BCube:
A High Performance, Server-centric Network Architecture for Modular Data Centers

Article by: Chuanxiong Guo, Guohan Lu et.al.

Almasi Adela-Diana
SCPD Master
"Politehnica" University, Bucharest

Table of contents

What is BCube?

- A new network architecture
- Designed for shipping-container based, modular data centers
- server-centric network structure
- Servers act as:
  - end hosts
  - relay nodes for each other

Glossary of terms

- MDC = modular data center
- COTS = commodity off-the-shelf mini-switches
- $h(A,B)$ = the Hamming distance of two servers A and B (the number of different digits of their address arrays)

Context

- **MDC**: a few thousands of servers are interconnected via switches to form the network infrastructure
- High degree of mobility
- 40-feet container: 12m x 2.35m x 2.38m

Restrictions

- Impossible to service the MDC once sealed & deployed
- Bandwidth intensive application support for typical traffic patterns:
  - **One-to-one** – basic traffic pattern, high inter-server throughput
  - **One-to-several** – needed for HDFS, GFS replication
  - **One-to-all** – upgrade system image, distribute binaries
  - **All-to-all** – useful in MapReduce reduce phase

THE BCUBE STRUCTURE (1)

- BCUBE is made of:
  - Servers – with multiple ports
  - Switches – connect a constant no.servers
- recursively defined:
  - $BCube_1$ = n $BCube_0$s and n^1 n-port switches.
  - $BCube_k$ = n $Bcube_{k-1}$s and n^k n-port switches
- Each server in a $BCube_k$ has $k + 1$ ports, which are numbered from level-0 to level-k

THE BCUBE STRUCTURE (2)

- Notations in $BCube_k$:
- A server = An address array

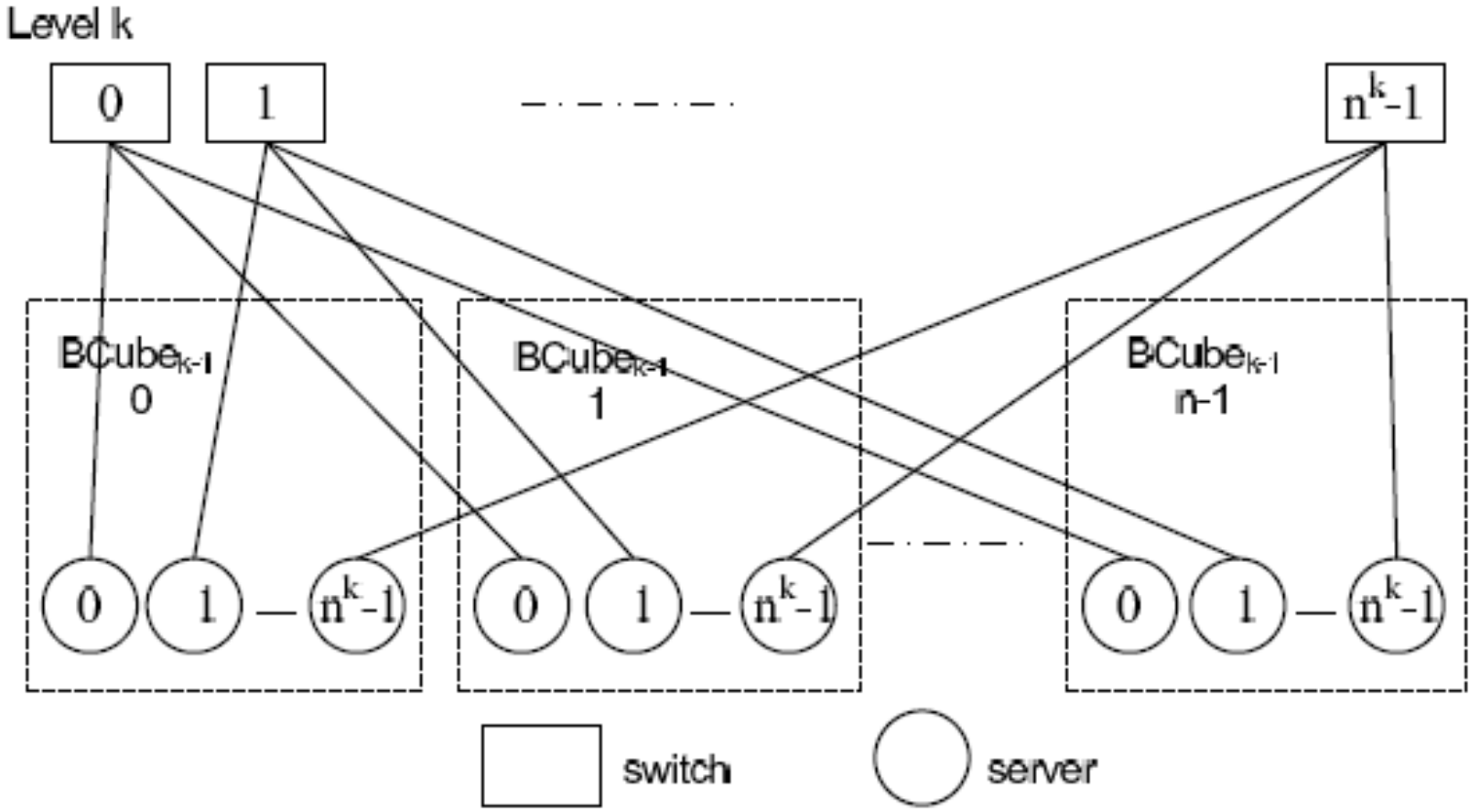
$$a_k a_{k-1} \cdots a_0 \ (a_i \in [0, n-1], i \in [0, k])$$

