Shortest Path Routing Algorithm for Ficonn in Load Balanced Data Center Networks

K Udaya Bhanu, K Chandra Shekar

Abstract--This paper explores a Ficonn interconnection structure for load balanced data center servers using bi-ports. The tree-based structures are increasingly difficult to meet the design goals of data centers because of a single-point failure spot for its sub tree branch. And using excessive switches does not fundamentally solve the problem, but results even higher cost. So we are going for a new structure to interconnect a large number of servers which contain dual Ethernet ports, in which one is used for network connections and another for the backup purposes. If we make that both the ports are used for network connections then we can build an effective inter connection structure for the data center networks. We call such a network structure called as Ficonn. In addition, in this paper, we are using A* algorithm to find the shortest path and boost up the routing of packets transmitted between the commodity servers. And we have also proposed the deployment of Ficonn.

Keywords*:* Ethernet ports, Data center, Networks, Routing

1. INTRODUCTION

The main aim of data center networking is to inter connect a large no. of servers with high performance, Ascendability, agility, Flexibility to support various services, Security. The present network architectures consists of switching elements arranged in a tree structure can't meet the design goals of a data center network.

Data center network must be able to provide high network capacity to better support bandwidth-hungry services. The network infrastructure must be ascendable to a large number of servers and allow for incremental expansion. Data center network must be fault tolerant against various types of server failures, link outages, or server-rack failures. Two observations motivate these goals. First, data center is growing large and the number of servers is increasing at an exponential rate. The current Data center network practice is to connect all the servers using a tree hierarchy of switches, core-switches or core-routers. With this solution it is increasingly difficult to meet the above three design goals. It is thus desirable to have a new network structure that can fundamentally address these issues in both its physical network infrastructure and its protocol design. To meet these goals we propose a novel network structure called Ficonn.

2. INTERCONNECTION STRUCTURE OF FICONN

Ficonn is a level-based iterative structure. A high-level Ficonn is constructed by using low-level Ficonns. When building a higher-level Ficonn, the lower-level Ficonns use half of their available backup ports for interconnections and form a mesh. Ficonn just needs to use the existing backup port on each server for interconnection, and no other hardware cost is introduced on a server. The wiring cost in Ficonn is less than any other structure because each server uses only two ports. Routing in Ficonn makes a balanced use of links at different levels; finally, with the A* algorithm, routing in Ficonn is further designed to exploit the link capacities according to current traffic state.

FiConn is a recursively defined structure. A high level FiConn is constructed by many low-level FiConns. Let us assume that level-k FiConn as FiConnk. FiConn0 is the basic construction unit, which is composed of servers and n-port switch connecting the servers. Typically n must be an even number such as 16, 32, or 48. Every server in FiConn has one port connected to the switch in FiConn0, and this is called level-0 port. The link connecting a level-0 port to the switch is called level-0 link. Level-0 port can be regarded as the original operation port on servers in current practice. If the backup port of a server is not connected to another server, this port is called available backup port. For instance, there are initially n servers each with an available backup port in a FiConn0. Now focus on how to construct FiConnk (k>0) upon FiConnk-1's by interconnecting the server backup ports. If there are totally b servers with available backup ports in a FiConn, the number of FiConnk-1's in a FiConnk, is equal to b/2+1.

In each FiConn(k-1), b/2 servers out of the servers with available backup ports are selected to connect the other b/2 FiConnk-1s using their backup ports, each for one FiConn(k-1) The b/2 selected servers are called level-k servers, the backup ports of the level-k servers are called level-k ports, and the links connecting two levelk ports are called levelk links. If take FiConn as a virtual server, FiConn is in fact a mesh over FiConn s connected by level-k links. A sequential number is to identify a server in FiConn. Assume the total number of servers in a FiConnk is Nk, and there is (0<=uk<nk) Equivalently can be identified by a(k+1) –tupel0,[ak,…,a1,a0] , where a0 identifies in its

FiConn0 , and a1(1<=l<=k) identifies the FiConn(l-1) comprising in its FiConnl. Fig.4 to illustrate the FiConn interconnection rule, in which n=4and k=2. FiConn0 is composed of four servers and a 4-port switch.

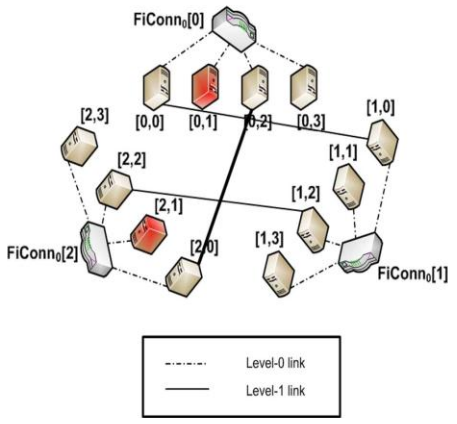

Fig.1.Construction of Ficonn1 with n=4, includes 3 Ficonn0

The number of FiConn0s to construct FiConn1 is 4/2+1=3. The servers [0,0], [0,2], [1,0],[1,2], [2,0], and [2,2] are selected as level-1 servers, and connection [0,0] with [1,0], [0,2] with [2,0], and [1,2] with [2,2]. In each FiConn, there are six servers with available backup ports, so the number of FiConn1's in a FiConn2 is 6/2+1=4 connection the selected level-2 servers. A level-1 FiConn includes three level-0 FiConn's, each consisting of four servers.

