International Journal of Web Services Research
Volume 15 • Issue 1 • January-March 2018

IBCube:
An Economical and Incremental Datacenter Network

Qiong Hu, Services Computing Technology and System Lab, Big Data Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China

Hanhua Chen, Services Computing Technology and System Lab, Big Data Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China

Hai Jin, Services Computing Technology and System Lab, Big Data Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China

Chen Tian, State Key Lab for Novel Software Technology, Nanjing University, Nanjing, China

Aobing Sun, G-Cloud Science and Technology Co., Ltd, Cloud Computing Center, Chinese Science Academy, Guandong, China

Tongkai Ji, G-Cloud Science and Technology Co., Ltd, Cloud Computing Center, Chinese Science Academy, Guandong, China

ABSTRACT

Datacenter networks have attracted a lot of research interest in the past few years. BCube is proved to be a promising scheme due to its low cost. By using a recursive construction scheme, BCube can exponentially scale a datacenter. Industry experiences, however, articulate the importance of incremental expansion of datacenter. In this article, the authors show that BCube's expanding scheme suffers low utilization of switch ports. They propose IBCube, a novel economical design for incrementally building datacenter networks. The insight is that: by letting the number of switches in each BCube layer equal the number of the building blocks, the authors can enable the switch ports to be fully utilized to support the total number of network interface cards of the deployed servers in the datacenters. Accordingly, their IBCube designs a novel automatic port allocation scheme. Simulation results show that the IBCube design reduces the budget for the datacenter networks by 94% as well as improves the packet delay and throughput by 10.3% and 11.5%, respectively, compared to the previous partial BCube design.

KEYWORDS

Data Center, Data Center Networks, Incremental Data Center Networks, Routing, Topology

1. INTRODUCTION

With the emergence of cloud services and applications, how to efficiently build datacenters becomes an important issue (Liu et al. 2016). Maximizing the profits of datacenters is a major concern of datacenter operators for economical consideration (Zhan et al. 2016). Among the budget of a fully functional datacenter, a fraction of 15% goes to networking, *i.e.*, the network equipments and the wires (Greenberg et al. 2008). In recent years, there have been a number of proposals for efficient datacenter networks (Lee et al. 2016; Ports et al. 2015; Zhu et al. 2015; Perry et al. 2014). Existing

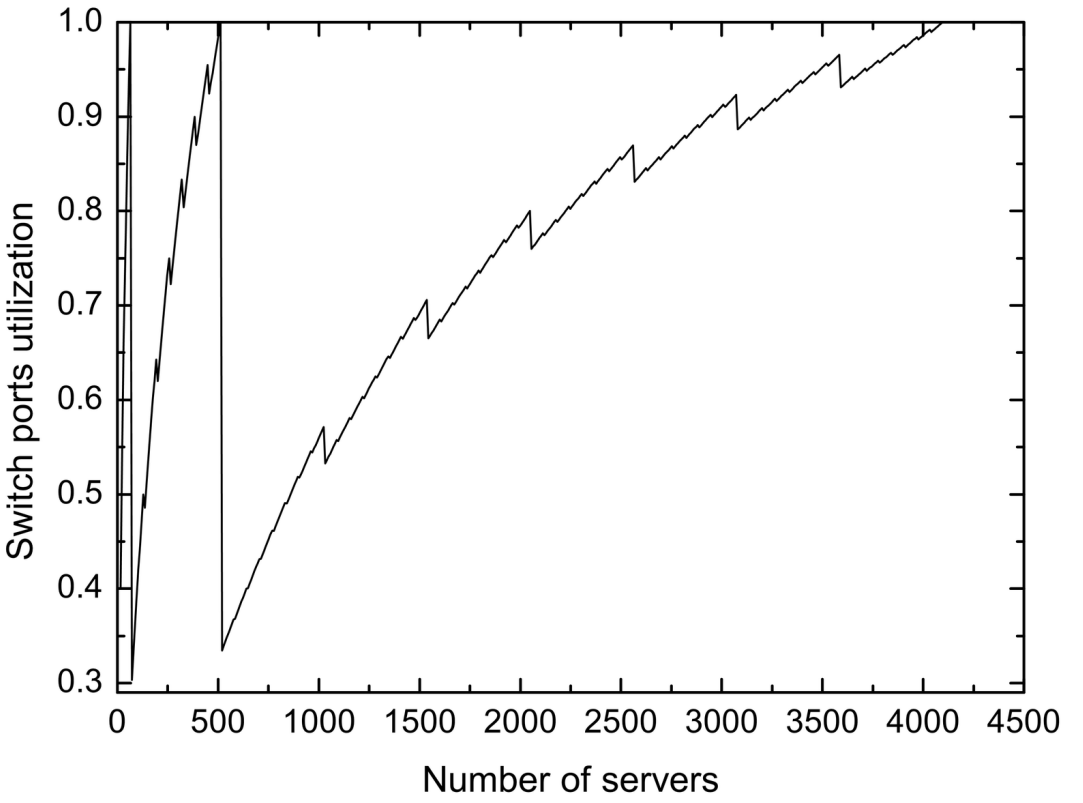
DOI: 10.4018/IJWSR.2018010102

systems adopt quite different approaches for constructing datacenter networks (Al-Fares et al. 2008; Guo et al. 2009; Guo et al. 2008).

Existing datacenter network architectures can be classified into three styles, switch-centric, server-centric and hybrid designs (Popa et al. 2010). The switch-centric architecture utilizes switches for packet forwarding (e.g., the fat-tree based architecture (Al-Fares et al. 2008) uses switches for packet forwarding; Arjun et al. (2015) propose to use Clos topologies for connecting the switches to achieve good scalability). Different from the switch-centric design, the server-centric architecture relies on servers for packet forwarding, i.e., packets are forwarded between servers instead of switches. For instance, Abu-libdeh et al. (2010) design an architecture which connects servers using a 3D torus structure. By considering hybrid architectures, Guo et al. propose the DCell (2008) and BCube (2009) structures. DCell (Guo et al. 2008) and BCube (Guo et al. 2009) architectures forward packets using a combination of switches and servers. To evaluate the cost efficiency of existing diverse datacenter networking designs, Stoica et al. (2010) propose a high-level model to quantify and compare the cost of a datacenter network. Stoica's results show that BCube achieves the lowest cost among those architectures (Popa et al. 2010).

In BCube, a server with multiple network ports connects to multiple layers of *Commodity Off-The-Shelf* (COTS) mini- switches (Guo et al. 2009). The servers in BCube act as not only the end hosts, but also relay nodes for network communication. Formally, BCube is a recursively constructed structure, where a BCube0 is constructed by connecting n servers to an n-port switch, while a BCube$_k$ ($k \geq 1$) is constructed from n BCube$_{k-1}$s and a new switch layer with n^k n-port switches for connecting those BCube$_{k-1}$s (In Section 2, we will review the BCube design in more detail). Thus, a BCube$k$ has n^{k+1} servers, where the number increases exponentially with the value of k. However,

Figure 1. The port utilization in partial BCube$_k$ with n=8 ($N \in$ [2, 512])



industry experiences articulate the importance of incremental expansion in datacenters. For example, Facebook's datacenters add capacity on a dailybasis by incrementally expanding existing facilities (Datacenterknowledge, n.d.); SGI's IceCube modular datacenter expands four racks at a time (SGI-Products, n.d.). To support an arbitrarily incremental structure, a datacenter requires its network capacity to increase on demand.

In practice, for constructing a $BCube_k$, the design first builds the needed $BCube_{k-1}$s and then connects those $BCube_{k-1}$s using level-k switches. It is not difficult to see that a par- tial BCube suffers from the problem of low utilization of network equipment. If the $BCube_{k-1}$s in a partial $BCube_k$ are complete $BCube_{k-1}$ structures, the switches in level-k of the partial $BCube_k$ are commonly not fully utilized and the increasing granularity of this partial $BCube_k$ with n-port switches is n^k. To support incremental expansion, a partial $BCube_k$ needs to be built with a number of (less than n) complete $BCube_{k-1}$s and a possible partial $BCube_{k-1}$. It is clear that such a design may have much lower switch port utilization than the $BCube_k$ which is built with complete $BCube_{k-1}$s. For example, the switch port utilization in a partial BCube1 with two BCube0s and 8-port switches is only 40%. Figure 1 illustrates the utilization of switch ports based on the analysis on BCube with different number of servers. The low utilization of the switch ports in partial $BCube_k$ leads to a serious waste of network equipment investment.

Figure 2. The network structure of $BCube_k$

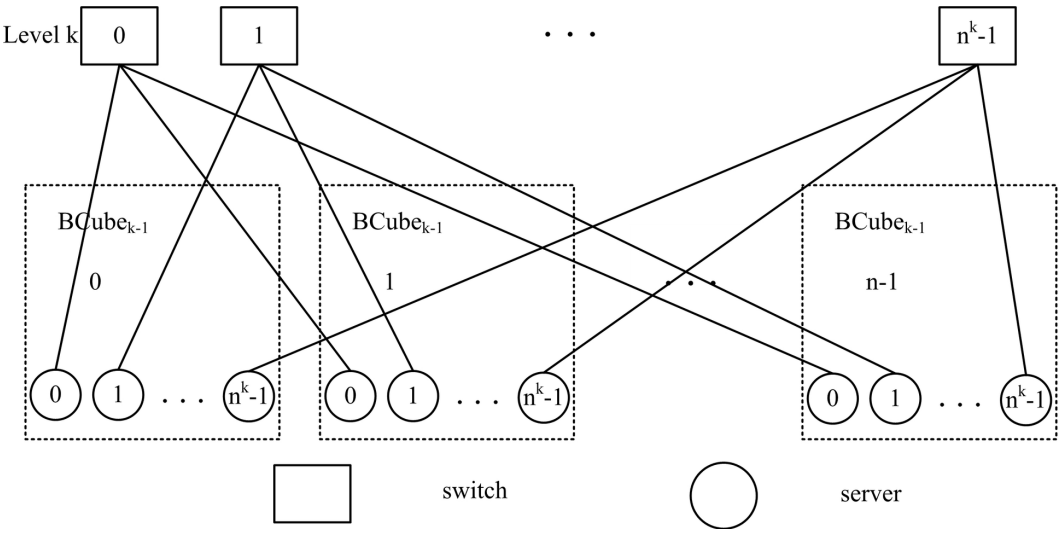
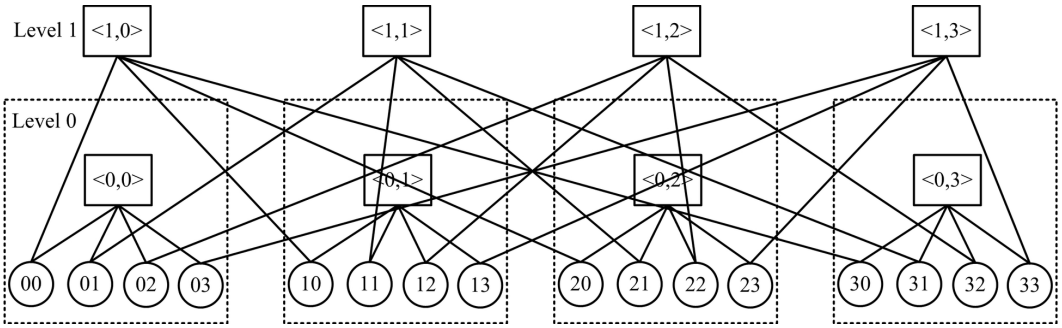


