# *PCube:* Improving Power Efficiency in Data Center Networks

Lei Huang, Qin Jia, Xin Wang
Fudan University
Shanghai, China
08300240053@fudan.edu.cn
08300240080@fudan.edu.cn
xinw@fudan.edu.cn

Shuang Yang
Stanford University
California, USA
shyang@stanford.edu

Baochun Li
University of Toronto
Toronto, Canada
bli@eecg.toronto.edu

***Abstract*—To alleviate the growing concern of energy waste in networked devices, we present *PCube*, a server-centric data center structure that conserves energy by varying bandwidth availability based on traffic demand. *PCube* not only supports a low-power mode for existing data centers offering full bisection bandwidth without hardware modification or re-wiring, but also provides an alternative for new data centers at a lower construction cost. The bandwidth demand could be dynamic and scalable in the optimal way when considering conserving the energy waste. To further reduce the cost, we take advantage of traffic locality, and propose *Hybrid PCube*, in order to offer different bandwidth to different servers. We use samples of traffic from a real-world production data center with full bisection bandwidth across thousands of servers, and evaluate our proposed algorithm using these collected samples. Our experimental results have shown convincing evidence that the proposed algorithm is able to substantially reduce energy costs incurred by networked devices in data centers.**

***Index Terms*—Energy Efficiency; Data Center Network; Locality Architecture;**

## I. Introduction

Energy use is a central issue for data centers. Power draw for data centers ranges from a few kW for a rack of servers in a closet to several tens of MW for large facilities [1]. For higher power density facilities, electricity cost is dominant operating expense and account for over 10% of the total cost of ownership (TCO) of a data center [2]. Data centers alone contribute significantly to the energy consumption problem: the US Environmental Protection Agency estimates that data centers consume about 61 billion kilowatt-hours of electricity in 2006, which costs $4.5 billion and accounts for 1.5% of the total bill for electricity used in the entire United States [3]. Due to the need of interconnecting servers in data centers, a large number of network switches are used to form network topologies within a data center. Network switches themselves, however, are computing devices and their energy consumption should not be ignored.

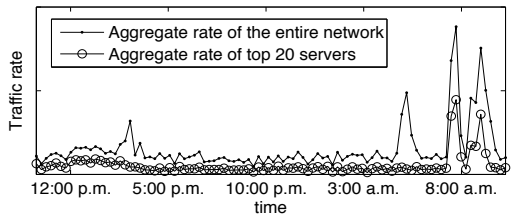
[1]This work is supported in part by 863 program of China under Grant No. 2009AA01A348, National Key S&T Project under Grant 2010ZX03003-003-03, Shanghai Municipal R&D Foundation under Grant No. 09511501200 and FDUROP (Fudans Undergraduate Research Opportunities Program). Xin Wang is the corresponding author.

The good news is that there exist abundant opportunities to optimize energy consumption of network switches in data centers. Recently proposed data center structures, such as BCube [4], fat-tree [5] and VL2 [6], offer full bisection bandwidth. With such network structures, observations in previous work have shown that fewer than 10% links are more than 95% utilized, and more than 80% of these links have very light traffic [7].

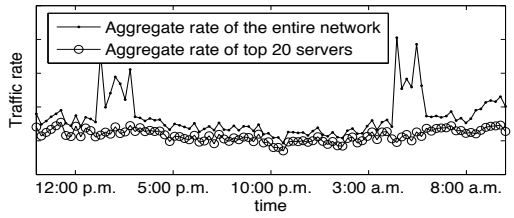
By budgeting power output for different Ethernet cable lengths, and by automatically powering down ports with no links, existing works have already taken initial steps towards conserving the energy used by switches. Gupta *et al.* have proposed several line cards sleeping algorithms, taking advantage of intermittent traffic patterns [8], [9]. Green switches as products have already appeared in the market [10]. However, most of such green switches are designed to reduce power on the line cards only. Recent experiments have identified that an idle switch still consumes about 70% of the electricity costs [11].

Fig. 1 shows samples that we collected on an operational data center in a 24-hour period, performing MapReduce for machine learning on July 20, 2010. It is clear that traffic volumes vary significantly over the 24-hour period, but remain far lower than the daily peak most of the time. It is therefore conceivable that, if new algorithms and structures could be designed to allow most of the switches in a network structure to remain dormant most of the time, energy consumption levels may adapt to traffic volumes in a rhythmic fashion over time.

In this paper, we present *PCube*, an elastic data center structure that is designed to optimize power efficiency. *PCube* allows energy consumption in a network structure to scale according to traffic demand patterns, yet still maintains full bisection bandwidth, enjoyed by recent data center network structures in the literature. The highlight in the *PCube* design is to power off some of the switches in order to support various network bandwidth demand and conserve energy. *PCube* can be directly applied to existing hypercube structured data centers, *e.g.,* BCube [4] and MDCube [12], without any hardware modification or re-wiring. After powering off a subset of switches, the residual network structure in *PCube* is still able to

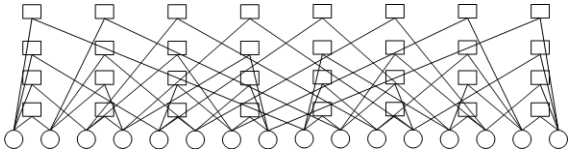



(a) Machine learning service using MapReduce.

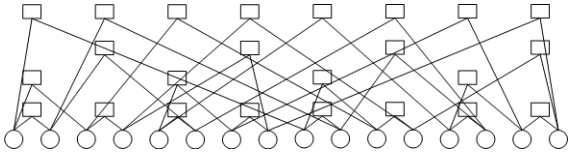



(b) Storage service.

