Thin trees: Cost-Effective Tree-like Networks

Javier Navaridas[1], Jose Miguel-Alonso[1], Wolfgang Denzel[2]

Abstract— **Interconnection networks based on the *k*-ary *n*-tree topology are widely used in high-performance parallel computers. However, this topology is expensive and complex to build. In this paper we evaluate an alternative tree-like topology that is cheaper in terms of cost and complexity because it uses fewer switches and links. This alternative topology leaves unused upward ports on switches, which could be rearranged to be used as downward ports. The increase of locality can be efficiently exploited by applications. We test the performance of these thin trees, and compare it with that of regular trees. Evaluation is carried out using a collection of synthetic traffic patterns that emulate the behavior of scientific applications. We also propose a methodology to perform cost and performance analysis of different networks. Our main conclusion is that, for the set of studied workloads, the performance drop in thin trees is very low, while the cost savings can be significant which leads to an increasement in performance/cost efficiency.**

Keywords—**interconnection networks; parallel job scheduling; resource allocation; trace-driven simulation.**

I. INTRODUCTION

Current high-performance computing facilities are composed of thousands of computing nodes executing jobs in parallel. An underlying interconnection network (such as Myrinet [10], Infiniband [7], QsNet [15], or an *ad-hoc* network) provides a mechanism for tasks to communicate. Most of these facilities belong to national laboratories or supercomputing centers, and are shared by many researchers (see the Top500 list [5]).

The *k*-ary *n*-tree topology [16], based on the classical fat-tree topology introduced by Leiserson [8], is often the topology of choice to build low latency, high bandwidth and high connectivity interconnection networks (hereafter IN) for parallel computers. Its main characteristics are the low mean path length and the multitude of paths from a source to a destination node, which increases exponentially with the distance between nodes (in number of hops). This high path diversity provides good communication performance for almost all kind of workloads, independently of their spatial, temporal and length distributions.

A network design that ignores locality could be a good option because it provides high performance even in worst-case scenarios. However, regular parallel applications usually arrange their processes in such a way that communicating processes are kept in close proximity, in order to obtain advantages from locality in communication. For this reason, the highest levels of this topology tend to be infra-utilized when managing application traffic.

Taking this into consideration, we can reduce cost and complexity of the IN by means of reducing the ratio between the number of links connected to the upper tree levels and those connected to the lower ones. This can be done reducing the radix of the switches or, alternatively, increasing the locality by rearranging the upward ports and making them downward. In both cases the cost of the system is reduced: fewer switches, fewer links and, in the former case, switches of lower complexity. If parallel applications exploit locality in communications or do not make intensive use of the network, performance should not suffer. They could even experience an improvement due to the increased locality of the latter case.

In this paper we evaluate thin trees, a network topology directly derived from the *k*-ary *n*-tree topology that uses a constant reduction of the ratio between the number of upward and downward ports in all switches. Therefore the aggregate bandwidth of each level is reduced accordingly to its height: lower levels have more aggregate bandwidth than higher ones. This leads to a reduction of the total amount of switches in the network, which, consequently, reduces cost and complexity of the interconnection network. Ideally we would evaluate performance using real traces taken from actual scientific applications running on very large systems but, as large traces are difficult to obtain and not very manageable, we have used a collection of synthetic workloads that emulate their behavior. This mimicry is done not only in terms of spatial patterns, but also in terms of the causality of the injected messages.

We have selected some instances of the topologies under study, fed the simulator with the proposed workloads and measured their performance. A comparison of alternatives is done taking into consideration raw performance and performance/cost efficiency. As performance is application-dependent, we define a model to compute a performance indicator that can be tailored to fit the characteristics of a given supercomputing center. We will see that, in terms of this indicator, the *k*-ary *n*-tree shows its superiority as a general-purpose topology, although slimmed topologies perform equally well for some relevant application mixes. If cost is considered too, the complexity of the *k*-ary *n*-trees plays against them and the thin tree is the clear winner: cost is lower and performance is good.