Figure 3. The network structure of $BCube_1$ with $n = 4$



To solve the problem, we design an novel structure, called IBCube (incremental BCube). Our observation is that in BCube structure if the number of switches in each layer equals the number of the building blocks (*i.e.,* $BCube_0$s), the ports can be fully utilized. Based on the observation, we design a novel port allocation scheme which elaborately utilizes the ports to achieve a minimal number of required switches. We further design a routing algorithm to exploit the merits of IBCube design. Compared to the previous BCube expanding design (Guo et al. 2009), IBCube achieves three advantages. First, IBCube supports the network structure with an arbitrary number of servers. Second, the switch ports in each layer are fully utilized to minimize the expenses. Third, IBCube minimizes the rewire tasks to further reduce the cost. We conduct comprehensive simulations based on real world system configurations to evaluate this design. The results show that our IBCube design significantly cuts down the budget for datacenter networking as well as achieves better latency and throughput than the previous partial BCube design (Guo et al. 2009).

The rest of the paper is organized as follows. Section 2 introduces the background of the BCube and the problem in a partial BCube. Section 3 presents the design of our IBCube and the routing algorithm. Section 4 evaluates the performance of our design. Section 5 concludes the paper.

2. RELATED WORK

In this section, we first review the related work of datacenter networks. Then we introduce the background information of the BCube (Guo et al. 2009) architecture, which is most related to our design. At last we present the problem of the partial BCube and the motivation of this design in detail.

2.1. Datacenter Networks

The network plays more and more important role in today's datacenters. They have recently attracted a lot of research interest in the community (Lee et al. 2016; Ports et al. 2015; Zhu et al. 2015; Perry et al. 2014; Gyarmati et al. 2013). Among the existing designs, the most fundamental work mainly focuses on improving the scalability and cost-effectiveness of the datacenter network architecture. Generally, existing datacenter network architectures can be classified into three types, switch-centric, server-centric and hybrid designs (2010). The switch- centric architecture relies on switches for packet forwarding. Traditional switch-centric designs, such as VL2 (Greenberg et al. 2008) and etc., use tree structures to inter-connect the switches in the datacenter network. Mohammad et al. (2008) organize the switches using a fat-tree structure to improve the capacity of the root of a tree. Recently, Arjun et al. (2015) use Clos topologies to inter-connect the switches. Such a scheme achieves good scalability for different datacenter sizes by adding stages to the topology. Different from the switch-centric design, the server-centric datacenter architecture relies on servers for both server inter-connecting and packet forwarding. Different from the switch-centric designs, the packets of the server centric datacenter network are forwarded between servers instead of switches. For example, Abu-libdeh et al. (2010) design an architecture which connects servers using a 3D torus structure. By considering hybrid architectures, Guo et al. propose DCell (2008) and BCube (2009) structures. In such designs, servers and switches are combined for the function of inter-connection and packet forwarding.

2.2. Background of BCube Design

BCube is a novel network structure for datacenter (Guo et al. 2009). The BCube design employs the server-centric architecture, rather than the switch-centric architecture. Following the server-centric design philosophy, BCube builds a novel interconnection structure as well as the routing protocol. BCube connects servers with a small number of network ports to multiple COTS mini-switch layers and puts routing intelligence at the server side. In BCube, the multiple parallel paths between any pair of servers provide high one-to-one bandwidth and achieve good fault tolerance. Such a design also accelerates one-to-several and one-to-all traffic. The low diameter of BCube provides high network capacity for all-to-all traffic. Thus, BCube supports various bandwidth-intensive applications. In the

following, we follow the framework pro- posed by Guo *et al.* (2009) and briefly introduce the structure of BCube and its routing algorithm and the problem of the partial BCube design.

There are two types of devices in BCube: servers with multiple ports and *n*-port COTS mini-switches. BCube is a recursively defined structure. A BCube_0 includes *n* servers connecting to an *n*-port switch, and a BCube_1 is constructed with *n* BCube_0s and *n* *n*-port switches. Generally, a BCube_k ($k \geq 1$) is constructed with n $BCube_{k|-|1}s$ and n^k *n*-port switches. In a BCube_k structure, each server has $k+1$ ports, which are numbered from level-0 to level-*k*. Thus, a BCube_k has $n^{k|+|1}$ servers and $k+1$ level of switches, where each switch level has n^k *n*-port switches.

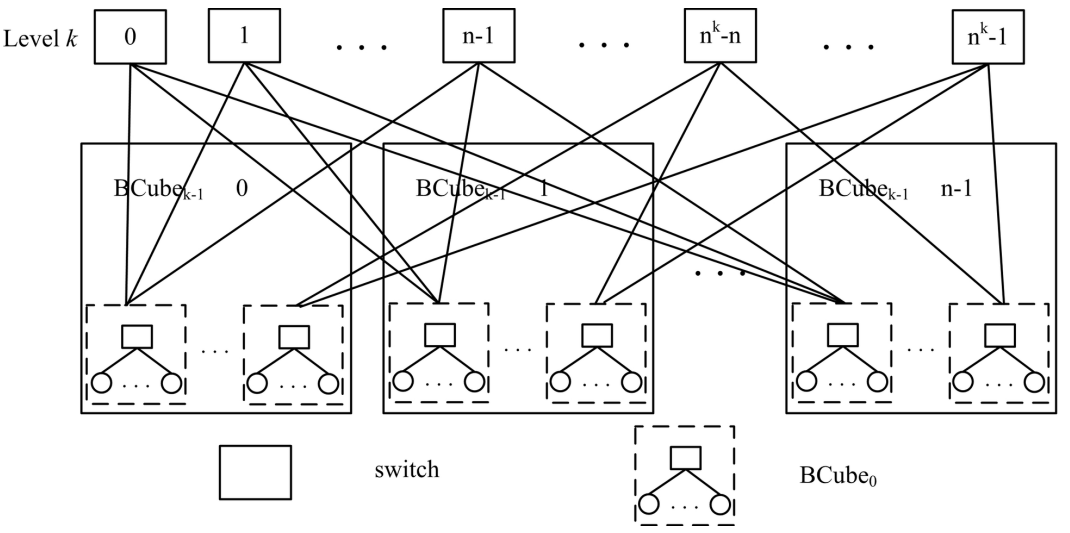
The network structure of BCube*k* is illustrated in Figure 2. The *n* BCube$k-1$s are numbered from 0 to (n-1) and the servers in each BCube$k-1$ are numbered from 0 to (n -1). In BCube_k, the *i*-th ($i \in$ [0, n-1]) port of the *j*-th ($j \in [0, n^k - 1]$) level-*k* switch connects to the level-*k* port of the *j*-th server in the *i*-th BCube$k-1$. For instance, the network structure of a BCube1 with $n = 4$ is shown in Figure 3, which is constructed with four BCube0s and four 4-port switches.

In BCube_k, the address of a server is denoted by an array $b_k b_{k|-|1\ldots0}$ ($b_i \in$ [0, n-1], $i \in$ [0, k]). Equivalently, the ID of a BCube server $\text{bsID} = \sum_{i=0}^{k} b_i n^i$ can also be used to denote a server. The address of a switch is denoted by $< l, s_{k|-|1} s_{k|-|2\ldots} s_0 >$ ($s_i \in$ [0, n-1], $i \in$ [0, k-1]), where l is the number of the switch level. According to the construction principle of BCube_k, we can achieve that the *i*-th ($i \in$ [0, n-1]) port of a switch $< l, s_{k|-|1} s_{k|-|2\ldots} s_0 >$ ($l \in$ [0, k]) connects to the level-l port of the server $s_{k|-|1\ldots} s_l i s_{l|-|1\ldots} s_0$.

2.3. Routing in BCube

The BCube routing is based on the hamming distance (Hamming, 1950) of the server addresses. The hamming distance of two servers is the number of different digits of their address arrays. Specially, when two servers in a BCube connect to the same switch, the two servers are neighbors and the hamming distance of them is one.

Figure 4. The connecting way of BCube_k



BCubeRouting provides single-path routing in BCube architecture to find a path from a source server to a destination server. Formally, the address of the source server is $S = s_k s_{k-1} \ldots s_0$ while the address of the destination server is $D = d_k d_{k|-|1} \ldots d_0$. BCubeRouting builds a path from S to D by continually "correcting" one digit of the current server each time until no different digits are remained. The digit correcting order is determined by a predefined permutation $\prod$, which is a permutation of the set {k, k-1, ?, 1, 0}. For example, the path between S(01) and D(10) in a BCube$_1$ is 01→11→10 with the permutation [1, 0]. The intermediate switches are not shown in the path, which can be determined by its two neighboring servers. It is easy to see that the diameter in BCube$_k$ is k +1. Since k is a small integer, BCube is a low-diameter network.