3. Routing In Ficonn:

In this paper, we are using A* search algorithm in order to find the shortest path between any pair of nodes so that we can balance the traffic in the network. A* algorithm is a variant of Djkstra's algorithm.

the classic representation of the A* algorithm.

$f'(n) = g(n) + h'(n)$

g(n) is the total distance the source to the current intermediate node.

h'(n) is the estimated distance from the current intermediate node to the destination node. A heuristic function is used to create this estimate on how far away it will take to reach the destination node.

f'(n) is the sum of g(n) and h'(n). This is the current estimated shortest path. f(n) is the true shortest path which is not discovered until the A* algorithm is finished.

One of the more difficult parts in solving A* is creating a good heuristic function to determine h'(n). A heuristic function differs from an algorithm in that a heuristic is more of an estimate and is not necessarily provably correct. An algorithm is a set of steps which can be proven to halt on a particular given set of input.

The heuristic function in A* is arbitrary, however the better your heuristic is, the faster and more accurate your solution will become. However, therein lies the problem -- deciding a good heuristic. Even with a shortest path example, the heuristic can change, depending on the implementation of the search, and how easy or complicated the heuristic function is going to be.

```
/* src: source node
dst:destination node
g[s]:shortest path
f[s]:path from the source to the next
c[ ]:nodes to be visited
v[ ]:visited nodes
nxt: neighbor next node
prv: previously visited node
h:heuristic_cost_estimate
crnt: current node
dis: distance */
A*algorithm(src,dst)
{
   c=0
   V[]= {src}
   prv=0
   g[src]=0
   f[src]= g[src] + h(src, dst)
   while(v!=NULL)
   crnt= v[]
   if crnt = dst
   return reconstruct_path(prv, d)
   v[]=v[crnt+1]
   c[]=c[]+ crnt
   while(nxt!=NULL)
   if (c!=nxt)
   continue
   g:= g[crnt] + dis(current,nxt)
   if (!nxt || g< g[nxt])
   {
   o=o+nxt
   prv[nxt] := crnt
   g[nxt] := g
   f[nxt] := g[nxt] + h(nxt, dst)
   return 1
 function reconstruct_path(prv, crnt)
   if (prv[crnt]==0)
      p := reconstruct_path(prv, prv[crnt])
      return (p + crnt)
   else
      return crnt
}
```

While A* is generally considered to be the best pathfinding algorithm (see rant above), there is at least one other algorithm that has its uses - Dijkstra's algorithm. Dijkstra's is essentially the same as A*, except there is no heuristic (H is always 0). Because it has no heuristic, it searches by expanding out equally in every direction. As you might imagine, because of this Dijkstra's usually ends up exploring a much larger area before the target is found. This generally makes it slower than A*.

4. Deployment of Ficonn:

In practice, it is much likely that the total number of servers need in FiConn does not exactly meet the number of servers in a certain FiConnk. Instead, the number is between FiConnk and FiConnk+1.such a structure called an incomplete FiConn. For incremental deployment, the need is to address the interconnection in incomplete FiConns. Our principle is that the interconnection should not only retain high bisection width in incomplete FiConn, but also incur low rewiring cost.

In the construction of complete FiConn, A bottom-up approach is used. In this way, first deploy a complete FiConn1, then a complete FiConn2, and so forth. One problem of this approach is that it may generate incomplete FiConn with low bisection width. For example, if there are only two FiConnk-1's in an incomplete FiConn, the two FiConnk-1's will be connected by a single level-k link. The bisection width of this structure becomes 1, and the single level-k link is the communication bottleneck in the structure.

FiConn is built in a bottom-up way, but add level-k shortcut links to increase the bisection width in an incomplete FiConnk. Assume there are m deployed FiConnk-1's in an incomplete FiConnk, and the number of FiConnk-1's in a complete FiConnk is m'. Thus, the number of undeployed FiConnk-1's in the complete FiConnk is m'-m. For a certain undeployed FiConnk, each of the m deployed FiConnk-1 has a level-k server connecting toward it in the complete FiConnk.