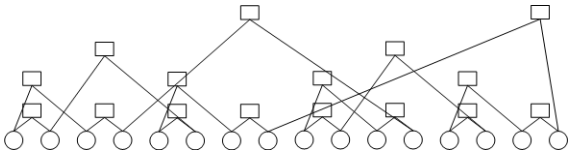
Fig. 1. Aggregate throughput collected from an operational data center with thousands of servers on July 20, 2010.



(a) BCube$_3$(2) or PCube(2, 4, 4)



(b) An example of PCube(2, 4, 3), uses 25% less switches and NIC power, and possesses 3 edge-disjoint paths for any two servers.



(c) An example of PCube(2, 4, 2), uses 50% less switches and NIC power, and possesses 2 edge-disjoint paths for any two servers.

Fig. 2. Examples of BCube and PCube with 16 servers.

maintain favorable properties of hypercube structures, such as fault tolerance and support for broadcast traffic. *PCube* can, of course, be also deployed when a new data center is constructed.

A highlight of PCube is its flexibility: PCube can be adaptive to traffic patterns over time. When traffic is light, the number of edge-disjoint spanning trees in PCube, $q$, could be small. For example, in Fig. 2(b), a PCube$(2, 4, 3)$ accommodates the same number of servers as PCube$(2, 4, 4)$, but powers off $25\%$ of all switches, offering a speedup of 3 for broadcast. Therefore, PCube$(2, 4, 3)$ saves 25% of the energy used by switches, as compared with PCube$(2, 4, 4)$. PCube$(2, 4, 3)$ could turn to PCube$(2, 4, 2)$ by powering 8 more switches off, thus saving 50% of the switch power while offering a speedup of 2 for broadcast, as shown in Fig. 2(c). The details of the PCube's parameters will be presented in Sec. III.

To take *PCube* a step further, we take traffic locality into consideration. In Fig. 1(b), 20 most heavily loaded servers account for about 80% of the total traffic load in the data center with thousands of servers. We propose *Hybrid PCube* so that differentiated bandwidth availability may be allocated to different servers. The simulation which establishes *Hybrid PCube* can conserve 20% to 50% electricity cost by network devices.

The remaining of the paper is organized as follows. Sec. II compares *PCube* and recently proposed data center network architectures. Sec. III describes the structure of *PCube*, including its construction and graph properties. In Sec. IV, we introduce *Hybrid PCube* to provision bandwidth with locality. The configuration and routing of *PCube* is introduced in Sec. V, and its energy efficiency is evaluated in Sec. VI. Finally, Sec. VII concludes the paper.

## II. Related Work

In recent literature, there are two main streams of data center network structure designs: server-centric and switch-centric. DCell [13] and BCube [4] are two representative server-centric data center structures. In server-centric data center structures, most routing tasks are performed at the servers. In this context, we focus on BCube, since it offers higher bandwidth with more potential for conserving energy.

BCube is a new network architecture designed for shipping container based, modular data center. The servers in BCube$_k(n)$ are organized in hypercube $n^k$. The BCube architecture exhibits high performance in one-to-one, one-to-all and all-to-all traffic cases. Fig. 2(a) illustrates BCube$_3(2)$.

Comparing with BCube, PCube, proposed in this paper, offers different levels of bandwidth. A PCube$(n, k, q)$ structure deploys $n^k$ servers with $q$ edge-disjoint spanning trees to provide bandwidth. Fig. 2(a) is also a special case of PCube — PCube$(2, 4, 4)$. BCube$_{k-1}(N)$ has $N^k$ $k$-port servers connected by $kN^{k-1}$ $N$-port mini-switches. BCube$_{k-1}(N)$ has $k$ edge-disjoint spanning trees, thus providing one-to-all traffic at a speedup of $k$.

Moreover, as BCube$_{k-1}$ is expanded to BCube$_k$ to serve more servers, all servers have to add one additional network interface (NIC), and the speedup of one-to-all traffic has to increase by one, which is not necessary in PCube. As a result, PCube could accommodate more servers using fewer switches, NICs and less power. For example, only 8 servers with 3 NICs can be connected by the 2-port mini-switches in BCube, but in Fig. 2(b), 16 servers are connected.

Fat-tree [5] and VL2 [6] are typical switch-centric data center structures. Based on fat-tree, ElasticTree [11] was proposed to

conserve energy by managing network-wide traffic. Although both PCube and ElasticTree try to design solutions to providing bandwidth and power scalability in data centers, they belong to two different categories, as PCube is server-centric and ElasticTree is switch-centric. A comparison between ElasticTree and PCube is very much like that between fat-tree and BCube [4]: PCube (BCube) provides better one-to-all traffic support than ElasticTree (fat-tree); besides, PCube is more feasible for deployment since servers are more programmable than switches. With respect to energy saving, the maximum energy consumption in network devices in PCube is about half of that in ElasticTree with the same number of servers [4].

Gyarmati László and Trinh Tuan Anh discussed the impact of data center architecture on reducing energy consumption [14]. They offer one solution by designing data center with more parameters [14]. Therefore, the data center could adapt to more application environment with more precise energy consumption. PCube offers more parameter than other architectures such as BCube and DCell and we will see this parameter of bandwidth will contribute to the conservation of energy.

The energy aware routing in the data centers [15] try to use less network switches to provide services. The energy aware routing method analyzes the bandwidth demand in a static environment and uses heuristic routing algorithm to reduce the number of switches in the active network. Comparing with applying energy aware routing in BCUbe, PCube provides a more simple and efficient path-finding. PCube can be dynamically reconfigured and be able to reduce more switches because of the graceful network properties of PCube discussed later.