The rest of this paper is organized as follows. Section II provides the motivation of this work. In Section III we present the topologies that we will evaluate. The experimental environment (topologies, switches and workloads) is explained in Section IV. In Section V we show the results of the experimental work and analyze them taking into account the raw performance as well as the performance/cost efficiency. To do so, we propose performance and cost functions. Some related work is discussed in Section VI. Finally, we close this work with some concluding remarks in Section VII.

[1] Department of Computer Architecture and Technology, The University of the Basque Country UPV/EHU. Contact E-mail: {javier.navaridas, j.miguel}@ehu.es

[2] IBM Research GmbH, Rüschlikon, Switzerland. Contact E-mail: wde@zurich.ibm.com

II. Motivation

To show intuitively the motivation of this work we executed traces of the well-known NAS [11] parallel benchmarks (class A and 64 processes) in a 4-ary 3-tree (64 nodes) and plotted the percentage of packets that used each level in Fig. 1. We can see how in all the benchmarks the first level is used by all the packets, but subsequent levels have lower utilization. In most cases the second level is used by less than 75% of the packets, and the third level is used by less than 33% of the packets. The only exceptions are the **FT** and **IS** benchmarks, whose communication patterns force every node to exchange data with all the other nodes; in other words, traffic does not exhibit locality. At any rate, even in these two benchmarks the utilization of level 3 is around 25% lower than the utilization of the first level.

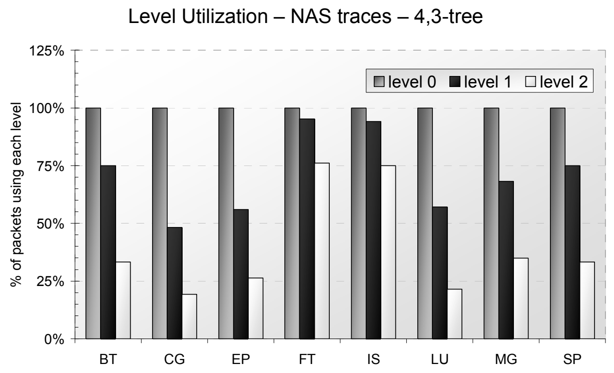

Figure 1. Fraction of packets using each level of the tree.

This motivates the reduction of the bandwidth in the upper levels of the topology as their utilization is noticeably lower than that of the first level. As the reduction of the bandwidth in each level would depend on the application (or application mix) to be executed, for the sake of simplicity in this paper we will focus only on constant reduction ratios between consecutive stages.

III. Topologies Under Study

In this section we will describe two different multi-stage, tree-based topologies. In these descriptions we assume that all switches used to build a given network have the same radix. For the purpose of this paper we leave unplugged the upward ports of the topmost level of switches. This assumption has advantages in terms of simplicity in the descriptions, and also provides scalability. The disadvantage is in terms of cost, because some resources are unused; this is particularly relevant for those topologies with more switches in the top level. In practical implementations, all ports of the highest switch level may be used as downward ports, eventually providing connectivity to a larger number of compute nodes. Alternatively, we may consider a single switch as an aggregation of lower radix virtual switches, which results in a smaller number of switches in the topmost stage of the system.

A. Definitions

In the graphical representations of the topologies (see Fig. 2), boxes represent switches and lines represent links between them. Note that we neither show the compute nodes and links connected to the first level switches, nor the last level of upward links (which, as we stated before, are unplugged). These elements are hidden for the sake of simplicity.

Throughout this paper we will use n to denote the number of levels in a network, and N to denote the number of compute nodes attached to it. We will denote the total number of switches in a topology as S, and the number of switches at level i as S_i. The total number of links will be denoted as L. The switch radix will be denoted as R. We call the relation between the number of downward ports of a switch and the number of upward ports the *slimming factor*. For example, taking a look at the switches in the topology shown in Fig. 2b, four ports are downward ports, linked to switches in the next lower level. The remaining two ports of each switch are upward ports that connect to switches in the next higher level; therefore the slimming factor is 2:1.

In the topological descriptions that follow, we denote each switch port within the system as the level where the switch is, the position of the switch in that level, and the number of the port in that particular switch. We call the lower level of switches (those attached to compute nodes) level 0; obviously, level n-1 is the one on the top of the tree. We number the switches in each level from left to right, starting from 0. Ports in a switch are denoted as upward (↑) or downward (↓), and numbered from left (0) to right. Thus, a port can be addressed as a 4-tuple <*level, switch, port, direction*>.

