HyperX: Topology, Routing, and packaging of Efficient Large-Scale Networks

Itay Snir
Advisor: Prof. Ori Rottenstreich

February 2021

1 Introduction

In order to achieve exascale performance, systems will grow to over 100,000 sockets, as growing cores-per-socket and improved single-core performance provide only part of the speedup needed. To meet the need, we consider HyperX, an extention to the Hypercube and flattened butterfly topologies, and give an adapive routing algorithm, DAL. Our main contributions include a formal descriptive framework, enabling a search method that finds optimal HyperX configurations; DAL; and a low cost packaing strategy for an exascale HyperX. Our analysis of efficiency, performance, and packaging demonstrates that the HyperX is a strong competitor for exascale networks.

We define radix to be the number of bidirectional ports that the switch supports, where a port consists of separate unidirectional input and output channels. We introduce a class of n-dimensional networks that we call HyperX. In a HyperX, each switch is connected to all of its peers in each dimension, and the number of switches in each dimension can be different. In addition, we present an algebric framework that describes the HyperX topology, and develop a search procedure that finds a least-cost HyperX configuration. Third, we introduce a new adaptive routing algorithm, DAL, and show via simulations that DAL takes better advantage than existing flattened butterfly routing of the path diversity. The novel contribution of this paper is the formalized framework, search procedure, and DAL.

2 HyperX

2.1 Topology

A HyperX is a direct network of switches in which each switch is connected to some fixed number T of terminals. A terminal can be a compute node, cluster of compute nodes, I/O node, or any other interconnected device. The switches are viewed as points in an L-dimensional integer lattice. Each switch is identified by a coordinate vector, or multi-index $I = (I_1, ..., I_L)$ where
$0 \leq I_k \leq S_k \quad for \quad each\, k = 1, ..., L.$
In each dimension, the switches are fully connected. Thus, there are bidirectional links from each switch to exactly $\sum_{k=1}^{L}(S_k - 1)$ other switches: a switch connects to all others whose multi-index is the same in all but one coordinate. The number P of switches in the HyperX satisfies $P = \prod_{k=1}^{L} S_k$.

In a simple HyperX all links have uniform bandwidth. The topology can be generalized by allowing the link bandwidths to be multiples of some unit bandwidth. We let $K = (K_1, ..., K_L)$ represent the relative link bandwidths in each of the dimensions. A *regular* HyperX is one for which $S_k = S$ and $K_k = K$ for all $k = 1..L$. Thus, a regular HyperX is determined by the parameters $L, S, K\, and\, T$.
The bisection bandwidth of a HyperX is realized by cutting one of its dimensions in half. The channel bisection of such a cut is

$$C_m = (1/4)K_m S_m \prod_{k=1}^{L} S_k = (P/4)K_m S_m$$

if dimension m is bisected. If $K_m S_m$ is smallest among all the dimensions of the HyperX, so that $K_m S_m < K_k S_k$ for all $k = 1, ..., L$, then the above equation determines the bisection bandwidth.

2.2 Routing algorithms

The set of shortest paths between two switches in a HyperX is determiend by their coordinate vectors. Consider paths from switch zero (all its coordinates are 0) to switch $I = (I_1, ..., I_L)$. The length of each shortest path is equal to the number of nonzero elements of I. Let Δ be the set dimensions k for which $I_k \neq 0$. We call these the $offset$ dimensions; the others are *aligned* dimensions. Every shortest path goes directly (by one step) from the source 0 to the destination in each of the offset dimensions in some sequence. If the number of offset dimensions is D, then there are $D!$ shortest paths.
A minimal deterministic routing method is obvious: route to the destination by the shortest path determined by moving in the dimensions of Δ in a fixed order; This is known as dimension-order routing. When one dimension is blocked due to buffer congestion at the downstream switch, we choose another offset dimension. We call this Min-AD. Min-AD adaptively explores the space of $D!$ shortest paths. Another routing algorithm, oblivious routing algorithm,

routes all packets through an intermediate node chosen at random with uniform probability.