## III. PCube Structures

PCube is designed to be able to dynamically adjust its network structure according to traffic volumes. When the traffic demand is low, we turn off a subset of switches for power saving; when traffic demand rises, we power on more switches to increase the network capacity. The adjustment is performed in a way that there are always multiple parallel paths between any two servers. PCube therefore provides high network capacity and is power efficient at the same time. Of course, even when dynamic network structure is not preferred, PCube can still be used for cost saving when full bisection bandwidth is not needed. To describe the PCube structure precisely, we start by introducing some notations in PCube.

### A. Identifying Servers and Switches

There are two types of devices in PCube network: servers and switches. A server has $q$ network ports ($q \geq 2$) and a switch has $n$ ports ($n \geq 2$). We use PCube$(n, k, q)$ to denote a PCube network, where $k$ is the number of levels of the switches ($k \geq q$).

Servers are organized as a hypercube of $n^k$, and switches connect the servers through the hypercube. Thus, each server and switch can be identified by a unique id of $k$ dimensions. Every server is assigned a server id $a_k a_{k-1} \dots a_1$, where $a_i (i \in [1, k]) \in [0, n-1]$. Every switch is assigned a switch id $b_k b_{k-1} \dots b_{j+1} x b_{j-1} \dots b_1$, where $b_i (i \in [1, k], i \neq j) \in [0, n-1]$. $x$ is just a place holder, and the position of $x$ (in this case is $j$) denotes which layer (or dimension) the switch locates.

A switch with id $b_k \dots b_{j+1} x b_{j-1} \dots b_1$ connects to $n$ servers, whose identifiers are only different at the $j^{\text{th}}$ dimension, in the form of $b_k \dots b_{j+1} d_j b_{j-1} \dots b_1$, where $d_j$ can be any number in $[0, n-1]$. For example, in PCube$(4, 4, 4)$, switch $x111$ connects to four servers with id 0111, 1111, 2111, 3111, respectively.

Fig. 3 gives a PCube$(2, 4, 3)$ network, from which we see each server uses 3 ports, and each switch has 2 ports. The server identifiers are $0000, 0001, \dots, 1110, 1111$. The first layer of switches are identified as $000x, 001x, \dots, 111x$. the second layer of switches are identified as $00x0, 00x1, \dots, 11x1$, etc.

Next, we introduce how PCube is constructed in detail.

### B. Constructing a PCube Structure

There are two guiding principles as we design PCube. First, PCube can be obtained by powering off some network switches in BCube, with the same set of servers. Second, PCube maintains $q$ parallel paths between all pairs of servers, yet with the use of the smallest number of switches and NICs.

Based on these principles, we recursively construct PCube$(n, k, q)$ from PCube$(n, k-1, q-1)$, where $k \geq q > 2$. We will first introduce the construction of the special case PCube$(n, k, 2)$, henceforth referred to as the *primary PCube*, as the base case in our recursive construction.

In the primary PCube$(n, k, 2)$, the switches in the $i^{\text{th}}$ dimension are as follows:

- $b_k b_{k-1} \dots b_2 x$ for $i = 1$;
- $b_k b_{k-1} \dots b_{i+1} x e_{i-1} o_{i-2} \dots o_1$ for $i \in [2, k-1]$;
- $x b_{k-1} o_{k-2} o_{k-3} \dots o_1$ for $i = k$.

Where $e_j$ is an *even number* in $[0, n-1]$ and $o_j$ is an *odd number* in $[0, n-1]$. Indeed, the odd number and even number are just one of the ways to split the numbers into two sets. Any other method of spliting could work in PCube. In addition, we choose the odd and even number for load balance and graceful expressions.

An example of primary PCube switch identifiers can be found in Fig. 3, where 8 switches are in PCube$(2, 3, 2)$, as illustrated with identifiers in lower 3-dimensions $00x$, $01x$, $10x$, $11x$, $0x0$, $1x0$, $x01$, $x11$.

Let us now consider the general case. A PCube$(n, k, q)$ ($k \geq q > 2$) is constructed from $n$ PCube$(n, k-1, q-1)$ and a new layer of $n^{k-1}$ switches. As we know, a server in a PCube$(n, k, q)$ is denoted as $a_k a_{k-1} \dots a_1$. $0a_{k-1} \dots a_1$ therefore denotes the servers in the first PCube$(n, k-1, q-1)$ and $1a_{k-1} \dots a_1$ denotes the servers in the second PCube$(n, k-1, q-1)$, etc. The newly introduced switches are denoted as $x b_{k-1} \dots b_1$. Server $a_k a_{k-1} \dots a_1$ is connected to switch $x a_{k-1} \dots a_1$. Eight switches that connect the two sub-PCubes are shown at the top of Fig. 3.

From the recursive construction procedure, we can explicitly identify the set of switches in any PCube$(n, k, q)$ ($n \geq 2, k \geq q \geq 2$). The set of switch identifiers at the $i^{\text{th}}$ dimension is listed as follows:

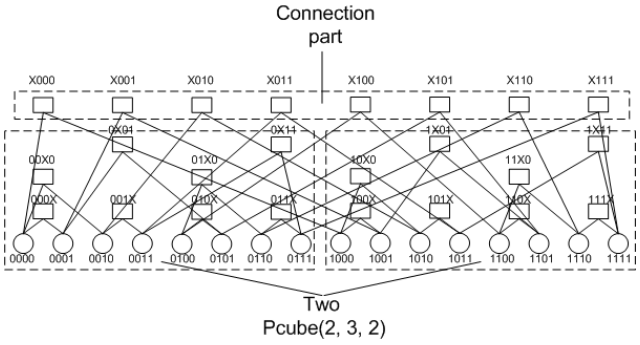



Fig. 3. Constructing PCube(2, 4, 3) with two PCube(2, 3, 2).