According to different permutations of the set {k, k-1, ?, 1, 0}, BCubeRouting can achieve different paths from the source server to the destination. Specifically, we have k + 1 parallel paths for any two servers in a BCubek. Here, two parallel paths between a pair of servers mean that the intermediate servers and switches on one path do not appear on the other. The permutation $\prod$ of each path in those k + 1 parallel paths is [i_j, (i_j - 1)mod(k + 1), ?, (i_j - k)mod(k + 1)] ($i_j \in [0, k]$), which starts from different locations of the address array and corrects the digits sequentially. For example, there are two parallel paths between S(01) and D(10) in a BCube1. One path of the parallel ones is 01→11→10 with $\prod$ = [1, 0] and the other is 01→00→10 with $\prod$ = [0, 1].

It is not difficult to see that if we build a partial BCube$_k$ with BCube$_{k-1}$s and use partial level-k switches to interconnect those BCube$_{k-1}$s, the corresponding partial BCube$_k$ will encounter the unreachable problem. That is to say, BCubeRouting does not work for some pairs of servers. Consider a simple example that we build a partial BCube$_1$ with n = 4 using two BCube$_0$s and the two switches <1, 0> and <1, 1> of level-1 switches for connecting those BCube$_0$s. It is not difficult to see that no matter which permutation is used, we can not find a path from the server with the address 02 to the server with the address 13 using BCubeRouting. The root cause of the unreachable problem is that some servers do not connect to any level-1 switch. To use the BCubeRouting in a partial BCube structure, the only effective way to build a partial BCube$_k$ is to first build the required BCube$_{k-1}$s and then use full level-k switches to connect those BCube$_{k-1}$s.

Based on the above analysis, a serious problem of a partial BCube is that the ports of switches are not fully utilized. Formally, when the number of BCube$_0$s in a partial BCube$_k$ is N ($k = \lceil \log_n N \rceil$), the port number of the switches is n. Correspondingly, the partial BCube$_k$ includes k + 1 level of switches, where the level-k switches have n^k switches and the other level-i switches have $m^* n^i$ ($m = \lceil N/n^i \rceil$) switches. In the partial BCube$_k$, the number of actually used switch ports are k + 1 times the number of servers. The port utilization of the switches in the partial BCube$_k$ is quantified as $U(N,n)$,

$$U\left(N,n\right) = \frac{N\left(log_n N + 1\right)}{n^{log_n N} + n^{\left(log_n N - 1\right)} \frac{N}{n^{\left(log_n N - 1\right)}} + \cdots + n^0 \frac{N}{n^0}} \tag{1}$$

According to Equation (1), we can achieve the switch port utilization of a partial BCube$_k$. As aforementioned, Figure 1 shows the port utilization of switches in a partial BCube$_k$ with the switch port n equaling eight and N varying from two to 512. Figure 1 illustrates that the utilization of switches can be quite low (e.g., 30.3% when the value of N is nine), resulting in a serious waste of network equipment investment.

$$N * n - 1 = \sum_{i=0}^{k} m_i n^i \tag{2}$$

In $BCube_k$, the number of network ports of a server is $k + 1$. As servers in current datacenters are equipped with a constant number of network ports to support a larger datacenter with the same value of k, the number of switch ports should be as large as possible. BCube can use the COTS switch to reduce the cost of a datacenter. In the current market, the switches up to 48/64 ports are at a fixed per-port price with the same trend extending to 100-150 ports (Popa et al., 2010). The BCube can let the value of n to be as large as possible to support a larger datacenter. For example, the BCube can support 4,096 servers with $n = 8$ and $k = 3$ while it can support 331,776 servers with $n = 24$ and $k = 3$. However, choosing a switch with a larger number of ports to support a larger scale datacenter results in lower utilization. For example, the partial $BCube_2$ consisting of 25 $BCube_0$s with 24-port switches has an extremely low port utilization of 11.6%, i.e., $U(25,24) = 11.6\%$, which brings a huge waste of equipment cost and space cost.

3. IBCUBE DESIGN

In this section, we present our IBCube structure, which supports the construction of datacenter with an arbitrary number of building blocks (i.e., $BCube_0$).

The IBCube uses the same devices as BCube architecture including servers with multiple ports and switches with a constant number of ports to connect servers. The expansion style of IBCube is to build the $BCube_0$s first and then connect the servers in those $BCube_0$s. With IBCube design, we can incrementally expand the existing datacenter with additional $BCube_0$s on demand. That is to say the increasing granularity of IBCube is a $BCube_0$ with n servers. For simplicity, we denote an IBCube constructed with N $BCube_0$ and n-port switches as IBCube (N, n). In IBCube (N, n), the address of a server can be represented as the array $b_k b_{k-1} \ldots b_0$ ($b_i \in [0, n - 1]$, $i \in [0, k]$). The value of k can be obtained from the expression $n^{k-1} < N <= n^k$, *i.e.*, $k = \lceil \log_n N \rceil$. Equivalently, an IBCube server is also denoted by the server ID $S = \sum_{i=0}^{k} b_i n^i$. Since there are at most $N * n$ servers in IBCube (N, n), the IBCube server ID S belongs to [0, $N * n$ - 1]. We can obtain the upper bound $m_k m_{k-1} \ldots m_0$ ($m_i \in [0, n\text{-}1]$, $i \in [0, k]$) of the server address $b_k b_{k|-|1} \ldots b_0$ from Equation (2),

To connect the servers in IBCube, we should decide the number of switch layers in IBCube and the number of switches in each switch layer first. Assuming that the number of network ports in each server is the same, it is not difficult to see that the number of switches in each layer should be N in IBCube (N, n); otherwise those switches will not be fully utilized. As the switches in IBCube are used to realize the connection among servers, the number of switch layers in an IBCube can be determined according to the function of each layer in the structure. To achieve a high network capacity, the connecting function of each switch layer in IBCube is in accordance with BCube. In the BCube structure, the level-0 switches realize the connection of servers in the same $BCube_0$. The level-k switches realize the connection of the servers in different $BCube_{k-1}$ but in the same $BCube_k$. More generally, level-l ($l \in [1, k]$) switches realize the connection of servers in different $BCube_{l|-|1}$ but in the same $BCube_l$.

Based on the above analysis, we define the connecting function of level-l ($l \in [1, k]$) switches in IBCube as: 1) connecting the servers in different $BCube_{l|-|1}$ but in the same $BCube_l$; 2) using the level-0 switches in each building block to connect the servers in the same $BCube_0$. To realize the connection among servers in IBCube, we should have corresponding level-l ($l \in [0, k]$) switches, as the absence of any layer may incur unreachable problem between some pairs of servers. Thus, the number of the switch layers needed in IBCube (N, n) is $k+1$ ($k = \lceil \log_n N \rceil$). Correspondingly, the number of ports needed by each server in IBCube (N, n) is $k + 1$. The switch address in IBCube is

the same as BCube, which is denoted as $< l, s_{k|-|1} s_{k|-|2\ldots} s_0 >$ ($l \in [1, k]$) and l indicates the level of the switch.

To realize the full utilization of the switches, we define a different connecting way between switches and servers for the IBCube structure. In Section 3.2, we further design a novel routing algorithm for the IBCube structure. We define the corresponding connecting way for IBCube to realize the connecting function of each switch layer. Specifically, we achieve the connecting way of IBCube by regarding BCube_0 as a vertex as the building block in IBCube, *i.e.*, the n servers in a BCube_0 is regarded as a unit. The address of a BCube_0 can be denoted as $b_k b_{k|-|1\ldots} b_1 *$, where * represents b_0 of all the servers' address in the BCube_0 and the value of * is actually in [0, n-1]. As the connection function of the level-0 switches in IBCube is realized, we further analyze the connecting way of the level-l ($l \in [1, k]$) switches in BCube. As BCube_0 is regarded as a vertex, a switch connecting to one server of a BCube_0 means the switch connecting to the BCube_0.

In BCube_k, those level-k switches numbered from zero to (n-1) realize the connection of the first BCube_0 in different $BCube_{k|-|1}$ through a complete bipartite graph, where every switch of the n switches is connected to every BCube_0 of the n BCube_0s. In the perspective of BCube_0, the level-k switches can be grouped according to their connecting BCube_0, where each switch group includes n level-k switches. The address of a level-k switch group connecting to the $b_k|-|_1 b_k|-|_2 \ldots b_1$*th BCube_0 of each $\text{BCube}_{k|-|_1}$ in the BCube_k, *i.e.*, the $ib_k|-|_1 b_k|-|_2 \ldots b_1$*th BCube_0 ($i \in [0, n\text{-}1]$), is denoted as $<k, b_k|-|_1 b_k|-|_2 \ldots b_1 *>$ (* = [0, n-1]). The connecting way of BCube_k which regards BCube_0 as a vertex is shown in Figure 4. More generally, the switch group $<l, b_k b_k|-|_1 \ldots b_l|+|_1 b_l|-|_1 \ldots b_1 *>$ ($l \in [1, k]$, * = [0, n-1]) connects to the $b_l|-|_1 \ldots b_1$*th BCube_0 in each $\text{BCube}_{l|-|_1}$ of the $b_k b_k|-|_1 \ldots b_{l+1}$ BCube_l, *i.e.*, the $b_k b_k|-|_1 \ldots b_l|+|_{+1} ib_l|-|_1 \ldots b_1$*th BCube_0 ($i \in [0, n\text{-}1]$).

From the above analysis, the BCube structure realizes the connection of BCube_0s by using a complete bipartite graph between the corresponding switch group and those BCube_0s to be connected.