Adaptive non-minimal routing is in general better than either deterministic or oblivious routing for both the flattened butterfly topology, as well as the HyperX. The adaptive Clos-AD algorithm chooses at a packet's source node between dimension ordered minimal routing and non-minimal routing, based on esimates of queuing delay. If it chooses to route non-minimally, Clos-AD routes to a randomly chosen lowest common ancestor switch, considering the network as a transformed folded Close network.

Clos-AD can be improved in two respects. First by deciding to route minimally or not only at the source, it loses significant ability to adapt to congestion encountered $en - route$. Second, by making a choice of intermidiate node that is informed by a mapping of the folded Close into the HyperX, it introduces dimensional asymmetry. To address the issues raised, we propose the DAL routing algorithm:

1. Mark all offset dimensions as deroutable on creation of the packet at its source. On arrival at a switch:

2. Find an offset dimension with an unblocked path to an aligned switch in that dimension. If none exist, then:

3. Find an unmarked offset dimension and unblocked switch that is offset from the destination, and if one exists, then route and mark the dimension as no-longer-routable. else

4. Push the packet into a minimal, dimension-order, deterministic routed virtual channel.

The virtual channel used as a last resort is used to prevent deadlock in DAL. DAL deroutes in one of the HyperX dimensions at a time, treating each HyperX dimension independently and symmetrically. DAL can deroute as many times as there are offset dimensions. Unlike Clos-AD, DAL routes a packet only through a sub-network of those nodes that are aligned with both the source and the destination. Moreover, DAL may deroute a packet at each hop on its route, thereby adapting to congestion not visible only at the source node.

3 Related work

3.1 Establishing short disjoint paths

3.1.1 Model and definitions

We refer to a HyperX topology (as an instance of a general network topology) as an undirected graph $G = (V, E)$. For the HyperX the number of nodes is given by $P = \prod_{k=1}^{L} S_k$. The transmission of data from a source node $s \in V$ to a destination node $t \in V$ is done through a path $\pi = (v_0, ..., v_l)$ that a link exists between any two adjacent nodes. For a set of paths Π between a source node and a destination node, a common node of Π is a node, other than the source and destination, that belongs to more than a single path. The overlap between paths can be restricted in various ways to eliminate mutual interference among them. We say that a set of paths Π is link-disjoint if any pair of path $\pi_i, \pi_j \in \Pi$ has no common links. We characterize the paths between a source and a destination node while referring to their lengths and counts. We also examine potential disjointness among paths. For two nodes $s = (s_1, ..., s_L)$, $t = (t_1, ..., t_L)$ we denote by $\Delta(s, t) = \sum_{k=1}^{L} I(s_k \neq t_k)$ the number of dimensions in which they do not share the same value.

3.1.2 Minimal length paths

$\Delta(s, t)$ is the length (number of hops) of the shortest path between these nodes. The number of such paths is given by $\Delta(s, t)!$. To build such a path, we fix those coordinates common for s, t. The other $\Delta(s, t)$ coordinates are changed gradually one by one from those values of s to those of t. These coordinates can be changed in any order, each implying a different path such that the number of potential shortest paths is $\Delta(s, t)!$.

let Π_s denote the maximal size node-disjoint set of shortest paths. In a HyperX graph G, for every pair of nodes s, t there exists a set of node-disjoint minimal length path of cardinality $\Delta(s, t)$ and there is no larger set with such a property, namely $|\Pi_s| = \Delta(s, t)$. Moreover, in a HyperX graph G, for every pair of nodes s, t the maximal set of link-disjoint shortest paths equals the size of a maximal set of node-disjoint paths.