$$BCube\ address\ baddr = \sum_{i=0}^{k} a_i n^i$$

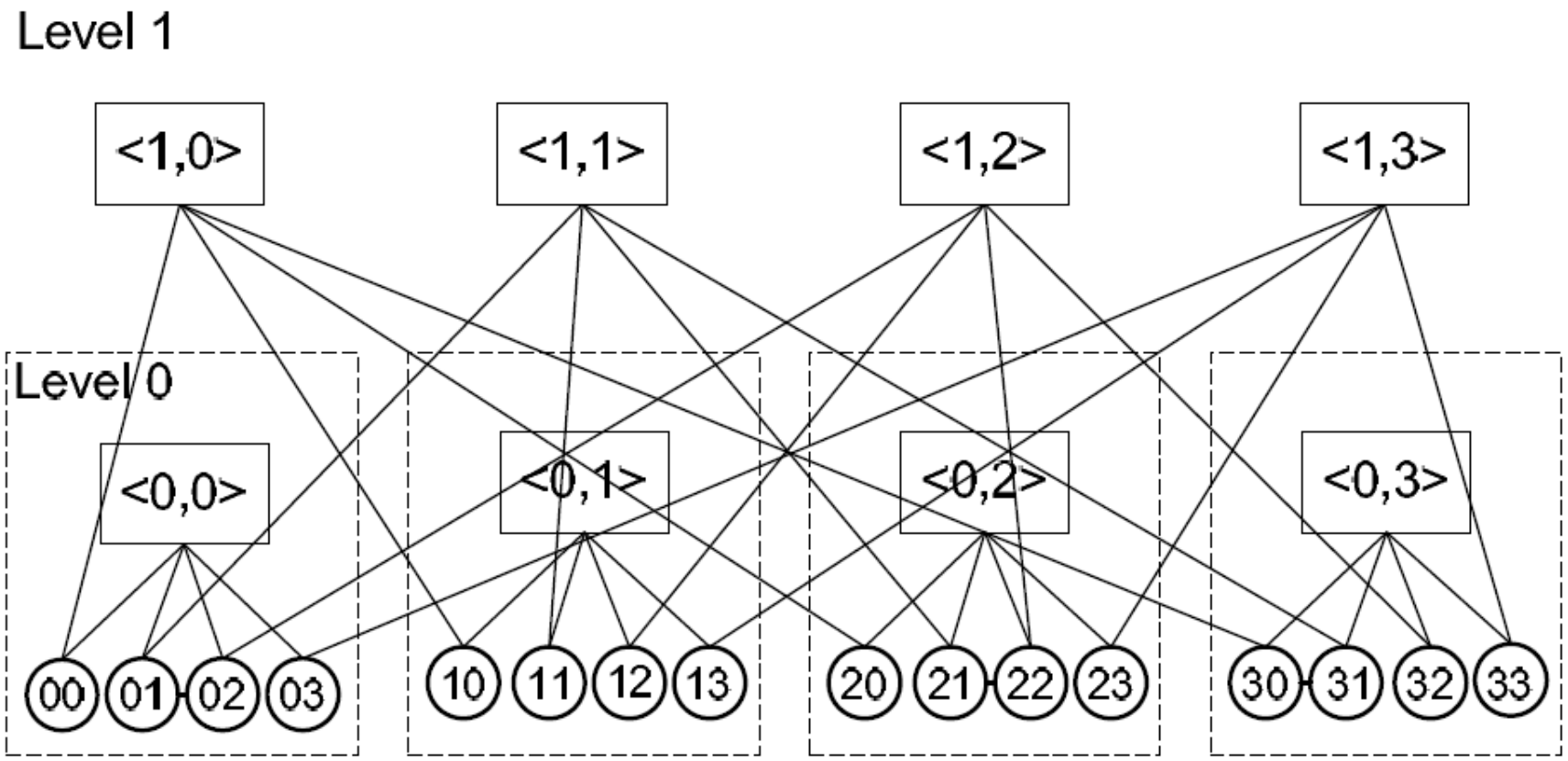
- A switch = < *l*; s_{k-1} s_{k-2} … s_0 >