Figure 5. The connection of level-2 switches in IBCube (6, 4)

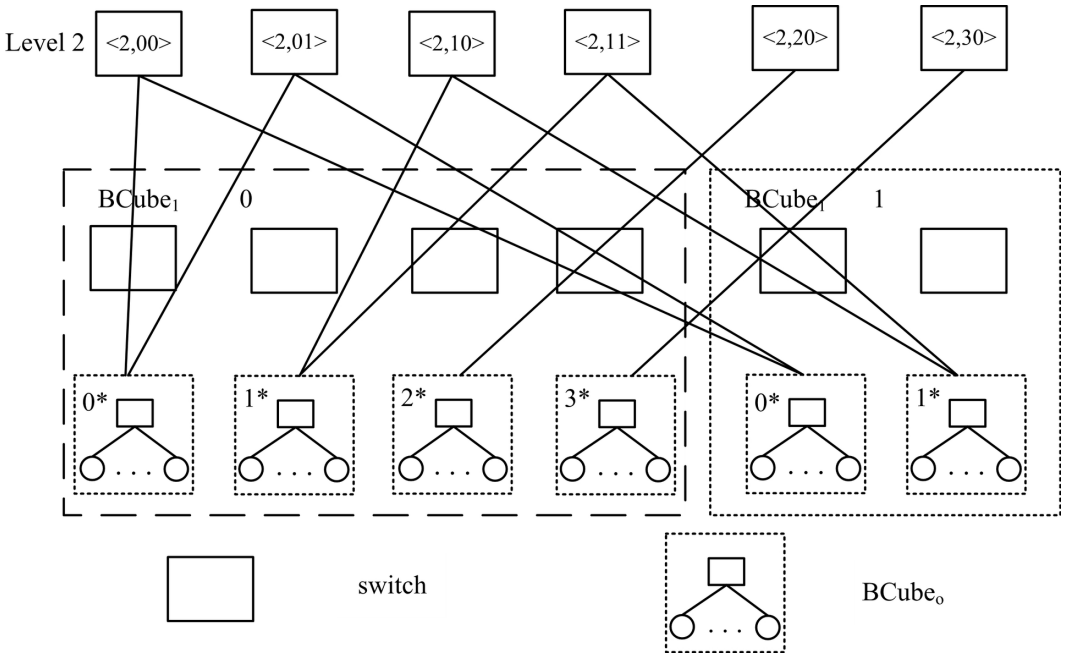


Figure 6. The connecting way of level-1 switches in IBCube (6, 4)

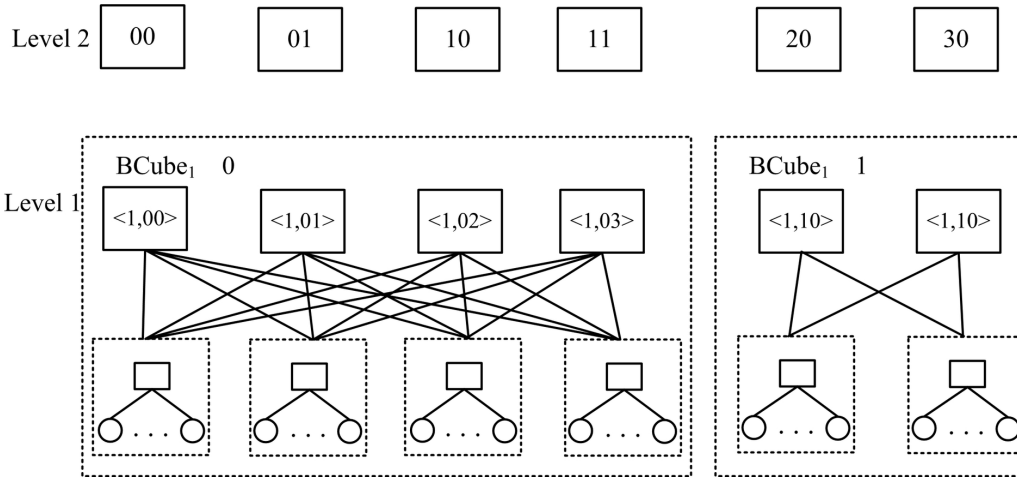
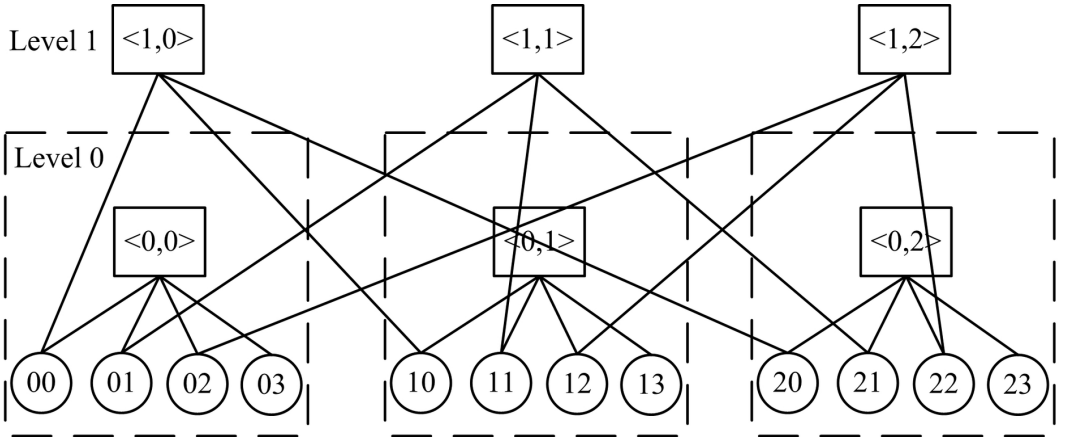


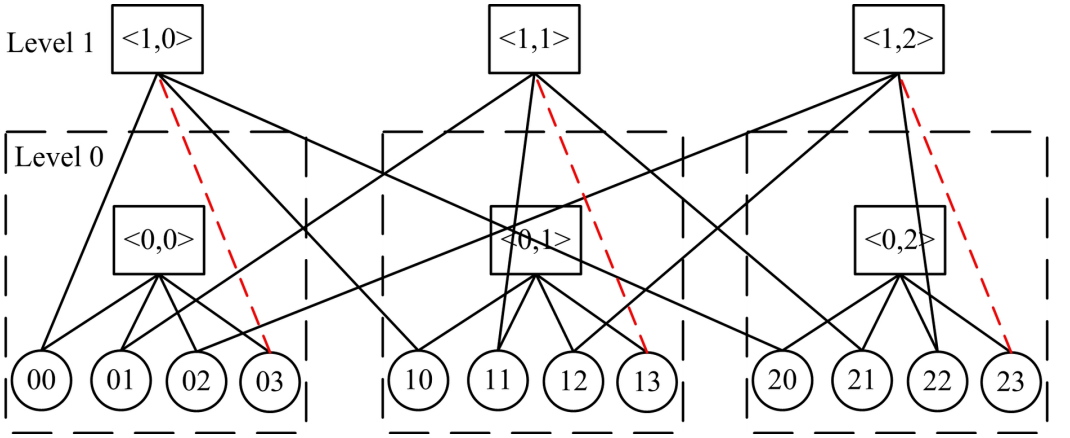
Figure 7. The connection of some servers in IBCube (3,4)



In the same way as BCube, IBCube realizes the connection between those $BCube_0$s in the same position of each $BCube_{k|-|1}$ using a level-k switch group with a complete bipartite graph. As the number of switches in level-k is the same as the number of $BCube_0$s in IBCube, the number of switches in a switch group should be the same as the number of its connecting $BCube_0$s. To obtain the switch number of the level-k switch group in IBCube (N, n) which realizes the connection between the $b_{k|-|1}b_{k|-|2}\ldots b_1$*th $BCube_0$ of each $BCube_{k|-|1}$, we only need to obtain the number of $BCube_{k|-|1}$ which has the $b_{k|-|1}b_{k|-|2}\ldots b_1$*th $BCube_0$ denoted as M (k, $b_{k|-|1}b_{k|-|2}\ldots b_1$*).

We can achieve the value of M (k, $b_{k|-|1}b_{k|-|2}\ldots b_1$*) in the IBCube ($N$, n) according to the network structure. From Equation (2), we can find that the IBCube (N, n) has m_k complete $BCube_{k|-|1}$s (numbered from zero to (m_k-1)) and the m_kth complete or incomplete $BCube_{k|-|1}$. Here, a complete or an incomplete $BCube_{k|-|1}$ in IBCube represents that all or a fraction of the servers of the $BCube_{k|-|1}$ are in the structure. We can obtain the number of servers in the m_kth complete/incomplete $BCube_{k-1}$ through Equation (2). Those m_k complete $BCube_{k|-|1}$s have the $b_{k|-|1}b_{k|-|2}\ldots b_1$*th $BCube_0$ without doubt. If the m_kth complete/incomplete $BCube_{k|-|1}$ has more than $b_{k|-|1}b_{k|-|2}\ldots b_1$* $BCube_0$, the m_kth

Figure 8. The structure of IBCube (3,4)



BCube$_k$|-|$_1$ will have the b_k|-|$_1b_k$|-|$_2$…b_1*th BCube$_0$. If not, there are only m_k BCube$_k$|-|$_1$s having the b_k|-|$_1b_k$|-|$_2$…b_1*th BCube$_0$. $M(k, b_k|\text{-}|_1b_k|\text{-}|_2\ldots b_1^*)$ can be calculated according to Equation (3),

$$M\left(k,b_{k-1}b_{k-2}\ldots b_1^*\right)=\begin{cases} m_k, & b_{k-1}b_{k-2}\ldots b_1 > m_{k-1}m_{k-2}\ldots m_1 \\ m_k+1, & b_{k-1}b_{k-2}\ldots b_1 \le m_{k-1}m_{k-2}\ldots m_1 \end{cases} \tag{3}$$

Correspondingly, the number of switches in the switch group used to connect the b_k|-|$_1b_k$|-|$_2$… b_1*th BCube$_0$ of each BCube$_k$|-|$_1$ in IBCube (N, n) is M (k, b_k|-|$_1b_k$|-|$_2$…b_1*) and the address of the switch group is <k, b_k|-|$_1b_k$|-|$_2$…b_1*> (* = [0, M (k, b_k|-|$_1b_k$|-|$_2$…b_1*) -1]). According to Equation (2), we can see that the number of BCube$_1$ which have the 3*th BCube$_0$ in an IBCube (6, 4) is one, *i.e.*, M (2, 3*) = 1. Thus, the switch number of the level-2 switch group used to connect the 3*th BCube$_0$ of each BCube$_1$ in the IBCube (6, 4) is only one. By using the complete bipartite graph, the connecting way of the level-2 switches in IBCube (6, 4) is shown in Figure 5.