3.2 Fault tolerance approach

3.2.1 Available shortest paths upon a single failure

Let s, t be two nodes for which we done $\Delta = \Delta(s, t)$ such that I_Δ is the set of coordinates of size $\Delta(s, t)$ by which nodes s, t differ. Consider the failure of some link e. The location of e with regards to s and t implies the impact on path availability. In HyperX a link crosses a single dimension. Let d_e denote that dimension for the link e.
If $d_e \notin I_\Delta$, namely the unavailable link e crosses a dimension which is not among those by which s and t differ the impact on path availability is limited. In

particular, since shortest paths are of length Δ and make only use of dimensions in I_Δ, the link e does not appear in any shortest path and its failure has no impact on availability of shortest path. In particular, by the analysis from Table 2 the number of available paths is $\Delta!$ and the maximal size of subset of disjoint paths is Δ. We now consider the case of $d_e \in I_\Delta$, namely the dimension of link e is among those by which s, t differ. Denote $e = (a, b)$ such that $\Delta(s, a) + 1 = \Delta(s, b)$. If $\Delta(s, a) + \Delta(b, t) + 1 > \Delta$, then again no shortest paths include e and its failure again has no impact. Otherwise, $\Delta(s, a) + \Delta(b, t) + 1 = \Delta$ and there are shortest paths between s, t that include e. A shortest path $\pi(s, t)$ that includes e can be described as a path from $\pi(s, a)$, e and $\pi(b, t)$. All such paths are eliminated with a failure in e. Let $\Delta_1 = \Delta(s, a), \Delta_2 = \Delta(b, t)$ such that $\Delta = \Delta_1 + \Delta_2 + 1$. There are of course $\Delta_1!$ shortest paths between s, a and $\Delta_2!$ shortest paths between b, t. Among the $\delta!$ shortest paths between s, t, the number of eliminated paths with the unavailability of e is $\Delta_1! * \Delta_2!$. If the link e appears as a link adjacent to s or t in a shortest path between s, t, then the maximal size of a set of disjoint shortest paths reduces to $\Delta - 1$. Else, the maximal size of set of disjoint shortest path is not affected and equals to Δ.

3.2.2 Available disjoint paths upon a single failure

Upon the unavailability of a single link, the number of disjoint paths between nodes s, t is at least $d - 1 = \sum_{k=1}^{L}(S_k - 1) - 1$. Moreover, for two nodes s, t, the maximal size of a set of paths which are of length at most $\Delta(s, t) + 1$ (shortest or almost shortest) and are link-disjoint is $\Delta(s, t) + \sum_{i \in I_\Delta}(S_i - 2)$. If e connects s or t to a node along a dimension $i \in I_\Delta$, then a pth among the $\sum_{i \in I_\Delta}(S_i - 2)$ disjoint paths of length $\Delta + 1$ becomes unavailable, and there is no option to construct an alternative set of disjoint paths up to length at most $\Delta + 1$. The path might also become unavailable when the link e is located among the $\Delta - 1$ links connecting the two nodes sharing the intermediate value in the dimension from I_Δ.

4 Results

We begin by showing simulation results that explore the performance of various routing algorithms on HyperX. They show that DAL is more effective than adaptive minimal, Valiant, and Clos-AD routing. Further simulations show that HyperX with DAL is generally better than an adaptively routed folded Clos with similar system configurations.

We present three standard traffic patterns: bit complement, bit rotate, and transpose, as well as one additional pattern, Swap2, which is a permutation designed to highlight the difference between Clos-AD and DAL. In this pattern, the even-numbered nodes exchange messages with a peer across the bisection in one dimension and the odd-numbered nodes exchange messages with a a peer across the bisection in another dimension.