- $b_k b_{k-1} \dots b_2 x$ for $i = 1$;
- $b_k b_{k-1} \dots b_{i+1} x e_{i-1} o_{i-2} \dots o_1$ for $i \in [2, k-q+1]$;
- $b_k b_{k-1} \dots b_{k-q+3} x b_{k-q+1} o_{k-q} \dots o_1$ for $i = k-q+2$;
- $b_k \dots b_{i+1} x b_{i-1} \dots b_1$ for $i \in [k-q+3, k]$.

### C. Graph Properties of PCube Structures

In this section, we will establish a number of important graph properties of PCube$(n, k, q)$.

***The size of PCube*$(n, k, q)$.** Based on our construction method, we can obtain the following theorem with respect to the size of PCube.

*Theorem 1:* PCube$(n, k, q)$ is constructed by $qn^{k-1}$ switches with $n$ ports and $n^k$ servers with $q$ ports.

*Proof:* The number of servers can be got straightforward because of the way of PCube construction. Since each server is connected to $q$ switches and each switch is connected to $n$ servers. We can get that the number of switches is $qn^{k-1}$. ■

With Theorem 1, we may infer the amount of energy consumption using a general PCube$(n, k, q)$ structure. Considering both network switches and network interface cards (NICs), the energy cost of PCube$(n, k, q)$ can achieve $\frac{q}{k}$ of BCube$_{k-1}(n)$ by turning the unused switches and NICs off because BCube is a special case of PCube.

***The subgraph property of PCube.*** A PCube structure can not only be deployed incrementally based on BCube, but also be evolved from another configuration of PCube. One of the most essential properties of PCube can be derived from its construction algorithm:

*Theorem 2:* PCube$(n, k, q-1)$ is a subgraph of PCube$(n, k, q)$, with the same set of servers, when $q > 2$.

*Proof:* Compare PCube$(n, k, q)$ and PCube$(n, k, q-1)$, only when a switch marked $x$ at the $(k-q+2)^{\text{th}}$ and $(k-q+3)^{\text{th}}$ positions is different between PCube$(n, k, q)$ and PCube$(n, k, q-1)$. It is not difficult to reveal that the set of switches in PCube$(n, k, q-1)$ is a subset of the switches set in PCube$(n, k, q)$ in both of the dimensions.

Therefore, PCube$(n, k, q-1)$ $(q > 2)$ can be obtained by powering some switches off in PCube$(n, k, q)$, and its set of switches is a subset of PCube$(n, k, q)$. ■

With this property, when building a data center with $n^k$ servers and the bandwidth damand is not as high as $k$, it is possible to build a non-full PCube$(n, k, q)$ instead of full PCube$(n, k, k)$ or BCube at the beginning. By powering off some switches in PCube$(n, k, q_1)$, the structure of the network will evolve to PCube$(n, k, q_2)$, $(k \geq q_1 > q_2 \geq 2)$. For example, it is possible to build a PCube$(8, 6, 4)$ using four-port servers, instead of six-port servers in BCube.

***The diameter of PCube.*** To investigate the diameter of the PCube structure, defined as the maximum length of the shortest paths, we start with a study of path length in the primary PCube.

It is clear that every server has two ports that connect to switches in the primary PCube. One connects to a switch at the $1^{\text{st}}$ dimension. Depending on the identifier at the last several dimensions, the other port connects to a switch at a corresponding dimension.

According to the construction of primary PCube, transforming the $i^{\text{th}}$ dimension requires preparation that transforms all lower dimensions. The shortest path algorithm in the primary PCube is described in Algorithm 1.

**Algorithm 1** Find the shortest path in primary PCube$(n, k, 2)$

`getShortestPathP`(node A, node B)

**Step 1**: If (A = B) `return` emptyPath.
**Step 2**: Let $p$ be the highest dimension that $a_p \neq b_p$. If $p = 1$, `return edge`(A, B).
**Step 3**: Let node C be A. Find the smallest $p'$ such that for all $i \in [p', p-2]$, $b_i$ is odd. If such $p'$ does not exist, go to step 4. If $p = k$ or $b_{p-1}$ is even, let $c_i$ be $b_i$ where $i \in [p', p-1]$, and $c_i$ be $a_i$ or $a_i \oplus 1$ where $i \in [1, p'-1]$, depending on $a_i$ is even or odd.
**Step 4**: Let D be C, except $d_p = b_p$.
**Step 5**: `return getShortestPathP`(A, C) + `edge`(C, D) + `getShortestPathP`(D, B).

Algorithm 1 is recursively defined by finding a relay edge (C, D). Transforming the $p^{\text{th}}$ dimension requires some preliminary work on transforming all lower dimensions. Therefore, we find the relay edge from the highest dimension, as defined in step 2. Since it is entirely possible that we could get some other dimensions ready during this preliminary work, step 3 establishes how an optimal relay node C is found, which is not only on the shortest path but also getting most dimensions in place.

Based on the shortest path algorithm in the primary PCube, the shortest path algorithm in a general PCube$(n, k, q)$ $(q > 2)$ can be recursively defined as follows:

When the two servers are in the same primary PCube, the path can be obtained by directly call the shortest path algorithm for the primary PCube defined in Algorithm 1. Otherwise, when two servers are in different sub-PCubes we firstly route starting server A to the server with the same id but in the sub-PCube where end server B locates, and recursively find the shortest path in the sub-PCube.

Based on these ideas, we are able to prove the following theorem with respect to the *diameter* of PCube, defined as the maximum length of the shortest paths.

*Theorem 3:* The diameter of PCube$(n, k, q)$ is $2^{k-q+1}+(q-2)$.

