* by a transfer x if $l \in x$. A link l is utilized by a traffic X if l is utilized by a transfer of X. Let a and b be transfers of a traffic X, the transfer b is in congestion with a, if b uses a link utilized by a. A sub-traffic of X (a subset of X) is *simultaneous* if it forms a collection of non-congesting transfers.

A simultaneous subset of a traffic is processed in the timeframe of a single transfer. The *load* of 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. Let T_{link} be the throughput of the network's single link. The *liquid throughput* of a traffic X is $\frac{\#(X)}{\Lambda(X)} \cdot T_{link}$. For example, the traffic X shown in Fig. 7 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. $(25/6) \times 100 MB/s$, supposing a single link throughput of 100 *MB/s*.

3.1. Partitioning

Recall that a *partition* of X is a disjoint collection of non-empty subsets of X whose union is X [23]. A *schedule* α of a traffic X is a collection of simultaneous subsets of X (simultaneous sub-traffics of X) partitioning the traffic X. A *step* of a schedule α is an element of α. The *length* $\#(\alpha)$ of a schedule α is the number of steps in α. A schedule of a traffic is *optimal* if the traffic does not have any shorter schedule. By definition, 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. 8 shows a liquid schedule of the collective traffic shown in Fig. 7.



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

Fig. 7. All-to-all traffic.

DEFINITION. A simultaneous subset of X is a *team* of X if it uses all bottlenecks of X.

THEOREM 1.1. Each step of a liquid schedule of a traffic is a team of the traffic.

PROOF. The duration of a traffic X is the load of its bottlenecks. Consider link l as one of the bottlenecks of X. The load of link l is the number of transfers in X using l. Let α be a schedule on X. By definition, α is a collection of simultaneous subsets of X, partitioning X. Since α partitions X, a transfer of X (using l) shall be found in one and only one of the steps of α. Since a step of the schedule α consists of simultaneous transfers, it may contain only one or no transfer using l. Therefore if the length of α is equal to the number of transfers in X using the bottleneck l, then each step of α shall contain a transfer using l. Since this argument is true for any bottleneck, then each step of α shall use all bottlenecks and therefore shall be a team of X.

THEOREM 1.2. A schedule of a traffic where each step is a team of that traffic is a liquid schedule.

PROOF. If each step of α has a transfer using l, then, since the steps of α are simultaneous they shall contain one and only one transfer using l and therefore the length of α shall be equal to the number of transfers using l. Hence if all steps of a schedule use all bottlenecks the duration of the traffic is equal to the length of the schedule, i.e. the schedule is liquid.

Thanks to Theorem 1.1 and Theorem 1.2 we have proven Theorem 1.

THEOREM 1. The equivalent condition for the liquidity of a schedule of a traffic is that each step of the schedule be a team of the traffic.

Our strategy for finding a liquid schedule will therefore rely on searching for teams of a traffic. Hence, we need to partition a traffic by collections of teams (whenever possible).

Let us show that by removing an element (step) from a liquid schedule, we form a new liquid schedule on the remaining traffic. Note that the remaining traffic may have additional bottlenecks (Fig. 8). This property will allow us to reduce the search space for creating a liquid schedule.

THEOREM 2. Let α be a liquid schedule on X and A be a step 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

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

step 1 step 2 step 3 step 4 step 5 step 6

Fig. 8. A liquid schedule for the collective traffic shown in figure 7, with bottleneck links associated to the sub-traffic at the current step in bold.

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 steps 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 2 implies that any non-empty subset of a liquid schedule is a liquid schedule.

3.2. Construction

THEOREM 3. 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 according to Theorem 1, a step A of α shall be a team of X. Further, according to Theorem 2 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 3 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.