As there are 'm' deployed FiConnk-1 's in an incomplete FiConnk, and the number of FiConnk-1 's in a complete FiConnk is m. Now, we add a new FiConnk-1 into the incomplete FiConnk. The wiring is as follows. First, for each of the level-k shortcut links that connect the level-k servers that should connect it to the newly added FiConnk-1, we unplug one end and connect it to the new FiConnk-1 . Second, m/2 level-k links are added to connect the new FiConn and each remaining deployed FiConn . Third, if is an odd number, m'-m-1level-k shortcut links are added, each connecting the new FiConnk-1 and a corresponding deployed FiConnk-1, based on the shortcut link addition rule.

We find that the shortcut links we add in an incomplete FiConn not only increase the bisection width of the incomplete FiConn, but are also fully utilized during the process of incremental deployment and easy to rewire.

5. Conclusion

In this paper, we proposed Ficonn inter connection structure for the load-balanced data center networks that uses two ports of a data center server. We proposed solutions to increase the bisection width in FiConns. We also proposed the design of FiConn, an inter connection structure for data centers that uses dual ports and eliminates the requirement of high-cost switches other than the lowest-level commodity ones. It is highly ascendable because it can support any no. of servers and also cost effective since it does not use more no. of switches and wiring is also simple in the proposed structure, that makes the datacenter network simple. A* algorithm in routing makes use of different levels of links that are interconnected. The shortest path is computed hop-by-hop by each intermediate server based on the available bandwidths of its two outgoing links. Our future work involves evaluating the performance of A* algorithm used in Ficonn.

6. References

[1] M.Al-Fares, A.Loukissas, and A. Vahdat, "A ascendable, commodity data center network architecture," in Proc. ACM SIGCOMM, Aug. 2008, pp. 63–74.

[2] H.Sullivan and T. R.Bashkow, "A large scale, homogeneous, fully distributed parallel machine, I," in Proc. ISCA, Mar. 1977, pp. 105–117.

[3] J.Snyder, "Microsoft: Datacenter growth defies Moore's Law," 2007 [Online]. Available: http://www.pcworld.com/article/id,130921/article. Html

[4] L.Bhuyan and D.Agrawal, "A general class of processor interconnection strategies," in Proc. ISCA, Apr. 1982, pp. 90–98.

[5] C.Guo, H.Wu, K.Tan, L. Shi, Y. Zhang, and S. Lu, "DCell: A ascendable and faulttolerant network structure for data centers," in Proc. ACM SIGCOMM, Aug. 2008, pp. 75– 86.

[6] "Dell powerage servers" [Online]. Available: http://www.dell.com/content/products/category.aspx/servers

[7] J.Dean and S.Ghemawat, "Map Reduce: Simplified data processing on large clusters," in Proc. OSDI, 2004, pp.

[8] M.Isard, M.Budiu, and Y. Yu et al., "Dryad: Distributed data-parallel programs from sequential building blocks," in Proc. ACM EuroSys, 2007, pp. 59–72.

[9] T. Hoff, "Google architecture," Jul. 2007 [Online]. Available: http://highascendability.com/google-architecture

[10] C.Guo, G.Lu, D.Li, H.Wu, X.Zhang, Y. Shi, C.Tian, Y. Zhang, and S. Lu, "BCube: A high performance, server-centric network architecture for modular data centers," in Proc. ACM SIGCOMM, Aug. 2009, pp. 63–74.

[11] J. Kim, W. Dally, S. Scott, and D. Abts, "Technology-driven, highly ascendable dragonfly topology," in Proc. ISCA, Jun. 2008, pp. 77–88.

[12] L. Bhuyan and D. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," IEEE Trans. Comput., vol. C-33, no. 4, pp. 323–333, Apr. 1984.

[13] M. Isard, M. Budiu, and Y. Yu et al., "Dryad: Distributed data-parallel programs from sequential building blocks," in Proc. ACM EuroSys, 2007, pp. 59–72.

[14] A. Carter, "Do it green: Media interview with Michael Manos," Dec. 2007 [Online]. Available: http://edge.technet.com/Media/Doing-ITGreen

[15] L. Rabbe, "Powering the Yahoo! network," Nov. 2006 [Online]. Available: http://yodel.yahoo.com/2006/11/27/powering-the-yahoo-network

[16] F. Leighton, Introduction to Parallel Algorithms and Architectures: Arrays. Trees. Hypercubes. San Mateo, CA: Morgan Kaufmann, 1992.

Short Bio Data for the Author

Mr. K. Udaya Bhanu received his B.Tech in Computer science and Engineering from Indur Institute Of Technology And Sciences, JNTU, Hyderabad and Pursuing M.Tech in Computer science and Engineering from Aurora's Technological And Research Institute, JNTU, Hyderabad.

Email: udayabhanukonne@gmail.com

Mr. K. Chandra Shekhar, working as an Associate Professor in the Department of Computer Science and Engineering. Aurora's Technological and Research Institute College with a teaching experience of 8years. He has received his M.Tech in Computer science. His areas of interest include Computer Networks, Data mining, and Information Security.

Email: chandhra2k7@gmail.com