Computer Science and Information Systems 14(3):611–627 https://doi.org/10.2298/CSIS170202019T

BHyberCube: a MapReduce Aware Heterogeneous Architecture for Data Center

Tao Jiang, Huaxi Gu, Kun Wang, Xiaoshan Yu, Yunfeng Lu

State Key Laboratory of ISN, Xidian University, Xi'an, China
taojiang127@foxmail.com

Abstract. Some applications, like MapReduce, ask for heterogeneous network in data center network. However, the traditional network topologies, like fat tree and BCube, are homogeneous. MapReduce is a distributed data processing application. In this paper, we propose a BHyberCube network (BHC), which is a new heterogeneous network for MapReduce. Heterogeneous nodes and scalability issues are addressed considering the implementation of MapReduce in the existing topologies. Mathematical model is established to demonstrate the procedure of building a BHC. Comparisons of BHC and other topologies show the good properties BHC possesses for MapReduce. We also do simulations of BHC in multi-job injection and different probability of worker servers' communications scenarios respectively. The result and analysis show that the BHC could be a viable interconnection topology in today's data center for MapReduce.

Keywords: Data center, MapReduce, topology.

1. Introduction

In a data center network, up to a few thousands of servers are interconnected via switches to form the network infrastructure. Data center networks (DCN) possess the characteristic of high performance computing (HPC) and mass storage naturally [1]. Based on these properties, data center is used as distributed storage and computing infrastructures for some online applications such as search, social networks, E-learning [2], and web 2.0 technology [3]. In addition, these data centers also support infrastructure services, such as distributed file systems (e.g., GFS [4, 5] and Chubby [6]), structured storage (e.g., BigTable [7], and Megastore [8]), distributed execution engine (e.g., MapReduce, Dryad and percolator) and large computing units' schedulers (e.g., Omega [9]). Traditional resource efficient architecture has become a barrier to meet the diverse application requirements, and it is inevitable that the future network should be application driven [10]. A data center should be equipped with specific infrastructure services to manage and process massive data efficiently [11]. MapReduce is one of the most important distributed execution engines for data processing. MapReduce works by dividing input files into chunks and processing these in a series of parallelizable steps in a good control and execution model. MapReduce is used by companies such as Facebook, IBM, and Google to process or analyze massive data sets [12].

In recent years, the scale of data center is growing at an exponential rate. Some Internet service providers, like Microsoft, are even doubling the number of servers every 14 months, exceeding Moore's Law. Additionally, diverse services emerged in data centers, calls for an improvement of the topological performances of a data center network, including scalability and reliability, etc. But the current DCN interconnects all the servers using a tree hierarchy of edge-switches, core-switches or core-routers generally. It is increasingly difficult to meet the requirements, such as scalability and high network capability. As some solutions, several new DCN architectures have been proposed, such as DCell [13], FiConn [14], and BCube [15]. These architectures have optimized some fundamental topological properties and provide good scalability and reliability. However, considering the distributed data processing mechanisms running on them, they may not have good performance. There are two main reasons. First, many distributed data processing mechanisms, especially MapReduce, require that all servers being partitioned into master servers and worker servers [16]. However, most data center architectures treat all the servers equally [17]. Second, as suggested by the name, mapping and reducing constitute the essential phases for a MapReduce job. Therefore, it requires a strong inner relationship among the servers that execute these operations to exchange the intermediate results. However, these new architectures ignore this relationship in MapReduce job. Obviously, we need dedicated data center architecture to meet users' increasing new service requirements in a complex MapReduce.

In this paper, a new network, called BHyberCube network (BHC) is proposed. BHC is a recursively defined topology to interconnect servers. Each worker server connects several other worker servers in a hypercube unit and one master server. Each master server not only connects several worker servers, but also connects other master servers via a high level switch. The interconnection relationships among master servers and worker servers are determined according to the procedure of MapReduce. A high-level BHC is recursively built from many low-level ones. Due to its heterogeneous architecture, it is well suited to support the data processing procedure of MapReduce. The evaluation and analysis results show that BHC has good topological performance with scalability.

A routing algorithm designed for BHC is also proposed in this paper. This routing algorithm is designed for four scenarios for MapReduce on BHC, routing between a master worker and its worker servers, routing between two worker servers belonging to the same master server, routing between master servers and routing between two worker servers belonging to different smallest recursive units. This routing algorithm is designed to utilize the recursively-defined structure, and accelerate the procedure of MapReduce by loop iterations.

The rest of this paper is organized as follows. Section 2 introduces the related work and our motivation. Section 3 proposes the physical structure and a construction method for BHC and evaluates several topological properties of BHC. Section 4 describes the routing algorithm for MapReduce on the BHC. Section 5 shows the procedure of MapReduce executing in BHC. Section 6 presents simulation results of multi-job injection and different probability of worker servers with dependency relationship. Section 7 concludes this paper.

2. Related Work

In these section, we will introduce the existing datacenters architectures, and the details of MapReduce. The motivation will be also demonstrated.