Given two ports P and P', they are linked ($P \leftrightarrow P'$) when there is a connection (link) between them. As links are full-duplex, in the expressions concerning linkage we avoid the redundancy of showing downward connections. We will call *level l*, *switch s* and *port p* the address components of a given downward port, and *level l'*, *switch s'* and *port p'* the address components of the port to which it is connected, *i.e.* a downward port in a switch located at the level above.

Along this paper, we will refer to *heavy* and *light* workloads. Light workloads are those in which the number of messages circulating simultaneously through the network is low. In contrast, heavy workloads are those in which most of the nodes are injecting messages

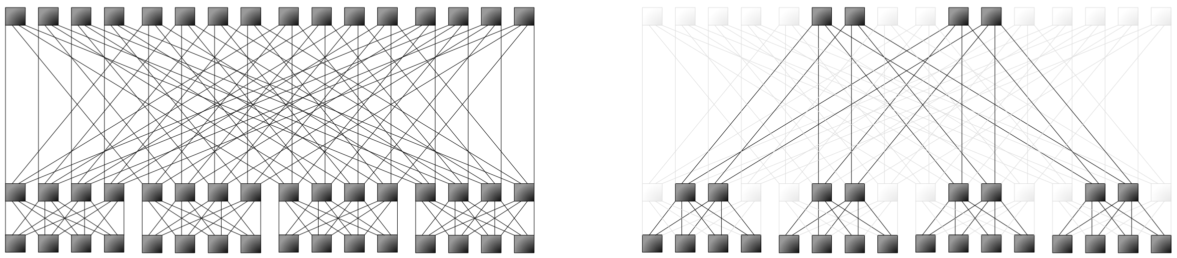

Figure 2. Samples of the topologies under study to build 64 nodes networks. A 4,3-tree (left) and a 4:2,3-thintree (right).

at once so that the network will experience peaks of congestion; this situation would be even worse if messages are addressed to distant destinations.

B. k-ary n-tree

This is the best-known of the topologies considered in this study. It will be the yardstick to compare the thin tree against. In *k*-ary *n*-trees [16] *k* is half the radix of the switches—*i.e.*, the number of links going upward (or downward) from the switch—and *n* is the number of levels. It will be denoted through this paper as *k*,*n*-tree. Notice that its slimming factor is always 1:1.

A *k*,*n*-tree is typically built in a butterfly fashion between each two contiguous levels, Fig. 2a shows a depiction of a 4,3-tree. The topological neighborhood description is as follows:

$$\forall l \in [0, n-1),\ \ \forall s \in [0, k^{n-l}),\ \ \forall p \in [0, k)$$
$$l' = l + 1$$
$$s' = \left((p \cdot k^{l}) + (s \bmod k^{l}) + \left(k^{l-1} \cdot \left\lfloor \frac{s}{k^{l}+1} \right\rfloor \right) \right) \bmod k^{n-1}$$
$$p' = \left(\left\lfloor \frac{s}{k^{l}} \right\rfloor - \left(k^{l+1} \cdot \left\lfloor \frac{s}{k^{l}} \right\rfloor \right) \right) \bmod k$$
$$\langle l, s, p, \uparrow \rangle \leftrightarrow \langle l', s', p', \downarrow \rangle$$

The main advantages of this topology are the high bisection bandwidth and the large number of routing alternatives for each pair of source and destination—a path diversity that can be exploited via adaptive routing. Nevertheless, they might be expensive and complex to deploy, because of the large number of switches and links.

C. k:k'-ary n-thintree

We define a thin tree as a cut-down version of a *k*-ary *n*-tree in which we apply a given slimming factor. We will denote it as *k*:*k'*,*n*-thintree, being *k* the number of downward ports, *k'* the number of upward ports and *n* the number of levels. The slimming factor is, the ratio between *k* and *k'*. Note that *k* does not need to be a multiple of *k'* so that we can produce a thin tree with arbitrary values of *k* and *k'*. A *k*,*n*-tree is actually a *k*:*k*,*n*-thintree.