Let us give an overall view to the liquid schedule search algorithm. The algorithm recursively searches for a solution by traversing a tree in depth-wise order (see Fig. 9). The root of the tree is the original traffic X. Associated to the traffic is the collection of all possible steps of a liquid schedule $\{A_1, A_2, \ldots, A_n\}$. Successor nodes are formed by subtraffics $X - A_1$, $X - A_2, \ldots$ $X - A_n$. Each of these successor nodes has its own collection of all possible steps. As before, each member of this collection will produce successor nodes at the next level of the tree.

Let us discuss how to characterize the collection of all possible steps for the current node. The sub-traffic of the current node comprises the set of transfers not yet carried out. A possible step of a liquid schedule shall be a subset of the current sub-traffic. Recall that for being liquid it is sufficient that all the steps of a schedule be teams of the original traffic X. Therefore a possible step at each sub-traffic is any team of X formed by not yet carried out transfers, i.e. each team A of the original traffic included in the current sub-traffic $\{A \in \Im(X) | A \subset X_{1,2}\}$, where the operator $\Im$ associates with a traffic the set of all its teams, and where $X_{1,2}$ is an example of a sub-traffic (Fig. 9).

We would like to reduce the search space. Instead of forming the set of possible steps by using teams of the original traffic $\{A \in \Im(X) | A \subset X_{1,2}\}$, we propose to form the set of all possible steps at the current node using all teams of the current sub-traffic, i.e. $\Im(X_{1,2})$. It can be shown that $\Im(X_{1,2}) \subset \{A \in \Im(X) | A \subset X_{1,2}\}$, and consequently $\#(\Im(X_{1,2})) \le \#(\{A \in \Im(X) | A \subset X_{1,2}\})$. Therefore less possible teams need to be considered when building the schedule and the solution space is not affected, since theorem 3 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

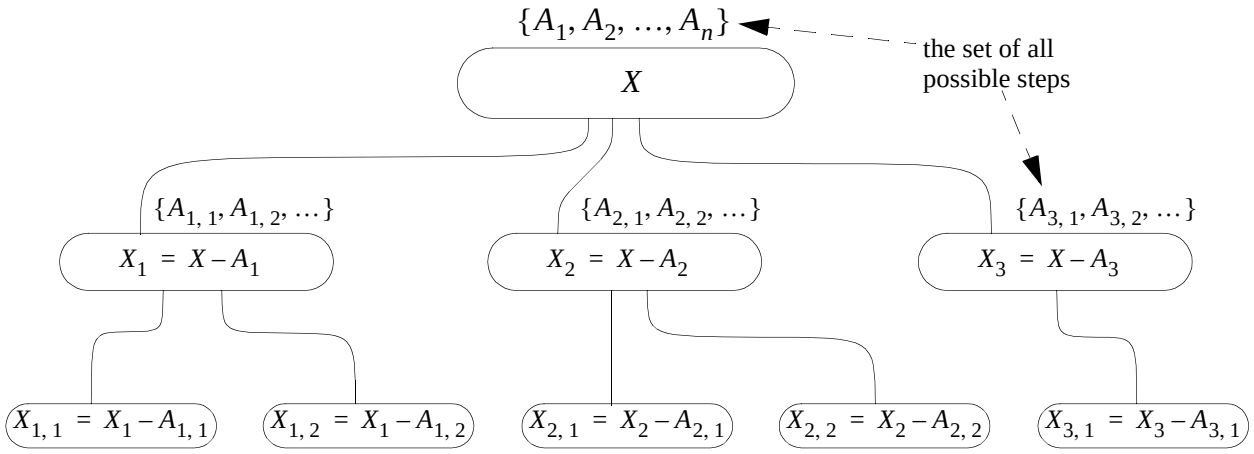
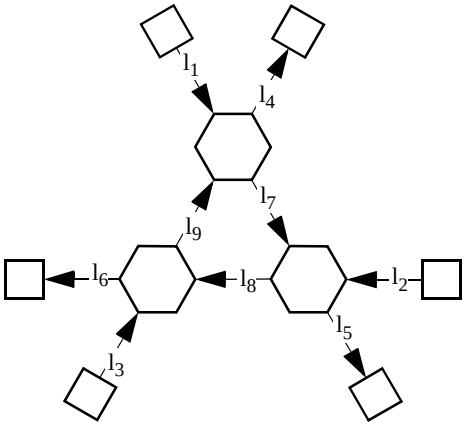