According to the simulations, we can see that the adaptive algorithms, Clos-AD and DAL are better choices then the oblivious routing algorithms. For low loads, both algorithms choose a minimal route and achieve low latency, whereas at high load, they adapt, choosing nonminimal routes to avoid congestion. The primary difference between the two algorithms is when they decide to adapt. Clos-AD estimates at the source the delay of a minimal route, and a non-minimal route and makes its adaptation decisions eagerly. DAL makes decisions lazily, as the packet traverses the network and decides on the route independently per dimension. Eager adaptation means that Clos-AD has less information which could be out of date, while lazy adaptation allows DAL to be far more nimble. Moreover, DAL has higher saturation throughput than the alternatives on all patterns tested.

In HyperX, because fewer terminal nodes can be connected to each switch as dimensions are added, the cost of each additional dimension is higher than a folded Clos, where additional dimensions result in a constant cost increase. The advent of high-radix switches changes this story because very large networks can be built with few dimensions. Because HyperX can handle a higher load than a folded Clos of equal bandwidth, this makes a compelling case for HyperX.

5 Extended Work

5.1 Introduction

As the number of nodes in a hypercube system increases, the probability of node failure also increases. Data-centers and large-scale distributed-memory systems are likely to suffer from faulty devices and links, which may drastically decreases the network's efficiency. Therefore, an important role within large-scale system engineering includes a design that takes into account an excellent fault-tolerance mechanism, which is crucial in modern times.
In our extended research we would like to contribute by designing a new, dynamic and adaptive routing algorithm for the HyperX topology. While DAL routing uses dimensional routing, based on local-information in order to achieve load balancing, it ignores some of the unique properties involved with the HyperX.
In particular, we would like to purpose a global-information-based routing algorithm. The algorithm will utilize properties regarding link and node disjoint paths, in order to handle fault tolerance.
In general, global-information based model may obtain optimal or sub-optimal result. However, it requires complex procedure in order to collect global information. While local-information models use a weaker and more reasonable assumption, it can only be used to achieve local optimization. Therefore the length of a routing path is unpredictable in general, and global optimization is impossible. Our new approach is hybrid, and consists the best out of the two worlds - performance close to the optimal global-optimization limit, as well as simplicity, similar to local-optimization scheme.

5.2 Problem Definition

Unicasting is a one-to-one communication between two nodes; one is called the source node, and the other the destination node. Our goal is to minimize the average *latency* for unicasting communication. Therefore an algorithm which minimizes the average hop count is required. While DAL algorithm only used local-switch information, meaning every switch could only receive information regarding its neighbors, a global optimization could not be achieved. Our approach softens the local information requirement, at the cost of complexity, to achieve semi-global optimization.

5.3 Safety Vectors

$safetyvector$ includes faulty link information, and provide accurate information about the number and distribution of faults in an HyperX. Each node (presenting a switch) in the HyperX is associated with a bit vector, called a safety vector, calculated through $L-1$ rounds of information exchange among neighboring nodes, where L represents the number of Dimensions for the HyperX. For the special case of an optimal unicast between two nodes is guaranteed if the kth bit of the safety vector of the source node is one (set bit), where k is the Δ between the source and the destination nodes. Recall we denote Δ as the number of offset dimensions between a source and destination node within the HyperX. if the Δth bit of the safety vector is set, an optimal unicast between two nodes is guaranteed.

We denote $u^{(i)}$ as the neighbors of u along dimension i.

The safety vector of a faulty node is (0,0,..,0). For a non-faulty node u, if node u is an end node of a faulty link, then node u's safety vector is (0,0,...,0) only from the view of the other end node adjacent to the same faulty node. The safety vector of each node, $(u_1^{(i)}, ..., u_L^{(i)})$, is calculated as the $bitwiseMinimum$, or the $bitwiseAND$ of safety vectors among all the nodes that have the same coordinates as node u, except at dimension i.
The calculation of the safety vectors is according to the
$ExtendedGlobalStatus(EGS)$ algorithm:
$Base$:
if u is an end node of a faulty link: $u_1 = 0$, otherwise: $u_1 = 1$
$Induction$:
if $\sum_{i=1}^{L} u_{k-1}^{(i)} \leq n-k$, $u_1 = 0$, otherwise: $u_1 = 1$