A 4:2,3-thintree is depicted in Fig. 2b. Removed switches and links from a full-fledged *k*-ary *n*-tree are shaded. The topological neighborhood relationship between ports in a thin tree is described as follows:

$$\forall l \in [0, n-1),\ \ \forall s \in \left[0, k \cdot \left\lfloor \frac{k}{k'} \right\rfloor^{n-l} \right),\ \ \forall p \in [0, k')$$
$$l' = l + 1$$
$$s' = \left((p \cdot k'^{l}) + (s \bmod k'^{l}) + \left\lfloor \frac{s}{k \cdot \left\lfloor \frac{k}{k'} \right\rfloor^{l}} \right\rfloor \cdot k' \cdot \left\lfloor \frac{k}{k'} \right\rfloor^{l} \right)$$
$$p' = \left\lfloor \frac{s}{\left\lfloor \frac{k}{k'} \right\rfloor^{l}} \right\rfloor \bmod k$$
$$\langle l, s, p, \uparrow \rangle \leftrightarrow \langle l', s', p', \downarrow \rangle$$

In this topology the bisection bandwidth has been reduced, as well as the number of switches and links (*i.e.*, cost and complexity). We want to investigate how applications suffer these reductions. Thin trees are easier to deploy than regular trees and, if *k* and *n* values are kept, the radix of the switches is smaller.

Table I closes this section showing the expressions to compute the total number of nodes (*N*), the total number of switches (*S*), the number of nodes per level (S_i), the number of links (*L*) and the radix of the switches (*R*) from the parameters of the networks (*k*, *k'* and *n*), both for *k*,*n*-trees and *k*:*k'*,*n*-thintrees.

TABLE I.
CHARACTERISTICS OF THE NETWORKS

	***k*,*n*-tree**	***k*:*k'*,*n*-thintree**
Nodes	$N = k^n$	$N = k^n$
Switch radix	$R = 2 \cdot k$	$R = k + k'$
Switches	$S = n \cdot k^{n-1}$	$S = \sum_{i=0}^{n-1} k'^{(n-i)-1} \cdot k^i$
Switches per level $\forall i \in [0, n-1]$	$S_i = k^{n-1}$	$S_i = k^{(n-i)-1} \cdot k'^i$
Links	$L = S \cdot k$	$L = S \cdot k$

IV. EXPERIMENTAL SET-UP

We used INSEE [17] to evaluate some different tree-like networks, feeding them with a collection of workloads. The simulator measures time in terms of cycles, the time required by a *phit* (physical transfer unit) to traverse one switch.

A. Switches

For this work, we have chosen simple input-buffered switches whose radices range from 10 to 16, depending on the topology. In order to keep things simple, we do not use virtual channels. The arbitration of each output port is performed in a random way, that is, every time an output port is free it randomly chooses among all the input ports that have requested this resource. Transit queues are located in the input ports and are able to store up to four packets. A schematic model of the switch is depicted in Fig. 3. The input ports (at the left) have the packets queues and are connected to the output ports (at the right) by means of a crossbar.

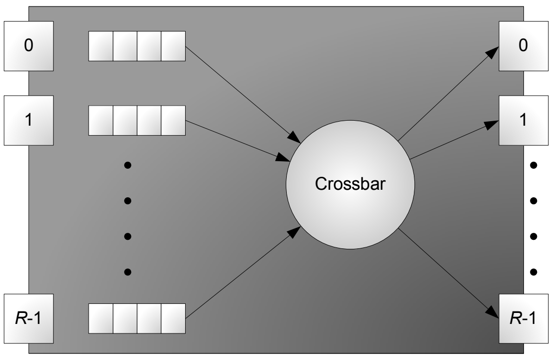

Figure 3. Model of a radix-*R* switch.

In this work we model the node as a traffic generation source with one injection queue, which is able to store four packets. It is also the sink of the arrived messages. When generating traffic, we consider reactive sources, meaning that the reception of a message may trigger the release of a new one. This way, we can model the causality inherent to actual application traffic. Messages are split into packets of a fixed size of 16 phits. One phit is the smallest transmission unit, fixed to 32 bits. If a message does not fit exactly in an integral number of packets, the last packet contains unused phits.