2.1. Data Center Network Architectures

Existing datacenters generally adopt traditional tree architectures, like Fat tree [18] to interconnect servers [19] [20]. In 0, a generic Fat tree network is presented. This architecture supports a variety of links between the aggregation switches and the core switches, which makes it an architecture with high connectivity and reliability. However, this traditional tree architecture does not scale well.

Some of architectures, like DCell [13] and BCube [15], are recursively constructed, as demonstrated in Figure 1. A high-level structure utilizes a lower-level structure as a unit and connects many such units by means of a given recursive rule [21]. One of this recursive rule's advantages is that more servers can be added into a hierarchical DCN without destroying the existing structure when the level of a network is increasing. Hence, the hierarchical topology is scalable naturally.

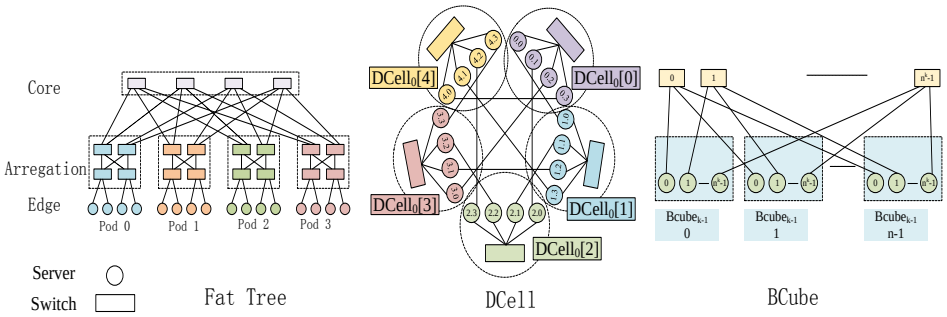

Fig. 1. Fat Tree, DCell and BCube architectures in DCN

2.2. MapReduce

The MapReduce paradigm has emerged as a highly successful programming model for large-scale data-intensive computing applications [22]. A complex MapReduce procedure processes a sequence of jobs, and each job consists of a map phase and a reduce phase [23 24].A unit based on MapReduce is composed of two server types: a master server and several worker servers. A master server controls many worker servers in executing map and reduce tasks. The master server coordinates MapReduce jobs. The worker server is responsible for running map tasks and reduces tasks. The map phase performs a map function where the master server partitions the input datasets into multiple even-sized smaller chunks and distributes them to the worker servers. Each chunk of the input is first processed by a map task, which will generate an enormous amount of intermediate (key, value) pairs on the local disks and report the keys and their

locations to the master server. The master node then partitions the (key, value) into different worker servers based on the keys. The reduce tasks will be activated to first pull the data from the map worker servers, and then apply a reduce function to the list of (key, value) pairs on each key [25]. Reduce tasks merge the intermediate values with the same key by means of predefined reduce programs and then generate the output values.

Considering implementation of MapReduce, the existing architectures addressed above, do not support the distribution data management or processing mechanisms like MapReduce very well. The reasons are as follows:

Homogeneous nodes. Existing DCN architectures do not partition servers into master servers and worker servers [26]. They simply assume that all servers possess the same function and interconnect all the servers in the same way. However, servers are classified into masters and workers based on the different functions in MapReduce [27]. Therefore, servers of different roles should be interconnected in dedicated ways.

Collective communication. In the MapReduce procedure, a master will control several worker servers, and assign different tasks to different worker servers simultaneously [28]. And among these worker servers, they will collect and transmit the intermediate information. Heavy collective operations communications happen in these phases. Hence, the topology for MapReduce should have a good performance on collective communications.

Network diameter. The topological properties should be sufficiently suitable in the DCN with the expanding of their scales. There will be a large number of data transmissions in a complex MapReduce [29]. Hence, it requires a low network diameter to shorten the transmission length between any pair of servers when the network is scaling up [30].

The BHC is motivated from the above analysis. It will treat the servers as a master server or a worker server, according to the function they will perform in the MapReduce. Leveraging the deployment of homogeneous nodes, BHC strongly supports the collective communications in the mapping and reducing phases. Furthermore, BHC employs recursive units, resulting a relatively low network diameter when the network scaling.

3. The BHC Architecture

Heterogeneous nodes, collective communications and network diameter are the main focus on network topologies proposed for DCN to support MapReduce. Servers are classified into masters and workers based on the different functions in MapReduce [27]. Servers in different roles should be interconnected in dedicated ways. If each master server interconnects its worker servers, it will improve collective communication greatly. Units are also implemented because they support collective communication naturally. The recursively defined architecture is implemented to reduce the network diameter when the scale increases.

Based on these observations, BHC is proposed for DCN to support MapReduce. BHC is a recursively-defined architecture with units attached.