5.4 Routing Using Safety Vectors

The routing process classifies dimensions as one of preferred, spare, and semi-preferred dimensions. Among neighbors ($S_k - 1$ for dimension k), only one node has the same ith coordinate as the one in destination node. This node is called the $preferredneighbor$. All the other neighbors along this same preferred dimensions, are called $semi - preferredneighbors$. All the other neighbors, located at other dimensions, are called $spareneighbors$. Note that it is possible to have multiple preferred dimensions, therefore multiple preferred neighbors.

The proposed routing algorithm using safety vectors contains two stages, one for the source node and the other for intermediate nodes. Safety vectors routing algorithm:
begin

while $currentnode != destinationnode$:

if $currentnode == sourcenode$:
if for some $offsetdimension\ i$, it holds that $s_{H-1}^{(i)} = 1$:
route to the preferred neighbor at dimension i.

else if for some $non - offsetdimension\ i$, it holds that $s_{H+1}^{(i)} = 1$:
route to the spare neighbor at dimension i.

if $currentnode == intermediatenode$:
if for some $offsetdimensioni$ there exists some preferred neighbor where its safety vector holds $u_{H-1}^{(i)} = 1$:
route to the preferred neighbor at dimension i.

else if for some semi preferred neighbor its safety vector holds $u_H^{(i)} = 1$:
route to the semi preferred neighbor at dimension i.

else if for some spare neighbor its safety vector holds $u_H^{(i)} = 1$:
route to the spare neighbor at dimension i.

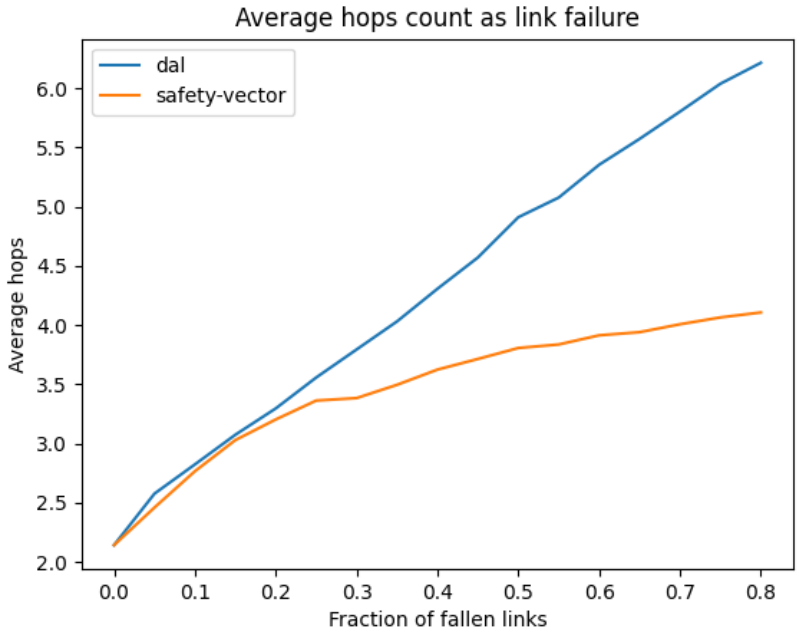
end

The safety-vectors routing algorithm purposes a semi-global optimization for minimizing the average hop count, under faulty links. We assume that its performance will outcome the local-optimization techniques, such as DAL routing algorithm.

5.5 Simulation Result

Simulations were performed using Python 3.8. The HyperX was configured the following dimensions: [4, 3, 3, 3]. The relative link capacity is constant among all dimensions, and the number of terminals each switch is connected to is 5.The traffic pattern assumed to be uniform.

article graphicx



As we can see, compared to DAL, the safety vectors routing procedure offers a better fault-tolerance approach. The average number of hops using the safety vectors method is lower than DAL, especially for very faulty networks.