The switching strategy is virtual cut-through. Routing is, when possible, adaptive using shortest paths. A credit-based flow-control mechanism is used, so that when several output ports are available, the one with more available credits is selected. Credits are communicated out-of-band, so they do not interfere with application traffic.

B. *Networks under Study*

In this work we fixed the number of downward ports per switch (set to 8) and slimming factors (set to 2:1 and 4:1). We also fixed the target number of connectable compute nodes to 4096. With these restrictions, we worked with the following topologies: 8,4-tree, 8:4,4-thintree, 8:2,4-thintree. Table II summarizes some characteristics of the studied networks. Reader should note that all the switches have 8 downward ports, but the actual radix of the switches is not always the same, being smaller in the more slimmed topologies. Thus, in this evaluation the complete tree has advantage compared with the thinner alternatives: it uses more links, and more switches that are also larger. Consequently, performance measurements are biased towards the 8,4-tree.

TABLE II.
CHARACTERISTICS OF THE TOPOLOGIES UNDER STUDY.

	8,4-tree	8:4,4-thintree	8:2,4-thintree
Radix	16	12	10
Switches	2048	960	680
Links	16384	7680	5440
Nodes	4096	4096	4096

C. *Workloads*

As we stated in the introduction, we would like to test the selected networks with realistic traffic, ideally taken from traces obtained from applications running in actual supercomputers. Since it is difficult to obtain and handle traces of applications running on thousands of nodes, we decided to use instead some synthetic traffic generators which emulate data interchanges typically used in scientific parallel applications. Note that we assume a mapping of one process to one compute node attached to one leave of the network.

The selected workloads are all-to-all (**AA**), binary tree (**BI**), butterfly (**BU**), nearest neighbor communication in 2D (**M2**) and 3D (**M3**) meshes, and wave-front communication in 2D (**W2**) and 3D (**W3**) meshes. Their spatial and causal patterns are defined and justified in [12] and [13]. They were generated for a fixed network size of *N*=4096 communicating nodes and message lengths of 10KB. The only exception is **AA** whose messages are noticeably shorter (512B length) to allow reasonably short simulation times—note that the number of messages in this communication pattern scales quadratically with the number of nodes.

Furthermore four synchronized random workloads [12] have been generated. These workloads are composed of 65536 messages of 1KB each and waves of 1 (**R1**), 1024 (**R2**), 4094 (**R3**) and 16386 (**R4**) messages, which emulate four different levels of application coupling.

BI, **W2** and **W3** can be considered light workloads as they have highly causal communication patterns which only allow a few messages traversing the network at once. In contrast, the remaining workloads can be considered heavy as most of the nodes try to use the network simultaneously, which can drive the network in a highly congested state. This is especially true for **AA** and the random workloads (**R1**, **R2**, **R3** and **R4**) as they do not exhibit locality in communications. Note that random workloads with a large amount of messages per wave are heavier than those with small wave length, as the processing nodes need to synchronize less often and therefore they keep injecting traffic in the network for long periods of time, which exacerbates congestion.

V. EXPERIMENTS AND ANALYSIS OF RESULTS

In this section we evaluate the previously described networks using the selected mix of workloads. We gathered the time (in simulation cycles) used by the networks to deliver all the messages of each workload. As these times differ widely, due to their characteristics, we normalized them, using the times taken by the complete trees as the reference. These normalized times are plotted in Fig. 4.

The reader can observe that the two thin tree networks perform acceptably well in light traffic patterns. With them, the *k*-ary *n*-tree cannot take advantage of its high bandwidth and path diversity, just because the network occupancy is low and so is the probability of two packets competing for an output port of a switch. In contrast, under heavy loads, the high bandwidth of the *k*-