Fig. 9. Liquid schedule search tree.

that leaf node forms the set of teams yielding the liquid schedule. A node presents a dead end if it is not possible to create a team out of that sub-traffic (see Fig. 10). In that case we have to backtrack to evaluate other choices. Evaluation of all choices ultimately leads to a solution if it exists.



$$X = \left\{ \begin{matrix} \{l_1, \mathbf{l_7}, \mathbf{l_8}, l_6\}, \\ \{l_2, \mathbf{l_8}, \mathbf{l_9}, l_4\}, \\ \{l_3, \mathbf{l_9}, \mathbf{l_7}, l_5\} \end{matrix} \right\}$$

Fig 10. No team and no liquid schedule can be found for the traffic *X*.

If a solution exists for *X*, 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*.

We intend to limit the search space when building a liquid schedule. Let us modify a liquid schedule so as to convert one of its teams into a full team. Let *X* (a traffic) have a solution α (a liquid schedule). Let *A* be a step of α. If *A* is not a full team of *X*, then, by moving the necessary transfers from other steps of α, we can convert step *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 steps 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. 8 shows a liquid schedule built as explained above.

In order to be able to explore the full solution space for obtaining a liquid schedule, we need to successively build all full teams. We designed a procedure capable of generating (without repetitions) all full teams for an arbitrary traffic. It first builds *skeletons*, an intermediate collection of teams from a sub-traffic including only those transfers which comprise bottlenecks. Then it extends each skeleton by applying variations of all non-congesting transfers in order to build up all full teams.

4. Results and conclusion

Let us compare the throughputs of the round-robin and the liquid schedules. The measured throughput of the round-robin schedule on a T1 cluster is shown in Fig 11.

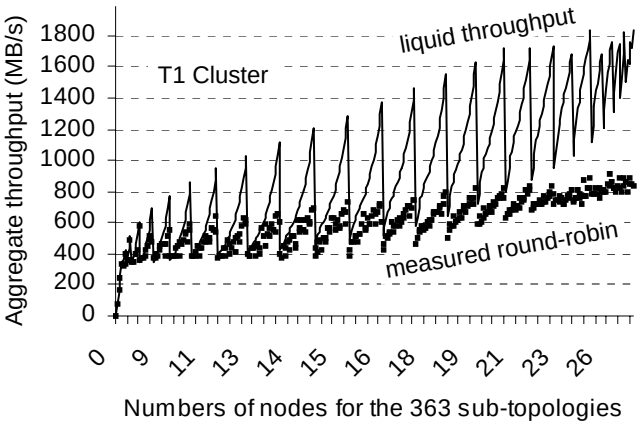


Fig. 11. Throughput of the round-robin schedule.

The amount of data transferred from a transmitting processor to a receiving processor is equal to 2*MB* and the transfer block size is 520*KB*. Fig. 11 presents the result of 4344 measurements for all-to-all data exchanges. For each topology, 20 measurements were performed. The median of

the collected results is represented as a small black square. The continuous curve represents the liquid throughput.

For many sub-topologies, the measured round-robin throughput is only 50% of the liquid throughput.

Fig. 12 shows the measured aggregate throughput of an all-to-all collective traffic executed on T1, optimized by applying our liquid schedule based traffic partitioning technique. 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. 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 a simple round-robin schedule.

Thanks to the presented algorithms, we also strongly reduced the search space of liquid schedules. 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 500MHz Alpha processor.

References

[1] 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.

[2] 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.

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

[4] V. Puente, C. Izu, J. A. Gregorio, R. Beivide, J. M. Prellezo, 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.

[5] 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.

[6] 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*

[7] 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.

[8] 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.