More generally, we use M (l, b_kb_k|-|$_1$…b_l|-|$_1b_l$|-|$_1$…b_1*) (l∈ [1, k]) to denote the number of BCube$_l$|-|$_1$ which have the b_l|-|$_1$…b_1*th BCube$_0$ and in the b_kb_k|-|$_1$…b_l|+|$_1$th BCube$_l$. If the b_kb_k|-|$_1$… b_l|+|$_1$ BCube$_l$ is the last BCube$_l$ in IBCube (N, n), *i.e.*, b_kb_k|-|$_1$…b_l|+|$_1$ equals m_km_k|-|$_1$…m_l|+|$_1$, the BCube$_l$ has the first m_l complete BCube$_l$|-|$_1$s numbering from zero to (m_l-1) and the m_lth complete/ incomplete BCube$_l$|-|$_1$. Those first m_l BCube$_l$|-|$_1$s definitely have the b_l|-|$_1b_l$|-|$_2$…b_1*th BCube$_0$ without doubt. If the last m_lth BCube$_l$|-|$_1$ has more than b_l|-|$_1b_l$|-|$_2$…b_1* BCube$_0$, the m_lth BCube$_l$|-|$_1$ will have the b_l|-|$_1b_l$|-|$_2$…b_1* BCube$_0$. If not, the b_kb_k|-|$_1$…b_l|+|$_1$th BCube$_l$ only has m_l BCube$_l$|-|$_1$ which have the b_l|-|$_1b_l$|-|$_2$…b_1*th BCube$_0$. If b_kb_k|-|$_1$…b_l|+|$_1$ BCube$_l$ is not the last BCube$_l$ in IBCube (N, n), the b_kb_k|-|$_1$…b_l|+|$_1$ BCube$_l$ is a complete BCube$_l$. Thus, the b_kb_k|-|$_1$…b_l|+|$_1$ BCube$_l$ has n BCube$_l$|-|$_1$ which has the b_l|-|$_1$…b_1*th BCube$_0$. M (l, b_kb_k|-|$_1$…b_l|+|$_1b_l$|-|$_1$…b_1*) can be computed by Equation (4),

$$M\left(l,b_kb_{k-1}\ldots b_{l+1}b_{l-1}\ldots b_1^*\right)=$$

$$m_l,\quad b_k\ldots b_{l+1}=m_k\ldots m_{l+1}\ \&\ b_{l-1}\ldots b_1>m_{l-1}\ldots m_1$$

$$m_l+1,\quad b_k\ldots b_{l+1}=m_k\ldots m_{l+1}\ \&\ b_{l-1}\ldots b_1\le m_{l-1}\ldots m_1$$

$$n, \quad b_k \ldots b_{l+1} \neq m_k \ldots m_{l+1} \tag{4}$$

In IBCube (N, n), the number of switches in the switch group used to connect the $b_l|\text{-}|_1\ldots b_1$*th BCube_0 of each $\text{BCube}_l|\text{-}|_1$ in the $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1$ BCube_l is M (l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*) and the address of the switch group is <l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*> (* = [0, M (l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*)-1]). The connecting way of the level-1 switches using the complete bipartite graph in IBCube (6, 4) is shown in Figure 6.

In the following, we will try to obtain a connecting way between switches and servers in the IBCube structure to realize the complete bipartite graph between the switch group and its connected BCube_0s. Realizing a complete bipartite graph is possible because there are n servers in a BCube_0 and the switch number of a switch group that a BCube_0 connects to is less than or equal to n.

It is clear that the rewiring is inevitable when an existing datacenter needs to be expanded by adding additional BCube_0s and switches. As rewiring in a datacenter is costly, we should minimize the scale of the rewiring when a datacenter needs to be expanded. We have two goals in the connecting design between servers and switches in IBCube. The first goal is to realize the complete bipartite graph between switch group and its connected BCube_0s. The second goal is to minimize the rewiring scale.

From the above analysis, we can see that the BCube_0 $b_kb_k|\text{-}|_1\ldots b_1$* connecting to every switch of the switch group <l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*> (* = [0, M (l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*) - 1]). Here, a BCube_0 connecting to a switch means that at least one server in the BCube_0 connects to the switch where the number of the switch group <l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*> is equal to or less than n. We connect the level-l port of server $b_kb_k|\text{-}|_1\ldots b_1b_0$ ($b_0\in$ [0, M (l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*) - 1]) to the b_l port of the switch <l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1b_0$>. This not only realizes the connection between the switches and BCube_0s as we define above, but also minimizes the rewiring scale when an existing IBCube needs to be expanded. Since the switch number of a switch group (*i.e.*, M (l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*)) will not decrease when an existing datacenter expands to a larger one, those servers $b_kb_k|\text{-}|_1\ldots b_1b_0$ ($b_0\in$ [0, M (l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*)-1]) in the existing datacenter will connect to the same switch in the expanded datacenter. Figure 7 shows the connection of some servers in IBCube (3, 4).

Having defined the connecting way of the server $b_kb_k|\text{-}|_1\ldots b_1b_0$ ($b_0\in$ [0, M (l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*)-1]) of the BCube_0 $b_kb_k|\text{-}|_1\ldots b_1$*, we design the connecting way of the $b_kb_k|\text{-}|_1\ldots b_1b_0$ ($b_0\in$ [M (l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*), n]) server of the BCube_0. It is not difficult to see that the connecting way has a direct impact on the network capacity of a structure. To achieve a higher network capacity in IBCube, we analyze the connecting way of BCube which supports bandwidth hungry services in more details (Ports et al. 2015). In the BCube structure, a BCube_0 $b_kb_k|\text{-}|_1\ldots b_1$* connects to its corresponding switch group <l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*> through connecting the $b_kb_k|\text{-}|_1\ldots b_1b_0$ ($b_0\in$ [0, n-1]) server of the BCube_0 to switch <l, $b_kb|\text{-}|_1\ldots b_{l+1}b_l|\text{-}|_1\ldots b_1b_0$>, where the network traffics of the BCube_0 are distributed quite evenly among those switches of its corresponding switch group.

In IBCube, we also balance the traffic of a BCube_0 among all the switches of its connected switch group to achieve a higher network capacity. Specifically, we achieve the balanced traffic of a BCube_0 by letting the value of b_0 to determine the address of the switch that a server $b_kb_k|\text{-}|_1\ldots b_1b_0$ connects to. The switch group <l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*> connects to M (l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*) BCube_0s with the address $b_kb_k|\text{-}|_1\ldots b_1$* ($b_l\in$ [0, M (l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*)-1]), which only differs at the digit b_l. If the address of the switch that a server $b_kb_k|\text{-}|_1\ldots b_1b_0$ connects to is determined only by the value of b_0, there will be a number of M (l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*) servers with the same value of b_0 connect to the same switch of the switch group. However, there are n-M (l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*) remaining ports to be connected in each switch. Thus, we determine the address of the switch to which a server $b_kb_k|\text{-}|_1\ldots b_1b_0$ connects using the values of b_l and b_0. Specifically, the level-l port of a server $b_kb_k|\text{-}|_1\ldots b_0$ ($b_0\in$ [M (l, $b_kb_k|\text{-}|_1\ldots b_l|\text{+}|_1b_l|\text{-}|_1\ldots b_1$*), n-1]) connects to the j-th port of

a level-l switch <l, $b_k b_k|\text{-}|_1 \ldots b_l|\text{+}|_1 b_l|\text{-}|_1 \ldots b_1 s_0$>. The value of j is b_0 and the value of s_0 is calculated by Equation (5),

$$s_0 = (b_0 + b_l - n + 1)\%M\left(l, b_k b_{k-1} \ldots b_{l+1} b_{l-1} \ldots b_1 *\right) \tag{5}$$

Thus, those n-M (l, $b_k b_k|\text{-}|_1 \ldots b_l|\text{+}|_1 b_l|\text{-}|_1 \ldots b_1$*) servers in the M (l, $b_k b_k|\text{-}|_1 \ldots b_l|\text{+}|_1 b_l|\text{-}|_1 \ldots b_1$*) $b_k b_k|\text{-}|_1 \ldots b_1$* BCube$_0$ are fully connected to the n-M (l, $b_k b_k|\text{-}|_1 \ldots b_l|\text{+}|_1 b_l|\text{-}|_1 \ldots b_1$*) remaining ports of the switch group <l, $b_k b_k|\text{-}|_1 \ldots b_l|\text{+}|_1 b_l|\text{-}|_1 \ldots b_1$*> (* = [0, M (l, $b_k b_k|\text{-}|_1 \ldots b_l|\text{+}|_1 b_l|\text{-}|_1 \ldots b_1$*)-1]). According to the above analysis, we can see that the value of s_0 should be determined by the value of b_0 and b_l. In Equation (5), we use the value of (1-n) to determine the value of s_0 to reduce the rewiring scale. We compare the achieved rewiring scale with the case using the sum of b_0 and b_l to determine the value of s_0.