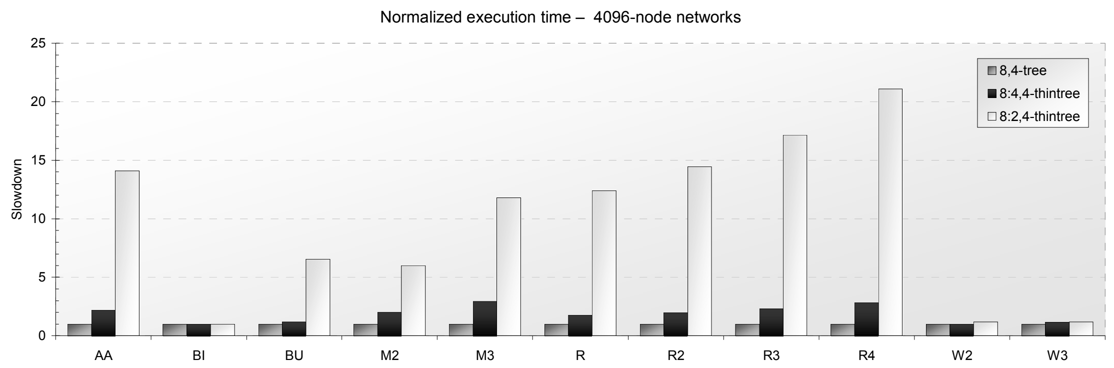

Figure 4. Time required by each network to deliver the complete workloads (normalized to the 8,4-tree).

ary n-tree topology is able to handle the high amount of packets inside the network. In the thin networks there is too much contention due to the bandwidth reduction between each level so the packet delivery is slower. However the 8:4,4-thintree obtained *not-so-bad* performance figures that required always below three times the time required by the 8,4-tree. Note that the measured times include only communication time and, therefore the actual execution time of an application will depend on its communication-computation ratio. Note also that the full-fledged tree used higher radix switches and, thus has advantage over the thin-trees. If we devote the unused ports of the thin-trees to be used as downward ports we will increase locality and therefore better communication times would be obtained. This evaluation will not be carried out because of the limits in paper length, but we encourage the interested reader to look at [14] for a detailed evaluation of this issue.

We can conclude, from the point of view of raw performance, that the k-ary n-tree is the best-performing topology in all experiments, therefore if we had unlimited (financial) resources we could just select it as the *best-performing* option. Nonetheless that option may not be the most *cost-effective*.

Here we propose a means to measure the effectiveness of a network taking into account the workloads using it. Actual workloads vary widely from site to site, depending on the applications in use. In this work we are not using actual applications, but a collection of synthetic—but representative—workloads. We describe a network-efficiency function in the context of these workloads that can be extended with further workload types.

For each given workload simulation reports a (relative) time T_W. For example, we have a certain execution time T_{BU} for butterfly. Note that as these values are relative: they are always 1 for the k-ary n-tree. Depending on the application mix of interest in a particular computing center, we may apply a weighting factor to each experiment w_W. This weight should be large for those applications that are used often. For a given network, we define its performance φ as follows:

$$\varphi = \frac{1}{\sum_W w_W \cdot T_W}$$

Note that for a given application mix and set of weights a higher value of φ represents a better-performing network. As we cannot identify all representative application mixes, in this paper we decided to use 1 for all the weights. With this, the denominator in our efficiency value is just the addition of the (relative) times obtained in the experiments. This yields a value of φ=1/11 for the k-ary n-tree. We further normalize this value (multiplying it by 11). Table III shows the normalized performance values for all the experiments.

Performing an exhaustive cost analysis of a complete system is, clearly, a difficult task, that requires the knowledge of a large number of parameters, including the choice of technologies and physical placement of the elements of the system (nodes, racks). This is out of the scope of this paper. For this reason, we will consider three simple cost functions to compute the cost of each network in order to be able to carry out a performance/cost comparison. In the first function (c_C) the cost of the switch is constant regardless of the radix of the switch, in the second one (c_L) the cost of the switch lineally depends on the radix and in the third one (c_Q) the cost increases quadratically with the radix—which is the more accurate as it does so for the heart of the switch (the crossbar [6]). To simplify these functions, the number of links is not taken into account as it depends on the number of switches. Hence, the three functions are as follows:

$$c_C = S, \qquad c_L = S \cdot R, \qquad c_Q = S \cdot R^2$$

The value for the three cost functions, as well as the cost-efficiency—computed as the performance divided by the cost—of the three systems under study are show in Table III.

TABLE III.
PERFORMANCE/COST EFFICIENCY OF THE TOPOLOGIES UNDER STUDY.