Where (*sj in* 2 [0; *n*-1]; *j in* [0; *k*-1]), *l* (0 ≤ *l* ≤ *k*) is the level of the switch.

A BCube$_k$ = n Bcube$_{k-1}$ and n^k n-port switches



A $BCube_1$ with $n = 4$



THE BCUBE STRUCTURE (3)

- The BCube construction guarantees that:
- switches only connect to servers and never directly connect to other switches.

⇒we can treat the switches as dummy crossbars that connect several neighboring servers

⇒Let servers relay traffic for each other.

- With 8-port mini-switches, we can support up to 4096 servers in one BCube3.

Remark: Hypercube

- In a BCube network, if we replace each switch and its *n* links with an *n £x* (*n* -1) full mesh that directly connects the servers
- => we get a generalized Hypercube.
- Compared to the generalized Hypercube:
- the server port number is much smaller in BCube.
- a $BCube_k$: It is *k* + 1
- generalized Hypercube: (*n* - 1)(*k* + 1)

Single-Path Routing

- the maximum Hamming distance in a BCube*k* is *k* + 1.
- Two servers are *neighbors* if they connect to the same switch.
- Theorem 1. *The diameter, which is the longest shortest path among all the server pairs, of a* $BCube_k$*, is* $k + 1$.

Single-path routing (2) - Algorithm

- A=a_k a_{k-1} … $a0$ is the source server
- B=bk bk-1 … $b0$ is the destination server
- Π is a permutation of [k; k-1; … ; 1; 0].
- We systematically build a series of intermediate servers by `correcting' one digit of the previous server.
- Hence the path length is at most k+1.