[9] Thilo Kielmann, Henri E. Bal, Sergei Gorlatch, Kees Verstoep, Rutger F.H. Hofman, “Network Performance-aware Collective Communication for Clustered Wide Area Systems”, Parallel Computing, Vol. 27, No. 11, 2001, 1431-1456.

[10] Il Kyu Park, Youngseok Lee, Yanghee Choi, “Stable load control with load prediction in multipath packet forwarding”, Proc. 15th International Conference on Information Networking, 2001, 437-444.

[11] Sibabrata Ray, Hong Jiang, Jitender S. Deogun, “A parallel algorithm for mapping a special class of task graphs onto

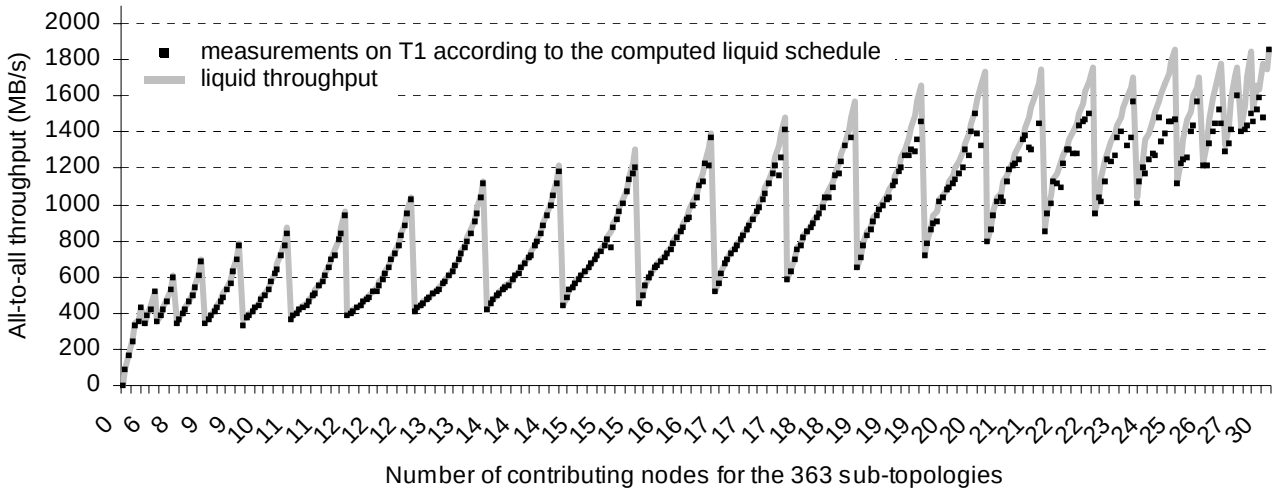


Fig. 12. Predicted liquid throughput and measured throughput according to the computed liquid schedule.

linear array multiprocessors", Proc. of the ACM Symposium on Applied Computing, April 1994, 473-477.

[12] Y. Xie, W. Wolf, "Allocation and scheduling of conditional task graph in hardware/software co-synthesis", Proc. of the Conf. on Design, Automation and Test in Europe (DATE 2001) March 2001, 620-625.

[13] Chiang Chuanwen, Lee Chungnan, Chang Mingjyh, "A dynamic grouping scheduling for heterogeneous Internet-centric metacomputing system", Proc. 8th International Conference on Parallel and Distributed Systems, ICPADS 2001, 77 -82.

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

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

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

[17] 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.

[18] ATLAS Collaboration, CERN, Technical Progress Report, *http://press.web.cern.ch/Atlas/GROUPS/DAQTRIG/TPR/PDF_FILES/TPR.bk.pdf*

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

[20] 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.

[21] 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*

[22] Ralf Gruber, "Commodity computing results from the Swiss-Tx project Swiss-Tx Team", *http://www.grid-computing.net/documents/Commodity_computing.pdf*

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

[24] 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.

[25] 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.