In summary, the level-l port of a server $b_k b_k|\text{-}|_1 \ldots b_0$ connects to the j-th port of a level-l switch <l, $b_k b_k|\text{-}|_1 \ldots b_l|\text{+}|_1 b_l|\text{-}|_1 \ldots b_1 s_0$>. The value of s_0 is calculated by Equation (6) and the value of the j port is computed by Equation (7),

$$s_0 = \begin{cases} b_0, & b_0 < M\left(l, b_k \ldots b_{l+1} b_{l-1} \ldots b_1 *\right) \\ (b_0 + b_l - n + 1)\%M\left(l, b_k b_{k-1} \ldots b_{l+1} b_{l-1} \ldots b_1 *\right), & \\ b_0, & b_0 \geq M\left(l, b_k \ldots b_{l+1} b_{l-1} \ldots b_1 *\right) \end{cases} \tag{6}$$

$$j = \begin{cases} b_l, & b_0 < M\left(l, b_k \ldots b_{l+1} b_{l-1} \ldots b_1 *\right) \\ b_0, & b_0 \geq M\left(l, b_k \ldots b_{l+1} b_{l-1} \ldots b_1 *\right) \end{cases} \tag{7}$$

In Figure 8 we obtain the structure of IBCube (3, 4) according to Equation (6) and Equation (7). The level-1 port of a server $b_1 b_0$ connects to the j-th port of a level-1 switch <1, s_0>. When b_0 is less than three, we have $s_0 = b_0$ and $j = b_1$. For example, 01 connects to the 0-th port of level-1 switch<1, 1>. When b_0 is not less than three, $s_0 = (b_0+b_1\text{-}3)\%3$ and $j = b_0$. For example, 03 connects to the third port of level-1 switch <1, 0>. The red dotted lines in the example in Figure 8 clearly show the IBCube's difference with that of BCube.

3.1. IBCube Routing

BCube uses the source routing to decide the path through which a packet from a source server flows by probing the network. The reasons for using the source routing are twofold. First, the source server controls the routing path without the need of coordination of intermediate servers. Second, the intermediate servers only need to forward the packets. As the connecting way of IBCube structure is obtained according to the BCube structure and the structure of IBCube including n^k BCube$_0$ is actually the same as BCube$_k$, the IBCube structure also uses the source routing protocol.

In the IBCube structure, two neighboring servers connecting to the same level-l switch may differ at the lth digit and the 0-th digit in their address arrays. Thus, we cannot apply the BCubeRouting directly to IBCube. We use the BFS algorithm on the source server to find a single path for the destination server. In IBCube, the way to achieve parallel paths is to remove the existing parallel paths from the IBCube structure and then use BFS to search for another. The newly found path is in parallel with the existing paths. When a new flow comes, the source sends the probing packets to all the parallel paths to obtain the required information of each path, *i.e.*, the available bandwidth

Table 1. Experiment setups

Parameters	Setups
# of switch levels	3 (k = 2)
# of nodes in BCube0 (n)	16
# of nodes in the structure	512 - 3,840
Running time	100s
Packet size	1,024bytes
Data rate for packet sending	1Mbps
Data rate for device channel	1,000Mbps
Communication pairs selection	Uniform and random selection
Traffic flow pattern	Exponential random traffic
Routing protocol	Nix-routing

of each path. In each intermediate server, the probing packets obtain the needed information. The destination returns a response packet for the probing packet to the source. When the source receives all the responses, it selects the best path according to the obtained information of each path.

In IBCube, to handle network failures, a source server periodically performs path selection to adapt to the network failures. When an intermediate server finds the next hop unreachable, it sends a path failure message to the source server. If there are still other available paths, the source server will switch the flow to another available path. When it is the time to explore the path, the source will perform a path selection and try to achieve all parallel paths.

3.2. Rewiring in IBCube

As aforementioned, the rewiring is required when the existing datacenter needs to expand. We can obtain the rewiring scale of IBCube according to its connecting way. When an existing network needs to expand by adding additional $BCube_0$s and switches, those cables to be rewired can be obtained by comparing the new structure with the original one.

For example, when we expand IBCube (3, 4) to IBCube (4, 4), we should rewire three cables and connect four new cables. The structure of IBCube (4, 4) is the same structure shown in Figure 3. By comparing the structure of IBCube (4, 4) in Figure 3 and the structure of the IBCube (3, 4) in Figure 8, it is not difficult to see that those cables represented using red dotted lines in Figure 8 should be reconnected to the switch <1, 3> and the four cables of the 3* $BCube_0$ should be connected to each switch of level-1 switches.

To cope with the link number scalability problem in BCube and accelerate the manual cabling when we build a datacenter or expand a datacenter, we can use the patch panel (Singla, 2012) so that we can plug cables from the switches into the panel in an easy-to-wire pattern. The total number of wires in the IBCube (N, n) is $k*N*n$, where N is a number in the set (n^{k-1}, n^k]. The total number of wires in the IBCube (N, n) is a value in the set ($k*n^k$, $k*n^{k-1}$]. When we expand an IBCube (N, n) ($N \in (n^{k-1}, n^k)$) by adding a $BCube_0$ and k switches to each switch level-i ($i \in [1, k]$), it is easy to see that the rewiring scale is less than $k*n^2$. This is because the rewiring only happens in the complete bipartite graphs, where the new $BCube_0$ will join in. When we expand an IBCube (N, n) ($N = n^k$) by adding a $BCube_0$, a layer - ($k + 1$) switch layer should be added to the new structure besides the k switches to be added to each switch layer - i ($i \in [1, k]$), that is to say, a number of $k + 1 + n^k$ switches should be connected to those servers in the new structure with no rewiring demand.

The detailed rewiring of expanding an IBCube (N, n) ($N \in (n^{k|-|1}, n^k)$) by adding a $BCube_0$ and k switch is as follows. The connecting way in the new IBCube can be obtained according to Equation (6) and Equation (7). In level-l switches, a new switch will be added to a switch group, which realizes the connection of those $BCube_0$s whose addresses are only different at the l-th digit with the new $BCube_0$. Correspondingly, the new $BCube_0$ will connect to those $BCube_0$s by the newly obtained switch group. It is easy to see that the connections of other switch groups in the new IBCube are the same as the original IBCube. The rewiring only happens in those switch groups where the new $BCube_0$ will join in.

Assuming the address of the new $BCube_0$ to be added to the original IBCube structure is $a_k a_k|-|_1 \ldots a_1 a_0$, the wires of those servers $a_k a_k|-|_1 \ldots a_l|+|_1 i a_l|-|_1 \ldots a_1 b_0$ with $b_0 = M\ (l, a_k a_k|+|_1 \ldots a_l|-|_1 a_l|-|_1 \ldots a_1{}^*)$ should reconnect to the new switch in level-l, and the wires of those servers with $b_0 > M\ (l, a_k a_k|-|_1 \ldots a_l|+|_1 a_l|-|_1 \ldots a_1{}^*)$ should reconnect to the switch with $s_0 = (b_0 + i - n + 1)\%(M\ (l, a_k a_k|-|_1 \ldots a_l|+|_1 a_l|-|_1 \ldots a_1{}^*) + 1)$. When the switch to be reconnected is the same as the connecting switch in the original IBCube, i.e., $(b_0 + i - n + 1)\%\ (M\ (l, a_k a_k|-|_1 \ldots a_l|+|_1 a_l|-|_1 \ldots a_1{}^*) + 1) = (b_0 + i - n + 1)\%M\ (l, a_k a_k|-|_1 \ldots a_l|+|_1 a_l|-|_1 \ldots a_1{}^*)$, the server $a_k a_k|-|_1 \ldots a_l|+|_1 i a_l|-|_1 \ldots a_1 b_0$ does not need to rewire its level-l wires. It is easy to see that those servers satisfying $(b_0 + i) \in [n\text{-}1, n\text{-}1 + M\ (l, a_k a_k|-|_1 \ldots a_l|+|_1 a_l|-|_1 \ldots a_1{}^*))$ do not need to rewire its level-l wires, where the value of $(b_0 + i)$ $(b_0 > M\ (l, a_k a_k|-|_1 \ldots a_l|+|_1 a_l|-|_1 \ldots a_1{}^*))$ is in $(M\ (l, b_k b_k|-|_1 \ldots b_l|+|_1 b_l|-|_1 \ldots b_1{}^*), n\text{-}1 + M\ (l, a_k a_k|-|_1 \ldots a_l|+|_1 a_l|-|_1 \ldots a_1{}^*)]$. This is the reason why we use $(1\text{-}n)$ to determine the value of s_0 to reduce the rewiring scale. When we only use the sum of b_0 and b_l to determine the value s_0, the level-l wires of those servers with $b_0 > M\ (l, a_k a_k|-|_1 \ldots a_l|+|_1 a_l|-|_1 \ldots a_1{}^*)$ will need to be rewired as the value of the $(b_0 + i)$ in those servers is larger than $M\ (l, a_k a_k|-|_1 \ldots a_l|+|_1 a_l|-|_1 \ldots a_1{}^*)$.

4. EXPERIMENT RESULTS

Since IBCube aims at reducing the expense on switches in the partial BCube, we compare the switch cost of a datacenter using IBCube with that of a datacenter using the partial BCube structure. We conduct comprehensive simulations based on ns-3 (Ns-3, n.d.), which is an open-source platform widely used by the research community for simulating the networks. In the evaluation, we compare the switch cost of the two structures using both 1Gbps switches and 10Gbps switches. After comparing the switch cost of IBCube with a partial BCube, we further examine the performance of IBCube structure.