Single-path Routing (3) – Algorithm

/*

A=$a_k a_{k-1} \cdots a_0$ and B=$b_k b_{k-1} \cdots b_0$; $A[i] = a_i$; $B[i] = b_i$;

$\Pi = [\pi_k, \pi_{k-1}, \cdots, \pi_0]$ is a permutation of $[k, k-1, \cdots, 1, 0]$

*/

BCubeRouting(A, B, Π):

$path(A, B) = \{A, \}$;

$I\_Node = A$;

for($i = k; i \geq 0; i--$)

if ($A[\pi_i] \neq B[\pi_i]$)

$I\_Node[\pi_i] = B[\pi_i]$;

append I_Node to $path(A, B)$;

return $path(A, B)$;

Multi-paths for One-to-one Traffic (1)

- Two *parallel* paths between a source server and a destination server exist if they are **node-disjoint**

THEOREM 2. *Given that two servers* $A = a_k a_{k-1} \cdots a_0$ *and* $B = b_k b_{k-1} \cdots b_0$ *are different in every digit (i.e.,* $a_i \neq b_i$ *for* $i \in [0, k]$*). BCubeRouting generates two parallel paths from A to B using two permutations* $\Pi_0 = [i_0, (i_0-1) \mod (k+1), \cdots, (i_0-k) \mod (k+1)]$ *and* $\Pi_1 = [i_1, (i_i-1) \mod (k+1), \cdots, (i_1-k) \mod (k+1)]$ *(*$i_0 \neq i_1$ *and* $i_0, i_1 \in [0, k]$*).*

Multi-paths for One-to-one Traffic (2)

- The permutations PI_0 and PI_1 start from different locations of the address array and then correct the digits sequentially.

=> used switches are always at different levels for the same digit position, thus producing the two parallel paths

Multi-paths for One-to-one Traffic (3)

/*A=$a_k a_{k-1} \cdots a_0$ and B=$b_k b_{k-1} \cdots b_0$; $A[i] = a_i$; $B[i] = b_i$;*/
BuildPathSet(A, B):
  PathSet = { };
  for$(i = k; i \geq 0; i--)$
    if $(A[i] \neq B[i])$
      P_i=DCRouting(A, B, i);
    else /*$A[i] == B[i]$*/
      C= a neighbor of A at level i; /*$C[i] \neq A[i]$*/
      P_i=AltDCRouting(A, B, i, C);
    add P_i to PathSet;
  return PathSet;

DCRouting(A, B, i):
  $m = k$;
  for $(j = i; j \geq i - k; j--)$
    $\Pi[m] = j \mod (k + 1)$; $m = m - 1$;
  path = BCubeRouting(A, B, Π);
  return path;

AltDCRouting(A, B, i, C):
  path={A,};
  $m = k$;
  for $(j = i - 1; j \geq i - 1 - k; j--)$
    $\Pi[m] = j \mod (k + 1)$; $m = m - 1$;
  path += BCubeRouting(C, B, Π);
  return path;

One-to-several Traffic

- edge-disjoint complete graphs with $k + 2$ servers can be efficiently constructed in a $BCube_k$.

Application of One-To-Several Traffic: GFS, HDFS

- File divided into chunks, and each chunk is replicated several times for reliability
- Source and the selected chunk servers form a pipeline

The complete graph built in BCube good for chunk repl. because:

1. selected servers at different levels of Bcube => reliable
2. **edge-disjoint complete graph** is perfect for chunk replication speedup.

When a client writes a chunk to r ($r <= k + 1$) chunk servers, it sends $1/r$ of the chunk to each of the chunk server. Meanwhile, every chunk server distributes its copy to the other r-1 servers using the disjoint edges.

=> r times faster than the pipeline model.