*Proof:* We consider the primary PCube$(n, k, 2)$ first. According to Algorithm 1, the distance of transforming the first dimension is 1 hop, *i.e.,* $d(1) = 1$. To transform the $i^{\text{th}}$ dimension, it has to transform at most $i-1$ lower dimensions for preparation. Therefore, $d(i) = \sum_{j<i} d(j)$. In addition, by getting the highest dimension in place, the $(k-1)^{\text{th}}$ dimension will be in place, since there are no restrictions for the $(k-1)^{\text{th}}$ dimension. Therefore, the diameter would be $d = d(k) + \sum_{j\in[1,k-2]} d(j) = 2^{k-1}$.

When $q > 2$, the PCube is constructed with $n$ smaller sub-PCube$(n, k-1, q-1)$. For two servers in the same sub-PCube, the distance would be no larger than the diameter of PCube$(n, k-1, q-1)$. For two servers in different sub-PCubes, it needs two more links to get to the same sub-PCube. Thus, the diameter for PCube$(n, k, q)$ is $d(n, k, q) = d(n, k-1, q-1)+1$. Therefore, it is easy to derive that $d(n, k, q) = 2^{k-q+1}+(q-2)$. ■

The diameter of a PCube structure is closely related to network performance such as latency and all-to-all throughput. Theorem 3 suggests that the diameter of PCube is acceptable, compared with other server-centric structures such as DCell [13], whose diameter is $2^{k+1}$. As a result, PCube is able to achieve an acceptably low latency and a reasonably high all-to-all throughput.

***Bandwidth availability in PCube.*** We now turn our attention to the bandwidth availability in PCube. We show that the throughput achieved by PCube is able to offer a $q$ times speedup, with respect to both one-to-one and one-to-all traffic.

*Theorem 4:* There are $q$ edge-disjoint paths between any two servers in PCube$(n, k, q)$.

*Proof:* We prove this theorem by constructing $q$ edge-disjoint paths, using Algorithm 2 for primary PCube and an modified algorithm of Algorithm 2 for general PCube.

We first study the primary PCube. Considering every server with exactly two links, the shortest path occupies one link at the starting point and the end point. We could find another path using the rest of links. Algorithm 2 presents the method for obtaining two edge-disjoint paths in the primary PCube.

In Algorithm 2, we find the path deductively, by using the vacant link from each server while always aiming at node B.

Now we turn to the general PCube structure. There are $q$ edge-disjoint paths between any pairs of servers in PCube$(n, k, q)$. These edge-disjoint paths can be found recursively by finding $q-1$ edge-disjoint paths in a sub-PCube first, and the disjoint path finding method of a general PCube is relatively complex and the general idea is presented in the following:

*In the first case*, the two servers are in two different sub-PCubes of $(k-1)$-dimensions with different id. We take $q-1$ edge-disjoint paths from A to $a_k b_{k-1} \dots b_1$, the server with the same id in the sub-PCube of $(k-1)$-dimensions as B. For $q-2$ of these paths, they cross the $k^{\text{th}}$ dimension at the second server. The remaining path extends to two paths, which cross the $k^{\text{th}}$ dimension at the first and the last servers.

**Algorithm 2** Find two edge-disjoint paths in the primary PCube$(n, k, 2)$

`getMultiPathP`(node A, node B)

**Step 1**: Path1 = `getShortestPath`(A, B)

**Step 2**: Start with the unused link from server A. For each server arrived, there will be only one link out, which is followed. For each switch arrived, the next hop would be $c_i = b_i \oplus 1$ or $b_i$, depends on the odevity required to use the link of different high dimensions in the next hop.

**Step 3**: If all dimensions are ready except one, and the direct link is used, switch another dimension to an arbitrary one, and switch these two dimensions respectively. Path2 is this path ended at B.

**Step 4**: `return` {Path1, Path2}.

*In the second case*, the two servers are directly connected at the $k^{\text{th}}$ dimension. The $q$ edge-disjoint paths can be easily constructed through the $q$ directly connected switches by one or three server hops.

*In the last case*, the two servers are in the same sub-PCube of $(k-1)$-dimensions. Recalling the construction from PCube$(n, k-1, q-1)$ to PCube$(n, k, q)$, there are $q-1$ edge-disjoint paths from A to B within this sub-PCube. Another path needs a detour to another sub-PCube.

In summary, we have provided algorithms finding $q$ edge-disjoint paths in PCube$(n, k, q)$. Therefore, the correctness of Theorem 4 has been proved. ■

Theorem 4 suggests that there exist $q$ parallel paths between any pairs of servers. Therefore, the one-to-one speedup is $q$ by sending packets through all $q$ paths simultaneously. Compared to a BCube structure of the same size, PCube$(n, k, q)$ achieves $\frac{q}{k}$ of the one-to-one throughput. In the next step we will show that PCube can build $q$ edge-disjoint arborescences to speed up one-to-all traffic.

*Lemma 1:* For any graph $G$, there exist $k$ mutually edge-disjoint arborescences rooted at $r$, if and only if there are $k$ mutually edge-disjoint paths from $r$ to every other node. [16]

An arborescence at root $r$ is a spanning tree with directed edges, where there exists a path from $r$ to all nodes in the graph. Since the switches in data centers are mostly full duplex, arborescences describe network properties better.

*Theorem 5:* In PCube$(n, k, q)$, there exist $q$ mutually edge-disjoint arborescences rooted at any server.

*Proof:* According to Lemma 1 and Theorem 4 that $q$ edge-disjoint paths exist between any two servers, this theorem is proved. ■

The $q$ edge-disjoint arborescences can be found by Tarjan's algorithm [17]. Theorem 5 suggests that the network achieves a one-to-all speedup of $q$ by building $q$ edge-disjoint arborescences, which is better than existing switch-centric structures, such as ElasticTree[11]. Comparing with BCube, the one-to-all throughput of PCube$(n, k, q)$ is also $\frac{q}{k}$ of the BCube.