All of the simulations are located within the added python files.

6 Conclusions

In this project, i was exposed to academic research for the first time. I learned multiple articles regarding the HyperX, its properties and potential, as well as basic routing algorithms (such as DAL). My goal was to expand the knowledge about the HyperX, and to describe a routing algorithm that may handle faulty links in an efficient manner. For this goal, I have read multiple articles regarding state-of-the-art routing techniques, especially focusing on fault tolerance approaches, including link-disjoint techniques.

Finally, i read articles focusing on fault tolerance routing on cubes based topologies, and adjusted their solution for the case of an HyperX.

Using simulations written with python, i showed that the safety-vectors method

is indeed an improvement in terms of fault tolerance to existing routing algorithms for the HyperX topology.

References

[1] Dennis Abts and John Kim. *High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities.* Morgan Claypool Publishers, 2011.

[2] Saar Eliad Vladimir Zdornov Barak Gafni Alexander Shpiner, Zachy Haramaty and Eitan Zahavi. *Dragonfly+: Low Cost Topology for Scaling Datacenters.* In IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB@HPCA), 2017.

[3] Maciej Besta and Torsten Hoefler. *Slim Fly: A Cost Effective Low-Diameter Network Topology.* In International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2014.

[4] Laxmi N. Bhuyan and Dharma P. Agrawal. *Generalized Hypercube and Hyperbus Structures for a Computer Network.* IEEE Trans. Computers 33, 4, 323–333, 1984.

[5] Bogdan Prisacari Germán Rodríguez Georgios Kathareios, Cyriel Minkenberg and Torsten Hoefler. *Costeffective diameter-two topologies: Analysis and evaluation.* In International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2015.

[6] Praveen Yalagandula Jayaram Mudigonda and Jeffrey C. Mogul. *Taming the Flying Cable Monster: A Topology Design and Optimization Framework for Data-Center Networks.* In USENIX Annual Technical Conference (ATC), 2011.

[7] Ivan R. Ivanov Yuki Tsushima Tomoya Yuki Akihiro Nomura Shin'ichi Miura Nie McDonald Dennis L. Floyd Jens Domke, Satoshi Matsuoka and Nicolas Dubé. *HyperX topology: First at-scale implementation and comparison to the fat-tree.* In International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Vol. 1, No. 1, Article . Publication date: November 2020., 2019.

[8] Steve Scott John Kim, William J. Dally and Dennis Abts. *Technology-Driven, Highly-Scalable Dragonfly Topology.* In International Symposium on Computer Architecture (ISCA), 2008.

[9] Al Davis Moray McLaren Jung Ho Ahn, Nathan L. Binkert and Robert S.Schreiber. *HyperX: Topology, routing, and packaging of efficient large-scale networks.* In ACM/IEEE Conference on High Performance Computing SC, 2009.

[10] Karolina Cynk Marek Konieczny Erik Henriksson Salvatore Di Girolamo Ankit Singla Maciej Besta, Marcel Schneider and Torsten Hoefler. *Fat-Paths: Routing in Supercomputers, Data Centers, and Clouds with Low-Diameter Networks when Shortest Paths Fall Short.* CoRR abs/1906.10885, 2019.

[11] Farshad Safaei Sadoon Azizi and Naser Hashem. *On the topological properties of HyperX.* J. Supercomputing 66, 1, 572–593., 2013.

[12] Farshad Safaei Sadoon Azizi and Milad Roozikhar. *A fault-tolerant routing algorithm in HyperX topology based on unsafety vectors.* J. Supercomputing 71, 4, 1224–1248, 2015.

[13] Jie Wu. *Adaptive Fault-Tolerant Routing Cube-Based Multicomputers Using Safety Vectors.* IEEE transactions on parallel and distributed systems, VOL. 9, NO. 4,, 1998.