	8,4-tree	8:4,4-thintree	8:2,4-thintree
φ	1.00	0.60	0.11
c_C	1.00	0.47	0.33
φ/c_C	1.00	**1.27**	0.35
c_L	1.00	0.35	0.21
φ/c_L	1.00	**1.70**	0.55
c_Q	1.00	0.26	0.13
φ/c_Q	1.00	**2.26**	0.88

We can see how, for the three cost functions, the performance/cost efficiency of the 8:4,4-thintree overtakes those from the complete tree. The efficiency of the 8:2,4-thintree, however, is lower than that of the 8,4-tree, even for the quadratic cost function. In this case savings are significant, but result in a very poorly performing system. The conclusion here is that a thin tree can achieve better performance/cost efficiency than a full fledged tree, but not for every slimming factor. In other words, the selection of a good sliming factor requires a thorough fine-tuning process.

VI. RELATED WORK

Indirect interconnection networks have evolved noticeably from the first multi-stage networks as those proposed by Clos in [2]. Those networks were built with low-radix switches (typically 2, 4 or 8) and aimed to interconnect at most a few hundred nodes. Current spines, as that on the Mare Nostrum supercomputer [1], have switches with hundreds of ports and are able to interconnect several thousands of nodes. Former trees were low-radix: for example, the CM-5 [9] had a radix-8 data network. Current ones use switches with higher radices, as those radix-24 of the Cray XD1 [3]. There are also recent tree-like proposals as the Black Widow Clos network [4] that takes advantage of the high availability of ports (radix-64 switches) to add side-links to the common tree-like arrangement. However, the most noticeable change in these networks is that former indirect networks were built *ad-hoc* for the target systems, whereas current high-performance networking technologies as QsNet [15], Myrinet [10] or InfiniBand [7] have favored building super-clusters with *off-the-shelf* components.

Network bandwidth and latency have also experienced noticeable improvements during the last decade, from the 800Mbps of the ASCI Red (1997) [18] to the 100-120 Gbps in the 100G Ethernet or the InfiniBand 12X-QDR. This takes us to a network bandwidth improvement of two orders of magnitude in the last decade. The latency of the full protocol and the network in the ASCI Red (taking into account the message passing library) is 12μs. Both Myri-10G and InfiniBand latencies are around 2μs. Thus, latency has been improved (one order of magnitude), but not as noticeably as bandwidth has.

Taking a look at the Top500 list [5], we can see two clear trends. On the one hand, the choice of topology for custom-made, massively parallel computers is the 3D cube. On the other, commodity-based systems (super-clusters) are built around the class of trees discussed in this paper. Most of the machines in the middle positions are arranged this way, which justifies our interest in tree-like topologies. At least three super-clusters that are in positions #1 (RoadRunner), #29 (Tsubame) and #47 (Thunderbird) of the November 2009 edition of this list were built with Infiniband networks arranged as thin trees—actually, spines. In Thunderbird the slimming factor is 2:1; RoadRunner and Tsubame goes further, to 4:1 and 5:1 respectively.

We have not found any evaluation work providing the rationale behind those decisions. However, in this work we have shown that, compared to full-fledged *k*-ary *n*-trees, thinner topologies provide comparable performance at much lower cost. The savings in the networking elements could be invested in other ways to improve the system (faster CPUs, better performing networking technology, enlarge the system, etc).

VII. Conclusions

In this paper we have described and characterized a slimmed version of *k*-ary *n*-trees: *k*:*k'*-ary *n*-thintrees. A thin tree can be seen as a *k*-ary *n*-tree after removing some links and switches. A thin tree costs a fraction of the price of a complete tree, in terms of deployment and maintenance. In terms of performance, thin trees with low slimming factor perform as well as comparable *k*-ary *n*-trees. Excessive slimming (beyond 2:1 in our workbench) results in awful performance results. In our experiments a comparison of performance/cost efficiency turns out to be very favorable for the thin trees with a 2:1 slimming factor. More slimmed topologies (4:1 in our set-up) showed to be unable to beat regular *k*-ary *n*-tree in terms of cost-efficiency.