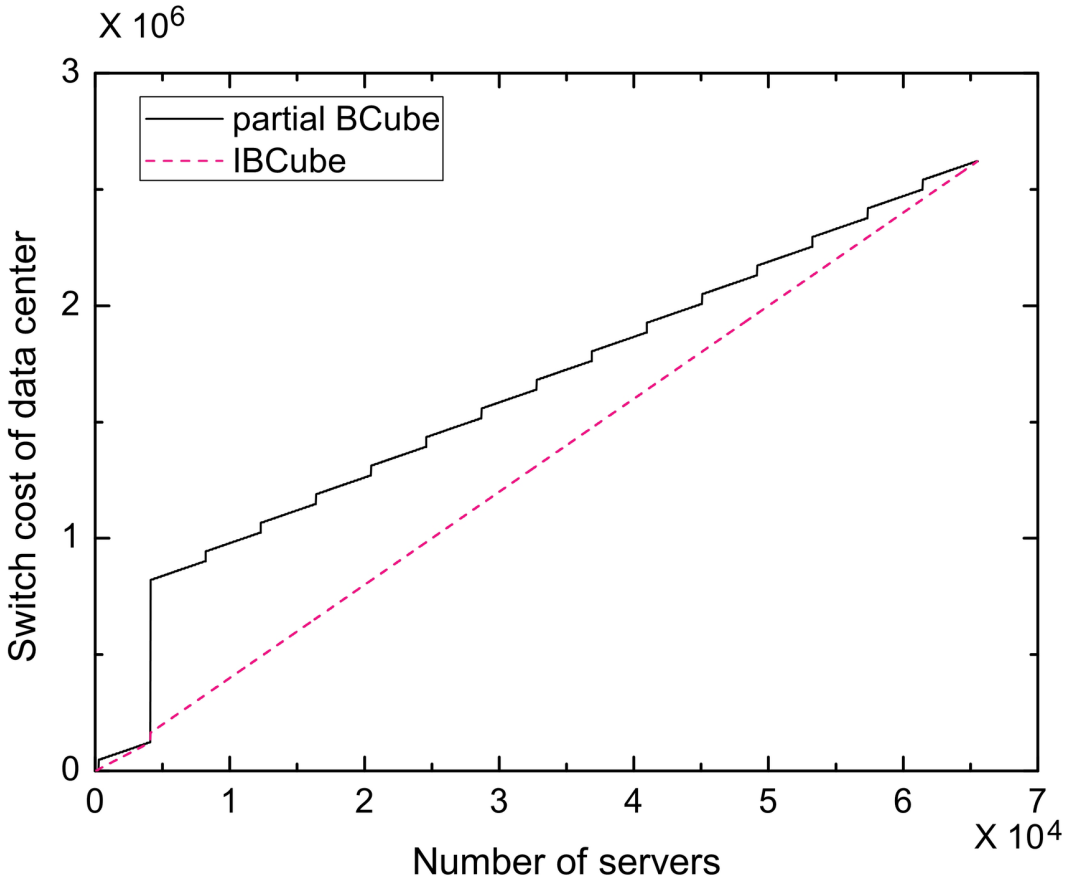
To compare the cost of our IBCube structure with that of the previous partial BCube, we examine a datacenter which could expand to 60,000 servers. As IBCube can support 65,536 servers with 16-port switches when the value of k is up to three, we use 16-port switches in each structure. The difficulty in the cost analysis is that the equipment prices vary greatly across vendors and products. As relatively aggressive prices, we use value of $10 per 1Gbps switch port (Chen et al. 2016).

In the following, we compare the switch cost of a datacenter which incrementally expands to 60,000 servers using the IBCube structure with that of the partial BCube. In the two structures, we expand the datacenter with one $BCube_0$ each time. Figure 9 compares the switch costs of the datacenters using different structures.

When a partial BCube with 4,112 servers needs to expand, the corresponding switch cost for expanding with the previous $BCube_0$ structure is $699,040. The switch cost for expanding the IBCube (256, 16) is only $41,600. The result shows that our IBCube design needs only $\frac{1}{17}$ of the cost of network equipments of the previous partial BCube design.

As using the 10Gbps switch is more and more common in datacenter (Popa et al. 2010), we also compare the switch cost of our IBCube design with that of the partial BCube both using 10Gbps switches. We set the value of price to $450 per 10Gbps switch port. We can see that the cost for expanding a previous partial BCube with 4,112 servers using a $BCube_0$ is $31,456,800, while the

Figure 9. The switch cost



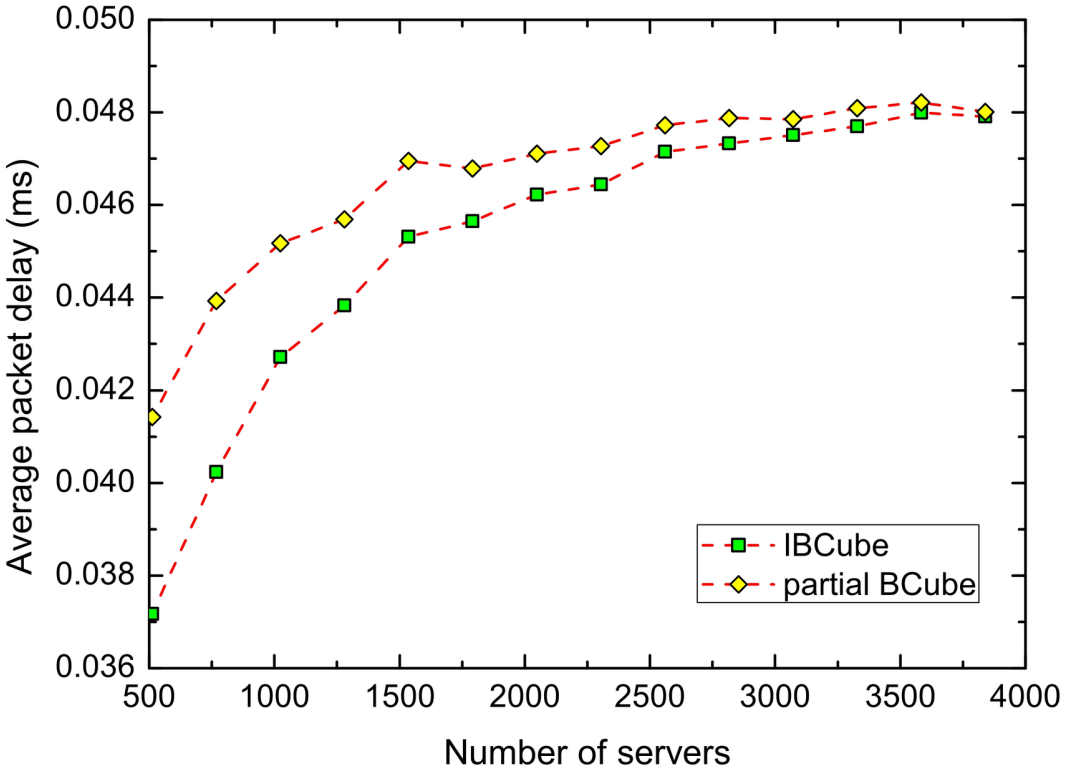
cost for expanding our IBCube (256, 16) using a $BCube_0$ is only \$1,872,000. The result also shows that our IBCube design decreases the cost of network by 94%.

For a fair comparison, in the evaluation, we use the same number of servers, the same configurations of network devices and examine the same traffic pattern in both structures. The main difference in the comparison is the network structure. We also use the BFS routing as the single-path routing in a partial BCube. Actually, the BFS routing and the BCubeRouting make no significant difference in a partial BCube. Table 1 summarizes the experiment setups used for the performance evaluation of our IBCube and the partial BCube structure in detail. The scale of both architectures ranges from 512 to 3,840 servers. The traffic flow pattern for each structure follows an on-off behavior with exponential random distribution, which is widely accepted as a reasonable model of traffic flows in real datacenters (Benson et al. 2010; Fu et al. 2015; Kong et al. 2017; Wang et al. 2016).

In the performance evaluation, we consider two performance metrics, the average packet delay and the average throughput (Chen et al. 2016). The average packet delay is calculated according to Equation (8),

$$D = \frac{1}{n}\sum_{i=1}^{n} d_i \tag{8}$$

Figure 10. Average packet delay



Here, the notation D represents the average packet delay; n denotes the total number of received packets; and d_i is the delay of packet i.

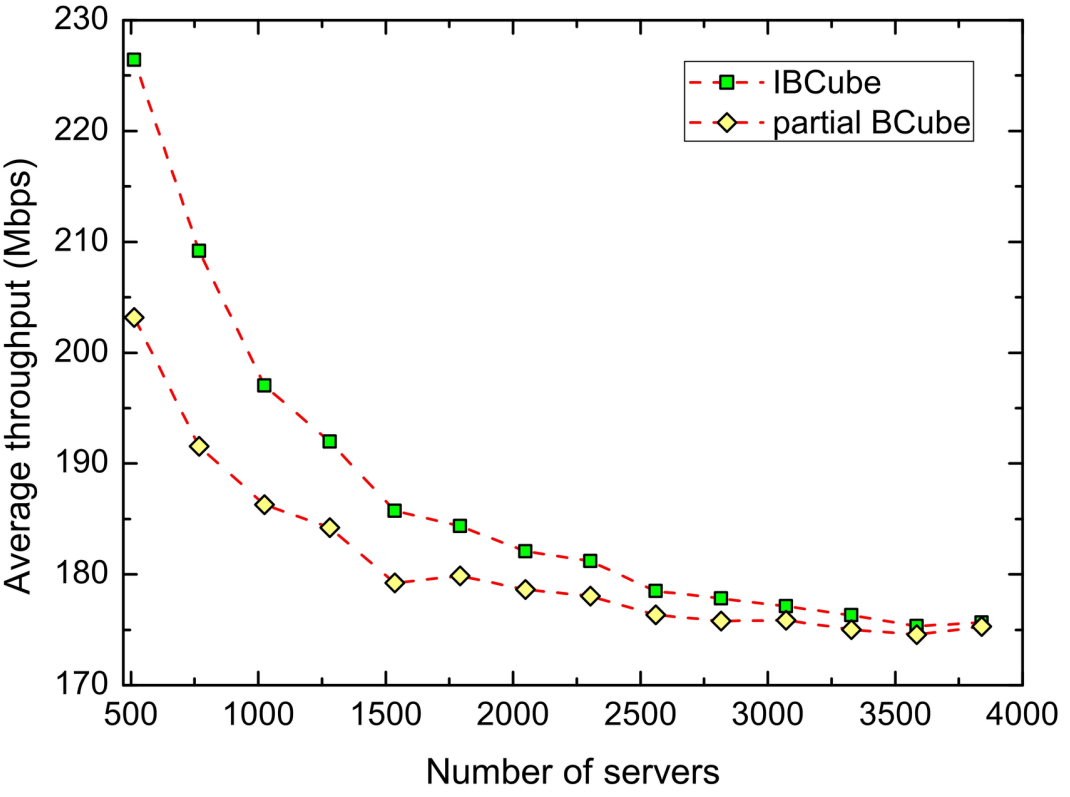
The average throughput is computed by Equation (9),

$$Tp = \frac{\sum_{i=1}^{n} p_i s_i}{\sum_{i=1}^{n} d_i} \tag{9}$$

where the notation T_p represents the average throughput. The value of p_i is zero if packet i is lost; otherwise, the value of p_i is one. The notation s_i represents the size of packet i; n is the total number of packets, and d_i is the delay of packet i.