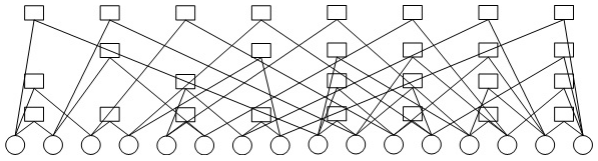


Fig. 4. An example of Hybrid PCube$(2, 4, \{2, 3\})$.

In closing, by evaluating graph properties of PCube structures, we are convinced that it achieves lower energy costs without sacrificing much margin with respect to performance.

## IV. Hybrid PCube

We have introduced the *Pure* PCube, which offers the same bandwidth and power consumption throughout the entire network. However, traffic locality is common in data center networks, which may lead to further energy saving. For example, we have collected traffic traces for storage service in an operational data center, as illustrated in Fig. 1(b). The traffic from the top 5 servers occupy more than 40% of the total traffic, and top 20 servers occupy about 80% traffic most of the time, while the number of servers in the entire network is in the thousands. In other words, most servers do not need high bandwidth in which occasion offering different servers different bandwidth would be helpful. In this section, we introduce *Hybrid PCube*, an interconnect method that provides different levels of bandwidth in different regions of the network.

In the construction method of Sec. III, to build a PCube with $(k+1)$-dimension, we add $n^k$ switches connecting $n$ sub-PCube of $k$-dimension with the same $q$. However, it is feasible to connect $n$ sub-PCube with a different bandwidth parameter $q$. Fig. 4, for example, illustrates a PCube in which the $4^{\text{th}}$ dimension switches connect PCube$(2, 3, 2)$ and PCube$(2, 3, 3)$ as its sub-PCube.

Taking Hybrid PCube into consideration, we extend the definition of the bandwidth parameter $q$ to $q(k)$, indicating the bandwidth at the $k-$dimension PCube. $q(k)$ can be an integer in $[2, k]$ for a Pure PCube, or a sequence of $n$ bandwidth parameters $\{q_0(k-1), q_1(k-1), \ldots q_{n-1}(k-1)\}$ for a Hybrid PCube connecting $n$ $(k-1)$-dimension PCubes whose bandwidth parameters are $q_0(k-1), q_1(k-1), \ldots, q_{n-1}(k-1)$, respectively.

Now we discuss the bandwidth availability in a Hybrid PCube. Consider a Hybird PCube with bandwidth parameter $\{q_0(k-t), q_1(k-t), \ldots, q_{n-1}(k-t)\}$. Servers in the $i^{\text{th}}$ sub-PCube have at most $q_i(k-t)+t$ edge-disjoint paths to other servers. Therefore, there are $\min(k-k_1+q_1(k_1), k-k_2+q_2(k_2))$ edge-disjoint paths between servers in the two sub-PCubes.

Furthermore, since there are $\min_{i\in[1,t]}(k-k_i+q_i(k_i))$ edge-disjoint paths within these sub-PCubes, Lemma 1 indicates $\min_{i\in[1,t]}(k-k_i+q_i(k_i))$ mutually edge-disjoint arborescences exist. Therefore, the broadcast speedup within some Pure sub-PCubes with bandwidth parameters $q_1(k_1), q_2(k_2), \ldots, q_t(k_t)$ in a Hybrid PCube of $k$-dimension is $\min_{i\in[1,t]}(k-k_i+q_i(k_i))$.

Clearly, Hybrid PCube offers biased bandwidth to different sub-PCubes. In addition, the two theorems above indicate that the total bandwidth is constrained by the sub-PCube with the lowest bandwidth. A PCube which provides equal bandwidth everywhere avoiding bottleneck edges has the ability to evolve to a Hybrid PCube. In the next section, we introduce how a Hybrid PCube can be configured according to traffic demands.

## V. Dynamic Configuration and Routing of PCube

The objective of configuring PCube is to dynamically adjust the network topology in order to provide sufficient throughput based on the traffic demand, and to conserve as much energy as possible. Consider a data center with thousands of flows. It is unnecessary and infeasible to monitor the network-wide traffic demand precisely. In addition, the network cannot respond to the traffic change quickly by monitoring current traffic, due to the starting time of the switch [10]. Therefore, in PCube, servers report their bandwidth requirement to a monitoring server, referred to as the *network manager*.

The PCube offers coarse-grained bandwidth level from 2 to $k$, so that every server sends its bandwidth requirement in $[2, k]$ to the network manager. A server sends its new bandwidth requirement, when the total traffic, including both server traffic and relay traffic, exceeds the current bandwidth, or can be served by a lower bandwidth. Since bandwidth availability is coarse-grained, this adjustment is acceptable.

Of course, to shrink or to extend the PCube still brings overhead on energy consumption and delay. To overcome the overhead cost we should make a trade off here, *e.g.,* adjust the length of the server report time period. The response time could also be optimized by using rapid response devices[18] such as SSD or use a asynchronous monitor to catch up the potential traffic volumes peak in the near future.

Assuming the network manager receives all server's bandwidth requirement $b_i$, where $i$ is the id of the server and $b_i$ is the larger one of the ingress and egress requirements, Algorithm 3 obtains the bandwidth parameter for a Hybrid PCube to satisfy the bandwidth needs.