Speedup for One-to-all Traffic

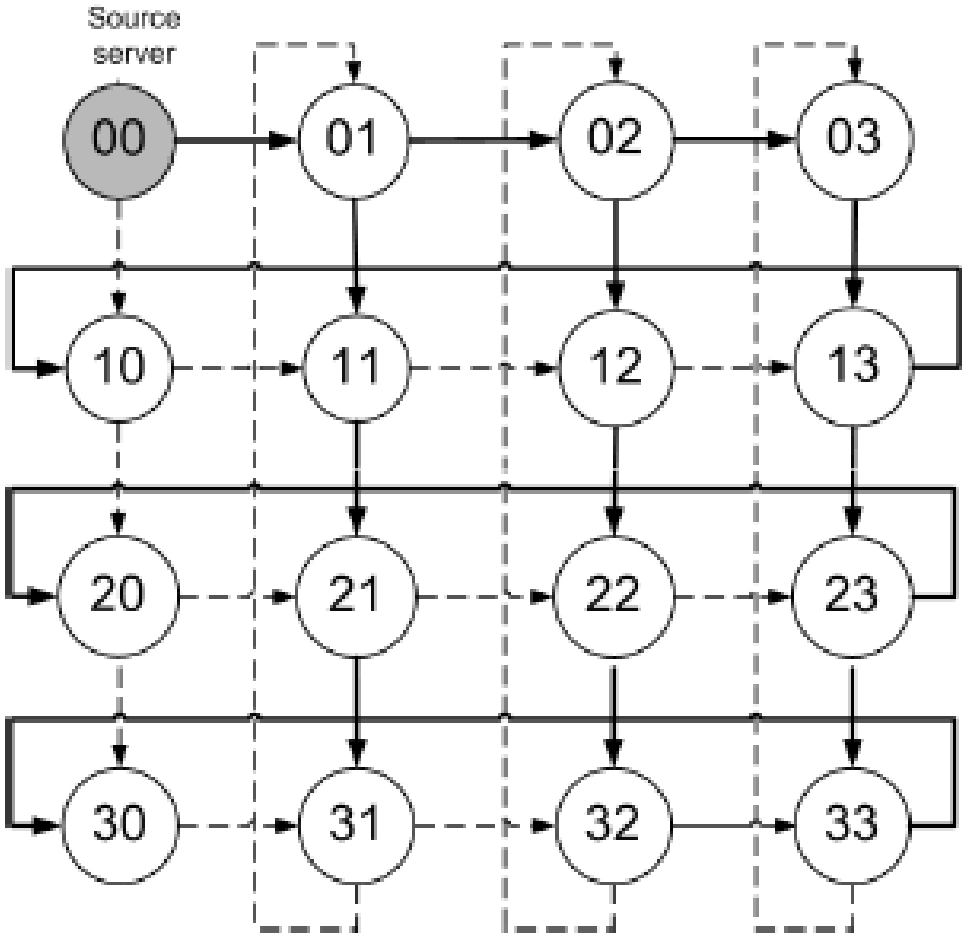
THEOREM 5. *A source can deliver a file of size L to all the other servers in $\frac{L}{k+1}$ time in a $BCube_k$.*

- a source server delivers a file to all the other servers.

L = file size

- We assume all the links are of bandwidth 1.
- We omit the propagation delay and forwarding latency.

The two edge-disjoint server spanning trees with server 00 as the source for the BCube1



Speedup for One-to-all Traffic (2)

- Source can split file in k+1 parts
- Send each part through a different spanning tree
- A receiving server is in all SPT => it gets all parts

⇒delivery time L / (k+1)

⇒No broadcast or multicast needed

BSR = BCUBE SOURCE ROUTING (1)

- the source server decides which path a packet flow should traverse
- probing the network
- Encodes the path in the packet header.
- We select source routing because:

1. the source can control the routing path without coordination of the intermediate servers.
2. intermediate servers do not involve in routing and just forward packets based on the packet header.

BSR = BCUBE SOURCE ROUTING (2)

When a new flow comes:

- SOURCE: sends probe packets over multiple parallel paths.
- INTERMEDIATE SERVERS: process the probe packets to fill the needed information
- DESTINATION: returns a probe response to the source.
- SOURCE: receives the responses
- SOURCE: uses a metric to select the best path, e.g., the one with maximum available bandwidth.

REMARK: When a source is performing path selection for a flow, it does not hold packets.

Thank you for your attention!

Questions?