We plot the average packet delay in Figure 10 and the average throughput in Figure 11. As IBCube uses fewer switches to connect the same number of servers as the partial BCube, this will increase the traffic load of each switch in IBCube. However, we can see that the performance of our IBCube design is much better than that of the previous partial BCube according to Figure 10 and Figure 11. The results in Figure 10 show that IBCube reduces the average packet delay by 10.3% compared to the partial BCube design. This is because there is a shorter average hop in IBCube due to the smaller number of switches. For example, the server 01 and the server 13 are neighbors in IBCube (3,4). However, the hamming distance of the server 01 and the server 13 is two in a partial $BCube_1$ with three $BCube_0$s. Figure 11 shows that IBCube outperforms previous partial BCube by 11.5% in terms of throughput.

Figure 11. Average throughput



5. CONCLUSION AND FUTURE WORK

In this work, we propose IBCube, a novel design for economically constructing a datacenter network. In summary, we have achieved the contribution in threefold. First, we propose an economically structure which can interconnect an arbitrary number of servers. Specifically, we design a novel automatic port allocation algorithm which fully utilizes the switch ports to achieve a minimized number of required switches for datacenters with an arbitrary number of servers. Second, based on the IBCube datacenter network architecture, we design a routing algorithm. Third, we examine the efficiency and performance of the IBCube design using comprehensive simulations based on real world system configurations. The results show that the IBCube design significantly reduces the expenses on network equipments by 94% as well as achieves better latency and throughput compared to the previous partial BCube design.

ACKNOWLEDGMENT

This research is supported in part by the National Key Research and Development Program of China under grant No.2016QY02D0302, NSFC under grants Nos. 61422202, 61370233, 61433019, Foundation for the Author of National Excellent Doctoral Dissertation of PR China under grant No.201345, and Research Fund of Guangdong Province under grant No.2015B010131001.

REFERENCES

Abu-Libdeh, H., Costa, P., Rowstron, A., O'Shea, G., & Donnelly, A. (2010). Symbiotic routing in future data centers. In *SIGCOMM* (pp. 51-62). ACM Press. doi:10.1145/1851182.1851191

Al-Fares, M., Loukissas, A., & Vahdat, A. (2008). A scalable, commodity data center network architecture. In *SIGCOMM* (pp. 63-74). ACM Press. doi:10.1145/1402958.1402967

Benson, T., Anand, A., Akella, A., & Zhang, M. (2010). Understanding data center traffic characteristics. *Computer Communication Review*, *40*(1), 92–99. doi:10.1145/1672308.1672325

Chen, Y., Hao, C., Wu, W., & Wu, E. (2016). Robust dense reconstruction by range merging based on confidence estimation. *Science China. Information Sciences*, *59*(9). doi:10.1007/s11432-015-0957-4

Chen, Y., Hao, C., Wu, W., & Wu, E. (2016). Robust dense reconstruction by range merging based on confidence estimation. *Science China. Information Sciences*, *59*(9), 092103. doi:10.1007/s11432-015-0957-4

Datacenterknowledge. (2015). Facebook now has 30000 servers. Retrieved from http://www.datacenterknowledge.com/archives/2009/10/13/facebooknow-has-30000-servers/

Fu, Z., Sun, X., Liu, Q., Zhou, L., & Shu, J. (2015). Achieving efficient cloud search services: Multi-keyword ranked search over encrypted cloud data supporting parallel computing. *IEICE Transactions on Communications*, *98*(1), 190–200. doi:10.1587/transcom.E98.B.190

Greenberg, A., Hamilton, J., Maltz, D. A., & Patel, P. (2008). The cost of a cloud: Research problems in data center networks. *Computer Communication Review*, *39*(1), 68–73. doi:10.1145/1496091.1496103

Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., . . . Lu, S. (2009). BCube: a high performance, server-centric network architecture for modular data centers. In *SIGCOMM* (pp. 63-74). ACM Press. doi:10.1145/1592568.1592577

Guo, C., Wu, H., Tan, K., Shi, L., Zhang, Y., & Lu, S. (2008). Dcell: a scalable and fault-tolerant network structure for data centers. In *SIGCOMM* (pp. 75-86). ACM Press. doi:10.1145/1402958.1402968

Gyarmati, L., Gulyás, A., Sonkoly, B., Trinh, T. A., & Biczók, G. (2013). Free-scaling your data center. *Computer Networks*, *57*(8), 1758–1773. doi:10.1016/j.comnet.2013.03.005

Hamming, R. W. (1950). Error detecting and error correcting codes. *Bell Labs Technical Journal*, *29*(2), 147–160. doi:10.1002/j.1538-7305.1950.tb00463.x

Kong, Y., Zhang, M., & Ye, D. (2017). A belief propagation-based method for task allocation in open and dynamic cloud environments. *Knowledge-Based Systems*, *115*, 123–132. doi:10.1016/j.knosys.2016.10.016

Lee, K. S., Wang, H., Shrivastav, V., & Weatherspoon, H. (2016). Globally synchronized time via datacenter networks. In *SIGCOMM* (pp. 454-467). ACM Press. doi:10.1145/2934872.2934885

Liu, Q., Cai, W., Shen, J., Fu, Z., Liu, X., & Linge, N. (2016). A speculative approach to spatial-temporal efficiency with multi-objective optimization in a heterogeneous cloud environment. *Security and Communication Networks*, *9*(17), 4002–4012. doi:10.1002/sec.1582

Ns-3. (2016). Retrieved from http://www.nsnam.org

Perry, J., Ousterhout, A., Balakrishnan, H., Shah, D., & Fugal, H. (2014). Fastpass: A centralized zero-queue datacenter network. In *SIGCOMM* (pp. 307-378). ACM Press. doi:10.1145/2619239.2626309

Popa, L., Ratnasamy, S., Iannaccone, G., Krishnamurthy, A., & Stoica, I. (2010). A cost comparison of datacenter network architectures. In CoNEXT (p. 16). ACM Press. doi:10.1145/1921168.1921189

Ports, D. R., Li, J., Liu, V., Sharma, N. K., & Krishnamurthy, A. (2015). Designing Distributed Systems Using Approximate Synchrony in Data Center Networks. In *NSDI* (pp. 43-57). ACM Press.

SGI-Products. (2016) Retrieved from http://sgi.com/products/data_center/ice_cube_air

Singh, A., Ong, J., Agarwal, A., Anderson, G., Armistead, A., Bannon, R., . . . Kanagala, A. (2015). Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network. In *SIGCOMM* (pp. 183-197). ACM Press. doi:10.1145/2785956.2787508

Singla, A., Hong, C. Y., Popa, L., & Godfrey, P. B. (2012). Jellyfish: Networking Data Centers, Randomly. In *NSDI* (pp. 17-17). ACM Press.

Wang, G., Zhang, S., Wu, K., Zhang, Q., & Ni, L. M. (2016). TiM: Fine-grained rate adaptation in WLANs. *IEEE Transactions on Mobile Computing*, *15*(3), 748–761. doi:10.1109/TMC.2015.2421938

Zhan, Y., Xu, D., & Yu, H. (2016). Pricing the spare bandwidth: Towards maximizing data center's profit. *Science China. Information Sciences*, *59*(10), 102303. doi:10.1007/s11432-016-0355-0

Zhu, Y., Kang, N., Cao, J., Greenberg, A., Lu, G., Mahajan, R., . . . Zheng, H. (2015). Packet-level telemetry in large datacenter networks. In *SIGCOMM* (pp. 479-491). London: ACM Press. doi:10.1145/2785956.2787483

Qiong Hu received the master degree from School of Computer Science and Technology, Huazhong University of Science and Technology in 2014. Her research interest includes data center networks.

Hanhua Chen received the PhD degree in computer science and engineering from the Huazhong University of Science and Technology, China, in 2010, where he is currently a professor at the School of Computer Science and Technology. His research interests include data center networks, distributed systems, and big data processing. He received the National Excellent Doctoral Dissertation Award of China in 2012. He is a member of the IEEE.

Hai Jin received the PhD degree in computer engineering from the Huazhong University of Science and Technology (HUST), China, in 1994. He is a Cheung Kung Scholars chair professor of computer science and engineering at HUST. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. He worked at the University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He received Excellent Youth Award from the National Science Foundation of China in 2001. He is the chief scientist of ChinaGrid, the largest grid computing project in China, and the chief scientist of National 973 Basic Research Program Project of Virtualization Technology of Computing System. He has coauthored 15 books and published more than 600 research papers. His research interests include computer architecture, virtualization technology, cluster computing and grid computing, peer-to-peer computing, network storage, and network security. He is named the steering committee chair of several International Conferences including GPC, APSCC, FCST, and ChinaGrid and is a member of the steering committee of CCGrid, NPC, GCC, ATC, and UIC. He is a senior member of the IEEE and a member of the ACM.

Chen Tian is an associate professor at State Key Laboratory for Novel Software Technology, Nanjing University, China. He was previously an associate professor at School of Electronics Information and Communications, Huazhong University of Science and Technology, China. Dr. Tian received the BS (2000), MS (2003) and PhD (2008) degrees at Department of Electronics and Information Engineering from Huazhong University of Science and Technology, China. From 2012 to 2013, he was a postdoctoral researcher with the Department of Computer Science, Yale University. His research interests include data center networks, network function virtualization, distributed systems, Internet streaming and urban computing.

Aobing Sun received his PhD from the College of Science and Technology of Huazhong University of Science and Technology in 2008. Currently, he is a research associate of the Cloud Computing Centre of Chinese Academy of Sciences. And he is also the CTO of GCloud Science and Technology Corp that aims to accelerate the technology application of cloud computing and big data. His research areas include cloud computing, big data, computer architecture and image processing.

Tongkai Ji received his PhD from the Remote-Sensing Technology Institute of Chinese Academy of Sciences. His research areas include remote-sensing, cloud computing and big data. Currently, he is the Director of the Cloud Computing Centre of Chinese Academy of Sciences.