Thin trees are obviously cheaper than the complete tree, but their bandwidth in the upper levels is greatly reduced. After removing links and switches in the upper levels, performance could be maintained (or even increased) due to an effective exploitation of locality [14]. Using fixed-radix switches, a thin tree devotes more ports as downward links, so more nodes can communicate without using the upper levels.

The reader should note that the network is only a part of the system, so that the execution time depends (greatly) on the behavior of the other components, and the interactions among all of them. In other words, the advantages or disadvantages of a given network might not be as clear as shown in our evaluations. This is an argument against the better-performing, more-expensive networks, because in real set-ups the benefit of using them will be diluted. The extent of this dilution is application-dependent. Furthermore, the collection of workloads used in this work might not be representative—and probably is not—of all actual workloads used at supercomputing centers. A thorough study should be customized for a particular site, taking in mind the applications that are executed frequently.

Acknowledgments

This work has been supported by the Spanish Ministry of Education and Science, grant TIN2007-68023-C02-02, and by Basque Government grant IT-242-07. Mr. Javier Navaridas is supported by a doctoral grant of the UPV/EHU.

References

[1] Barcelona Supercomputing Center “Mare Nostrum”. Available at: http://www.bsc.es/

[2] C Clos: “A Study of Non-Blocking Switching Networks” Bell System Technical Journal, March 1953, pp.406-424.)

[3] Cray Inc., “Cray XD1 Overview”. Available at: http://www.cray .com/ products/xd1/

[4] WJ Dally et al., “The BlackWidow High-Radix Clos Network”, Proceedings of the 33rd annual international symposium on Computer Architecture, p.16-28, 2006, June 17-21.

[5] JJ Dongarra, HW Meuer, E Strohmaier. “Top500 Supercomputer sites”. Nov. 2007 edition. Available at: http://www.top500.org/

[6] H El-Rewini, and M Abd-El-Barr “Advanced computer architecture and parallel processing” Wiley, 2005 (ISBN: 978-0-471-46740-3)

[7] Infiniband Trade Association. “Infiniband® Trade Asociation”. Available at: http://www.infinibandta.org

[8] CE Leiserson. “Fat-trees: Universal networks for hardware efficient supercomputing”. IEEE transactions on Computers, C-34(10):892–901, October 1985.

[9] CE Leiserson et al., “The Network Architecture of the Connection Machine CM-5”, Symposium on Parallel Algorithms and Architectures (April 1992).

[10] Myricom. “Myrinet home page”. Available at: http://www.myri.com/

[11] NASA Advanced Supercomputing (NAS) division. “NAS Parallel Benchmarks” Available at: http://www.nas.nasa.gov/ Resources/Software/npb.html

[12] J Navaridas and J Miguel-Alonso. “Realistic Evaluation of Interconnection Networks Using Synthetic Traffic”. 8th International Symposium on Parallel and Distributed Computing. June 30-July 4, 2009. Lisbon, Portugal.

[13] J Navaridas, J Miguel-Alonso, FJ Ridruejo. "On synthesizing workloads emulating MPI applications". The 9th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC-08). April 14-18, 2008, Miami, Florida, USA.

[14] J Navaridas, J Miguel-Alonso, FJ Ridruejo, W Denzel. "Reducing Complexity in Tree-like Computer Interconnection Networks". Technical report EHU-KAT-IK-06-07. Department of Computer Architecture and Technology, UPV/EHU. Submitted to Journal on Parallel Computing.

[15] F Petrini, W Feng, A Hoisie, S Coll and E Frachtenberg. “The Quadrics Network: High-Performance Clustering Technology”. IEEE Micro 22, 1 (Jan. 2002), 46-57.

[16] F Petrini and M Vanneschi. “k-ary n-trees: High Performance Networks for Massively Parallel Architectures”. In Proceedings of the 11th International Parallel Processing Symposium, IPPS’97, p. 87–93, Geneva, Switzerland, 1997.

[17] F.J. Ridruejo, J. Miguel-Alonso. "INSEE: an Interconnection Network Simulation and Evaluation Environment". Lecture Notes in Computer Science, Volume 3648 / 2005 (Proc. Euro-Par 2005), Pages 1014 - 1023.

[18] Sandia National Labs, “ASCI Red”. Available at: http://www.sandia .gov/ASCI/Red/