**Algorithm 3** Building a Hybrid PCube based on server requirements

`getHybridPCube`(int k, subCube C)

**Step 1**: If the requirements of all servers in $C$ are 2, `return` 2.

**Step 2**: For all sub-PCube $C_i$ of $C$ at the $(k-1)$-dimension, $q_i \leftarrow$ `getHybridPCube`$(k-1, C_i)$.

**Step 3**: `return` $\{\max(2, q_0-1), \max(2, q_1-1), \ldots \max(2, q_{n-1}-1)\}$.

Algorithm 3 builds a Hybrid PCube and returns its bandwidth parameter. As stated in Sec. IV, the bandwidth parameter can either be an integer or be a set of bandwidth parameters. This algorithm may also build a Pure PCube, if all servers report the same bandwidth requirement.

PCube employs dynamically configured source routing. When the network manager decides how the PCube structure

can be transformed, it also calculates the modified routing paths. With global information, the network manager is able to balance the load on the new topology. Both topology transformation and routing information is broadcast to all servers. If the network manager decides to power some switches off, switches will be powered off after a few seconds, which is enough for most servers to receive and react to the topology change. On the other hand, if some switches need to start again, they will start immediately, since a longer period of time, on the order of 10-20 seconds, is necessary for switches to resume from their power-off states.

When servers receive the topology transformation message, they immediately switch to a new path for every flow according to the routing message received from the network manager. If servers are not able to receive these messages on time, which rarely happens, they will react as if they are dealing with link failures and probe for new paths [4].

Other details of routing are similar to BCube source routing [4], since BCube and PCube share essential properties.

## VI. Evaluation

In this section, we evaluate PCube in four aspects. We obtain the amount of relay traffic in PCube, demonstrate the stepwise energy consumption with traffic, evaluate the performance of PCube, and finally, we establish our simulation results with respect to power efficiency using traffic samples collected in a real-world production data center.

### A. Average Relay Ratio

First, in order to obtain the total traffic at one server, we get the average relay ratio $r_q$ for different $q$ in PCube$(8,4,4)$ with 4096 servers, indicating the relay traffic is $r_q$ times of the service traffic. In other words, the total traffic at one server is $v(1+r_q)$, when the service traffic is $v$. We randomly generate the traffic on the PCube$(8,4,4)$ and Simulation results of $r_q$ on the all-to-all traffic pattern are shown in Table I. In this traffic pattern, there are flows between all pairs of servers, and the algorithms described in Sec. III-C is used to find the paths. The relay ratio is the highest in the all-to-all traffic pattern, for this reason $v(1+r_q)$ is larger than the bandwidth demand in reality.

TABLE I
The ratio of relay traffic to server traffic

| structure | ratio |
|---|---|
| PCube$(8,4,4)$ | 2.500 |
| PCube$(8,4,3)$ | 3.048 |
| PCube$(8,4,2)$ | 5.161 |

### B. Energy Consumption

With the relay ratio, we can obtain the relationship between traffic and energy consumption by switches and NICs. We manually generate three kinds of traffic. Traffic pattern 1 indicates all servers have the same traffic demands. In traffic pattern 2, all servers have 10% traffic except the traffic from one server. In traffic pattern 3, one server has 100% traffic and all other servers have 10% traffic.

We use PCube$(8,4,4)$ as testbed, PCube$(8,4,4)$ can shrink to PCube$(8,4,3)$ and PCube$(8,4,2)$, hence the stairs of energy consumption have three steps. The result shows that: With traffic pattern 1, each step accounts for 25% of the total energy. With respect to traffic pattern 2, the PCube evolves from PCube$(8,4,2)$, to PCube$(8,4,3)$ and finally to PCube$(8,4,\{2,\ldots,2,3\})$. Therefore, when only one server has a significant demand of traffic, the energy consumption at the last step is only 3.125% of the total traffic, which is $\frac{1}{n}$ of 25%. Shown by traffic pattern 3, it is also 3.125% more energy when PCube$(8,4,3)$ grows to PCube$(8,4,\{2,\ldots,2,3,3\})$.

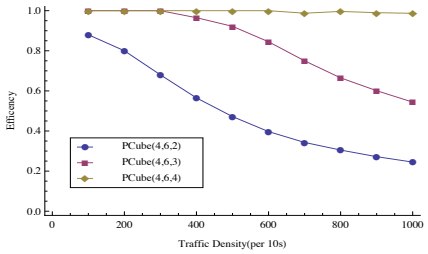



Fig. 5. relationship between the traffic density and efficiency of PCube$(4,6,q)$.

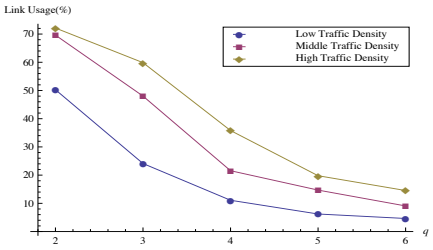



Fig. 6. The relationship between bandwidth parameter $q$ and link usage ratio.

### C. Performance Evaluation

The average amount of traffic handled in a short time slide obviously indicates the *efficiency* of a network. We measured the link usage ratio and the efficiency to evaluate the performance of PCube.

Fig. 5 shows the relationship between the traffic density and efficiency of PCube$(4,6,q)$. It is clear that while the traffic density grows, the PCube with higher bandwidth $q$ stays more efficient than the lower bandwidth ones. Therefore, we should choose a higher bandwidth if the network traffic demand grows.

Fig. 6 indicates the relationship between the bandwidth of PCube and the link usage ratio when the traffic demand keeps unchanged. The figure shows that the lower bandwidth parameter $q$ can help to achieve a higher link usage ratio, which also means, a higher energy efficiency. The traffic density certainly affect the link usage ratio as well. In the figure, high density means about 75 traffic queries per second, while 50 and 25 queries per second for middle and low respectively.

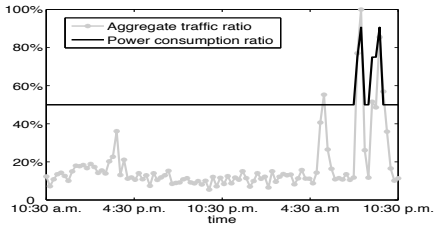



Fig. 7. Simulation results on energy efficiency for a machine learning service.

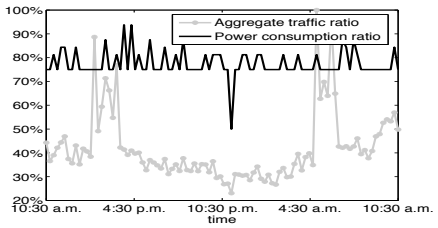



Fig. 8. Simulation results on energy efficiency for a storage service.

Comparing with BCube, which can be refered as a PCube with full bandwidth parameter, PCube offers a scalable bandwidth which could definitely increase the link usage ratio and decrease the processing capacity of the network. When the traffic density is lower than the peak, which is common in general usage of data center, the PCube provides adaptable method to balance the needs of performance and energy consumption.

### *D. Efficiency in the Real World*

To demonstrate the efficiency of PCube in the real world, we have collected traffic traces from a non-blocking data center with several kinds of service including machine learning using MapReduce, storage, etc. We take samples every 15 minutes on the current egress rate of every server on its own service for a 24-hour period. In other words, the sampling does not count relay traffic, if any. Some sampling results are shown in Fig. 1. We extend the sampling proportionally and map them randomly to 4,096 servers in PCube.

Fig. 7 illustrates the daily energy consumption for a machine learning service on July 20, 2010. The traffic is light most of the time, thus leading to a 50% energy consumption of network devices including NICs and switches in comparison with BCube, where 50% energy is the minimal energy consumption for PCube with $k = 4$. In fact, in traffic traces for other days, the daily peak might be less than 10% of the peak on July 20, leading to the minimal energy consumption in an entire day. Therefore, applying PCube leads to considerable energy saving for this kind of service.

Fig. 8 illustrates another case. With respect to the storage service, a few servers occupy about 80% of the total traffic as shown in Fig. 1(b). In other words, there are a few servers with a substantial amount of traffic volume. In addition, the traffic of those servers do not vary much, even when the total traffic grows significantly. Shown in the simulation results, the energy consumption is dominated by these servers with large traffic. Therefore, the energy consumption does not increase with the total traffic.

## VII. Conclusion

In this paper, we have proposed a server-centric data center structure PCube. Compared with existing data centers, PCube succeeds in the flexibility to dynamically adjust the network topology according to the traffic demands. Therefore, PCube not only offers full bisection bandwidth, but also conserve substantial amount of energy costs. PCube also provides an alternative for low cost data center construction, with several favorable graph properties . In our experiments on real-world traffic samples, PCube has shown its significant improvement on energy conservation.

## References

[1] M. Gupta and S. Singh, "Greening of the Internet," *in Proc. ACM SIGCOMM*, pp. 19 – 26, Aug. 2003.

[2] J. G. Koomey, C. Belady, M. Patterson, A. Santos, and K.-D. Lange, "Assessing Trends over Time in Performance, Costs, and Energy Use For Servers," *http://www.intel.com/assets/pdf/general/servertrendsreleasecomplete-v25.pdf*, August 2009.

[3] "EPA Report on Server and Data Center Energy Efficiency." [Online]. Available: http://www.energystar.gov/index.cfm?c=prod_development.server_efficiency_study.

[4] 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*, pp. 63–74, 2009.

[5] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," *in Proc. ACM SIGCOMM*, pp. 63–74, 2008.

[6] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a Scalable and Flexible Data Center Network," *in Proc. ACM SIGCOMM*, pp. 51–62, 2009.

[7] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding Data Center Traffic Characteristics," *in Proc. 1st ACM workshop on Research on Enterprise Networking (WREN)*, pp. 65–72, 2009.

[8] M. Gupta, S. Grover, and S. Singh, "A Feasibility Study for Power Management in LAN Switches," *in Proc. 12th IEEE International Conference on Network Protocols (ICNP)*, pp. 361–371, Oct. 2004.

[9] M. Gupta and S. Singh, "Using Low-Power Modes for Energy Conservation in Ethernet LANs," *in Proc. IEEE INFOCOM*, pp. 2451–2455, May 2007.

[10] [Online]. Available: www.dlink.com.

[11] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving Energy in Data Center Networks," *in Proc. 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, April 2010.

[12] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang, "MDCube: A High Performance Network Structure for Modular Data Center Interconnection," *In Proc. ACM CoNEXT*, December 2009.

[13] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "DCell: A Scalable and Fault-tolerant Network Structure for Data Centers," *in Proc. ACM SIGCOMM*, pp. 75–86, 2008.

[14] G. László and T. T. Anh, "How can architecture help to reduce energy consumption in data center networking?" *e-Energy '10*, pp. 183–186, 2010. [Online]. Available: http://doi.acm.org/10.1145/1791314.1791343

[15] S. Yunfei, L. Dan, and X. Mingwei, "Energy-aware Routing in Data Center Network," *In Proc. Green Networking*, pp. 1–8, 2010. [Online]. Available: http://doi.acm.org/10.1145/1851290.1851292

[16] J. Edmonds, "Edge-disjoint Branches," *Combinatorial Algorithms, Academic Press*, pp. 91–96, 1973.

[17] R. E. Tarjan, "A Good Algorithm for Edge-disjoint Branching," *Information Processing Letters*, vol. 3, no. 2, pp. 51–53, 1974.

[18] M. David, G. B. T., and W. T. F., "Powernap: eliminating server idle power," *SIGPLAN Not.*, vol. 44, pp. 205–216, March 2009. [Online]. Available: http://doi.acm.org/10.1145/1508284.1508269