3.1. BHC Architecture Specification

BHC uses servers, equipped with multiple network ports, and switches to construct its recursively defined architecture. In BHC, servers and switches are connected via communication links, which are assumed to be bidirectional. A high-level BHC is constructed from low-level BHCs. BHC_k ($k \geqslant 0$) denotes a level-k BHC. The smallest recursive unit of BHC and how to construct a high-level BHC recursively is presented as follows:

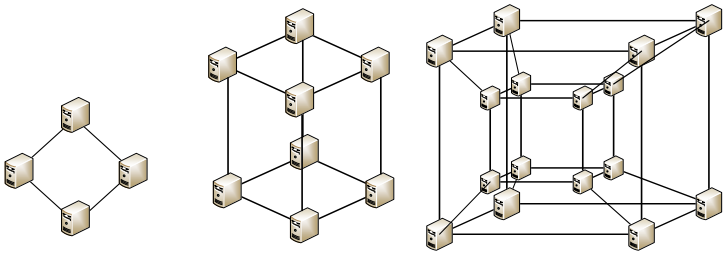

Fig. 2. level-0, 1, 2 worker units

1. Smallest recursive unit and worker units

BHC_0 is the smallest recursive unit. Meanwhile, it is also the building block to construct larger BHCs. It has W worker servers, M master servers also M switches with *P+1* ports, and *P* worker units.

The worker unit is constructed by several worker servers. These worker servers are interconnected by a *level-k* hypercube, for some different applications' requirements. 0 illustrates the *level-0, 1, 2* worker units.

In the BHC_0, each master server connects to a switch. The master servers do not connect each other directly. Neither do the switches. Each switch connects to *P* worker servers in each worker unit. So the number of worker unit in the BHC_0 is *P*.

The construction of a BHC_0 is as follows. A BHC_0 is constructed from P worker units and 2^{i+2} switches. 2^{i+2} switches are numbered from 0 to $2^{i+2}-1$. The *P* worker units are numbered from *0* to *P-1* and the worker servers in each worker unit are numbered from *0* to $2^{i+2}-1$. The *jth* ($j \in [0, P-1]$) port of the *ith* ($i \in [0, 2^{i+2}-1]$) switch is connected to the *ith* ($i \in [0, 2^{i+2}-1]$) worker servers in the *jth* ($j \in [0, P-1]$) worker unit.

There are four advantages for designing the smallest recursive unit in such a way.

Scalability. The worker unit is designed as a hypercube, which makes the worker unit increase exponentially. It means that BHC can scale the worker servers quickly and efficiently, to support different applications' requirement.

Collective communication. The worker servers can transmit intermediate information in the worker unit, which will support good collective communication performance.

Heterogeneous nodes. This recursive unit treats servers as masters and workers naturally, compared with the current recursive units, like in BCube and DCell.

Low percentage of switches. The switch connects with more worker servers, and a master server can control as many worker servers as possible. The evaluation will be presented in the following sections.

2. Bild BHC: the Procedure

As assumed above, the BHC_0 has W worker servers, M master servers also connect M switches with $P+1$ ports, and P worker units. Besides, the worker unit is assumed in the level i ($i \geqslant 0$). More generally, a BHC_k ($k \geqslant 1$) is constructed from P BHC_{k-1} and P *level-k* $P+1$-port switches.

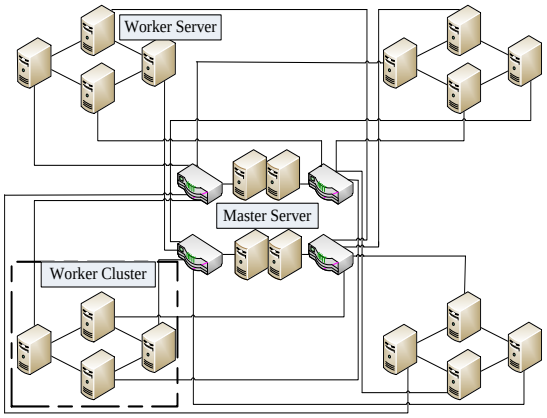

Fig. 3. An example of BHC_1 with $k=1$, $P=4$ and $i=0$

The construction of a BHC_k is as follows. BHC_{k-1} is numbered from *0* to *P-1*, the 2^{i+2} *level-k* switches from *0* to 2^{i+2}*-1* and the 2^{i+2} master servers in each BHC_{k-1} are numbered from *0* to 2^{i+2}*-1*. The *ith* ($i \in [0, 2^{i+2}-1]$) master servers in the *jth* ($j \in [0,P-1]$) BHC_{k-1} is connected to the *jth* ($j \in [0,P-1]$) port of the *ith* ($i \in [0, 2^{i+2}-1]$) *level-k* switch. 0 illustrates an example with $k=1$, $P=4$ and $i=0$, and 0 shows an example of BHC_k.

3.2. Properties of BHC

For a high-level BHC_k, it is constructed in the same way as stated above. If BHC_{k-1} has been built and each BHC_{k-1} has M^{k-1} master servers and W^{k-1} worker servers. Each BHC_{k-1} is treated as a virtual node, and fully connects these virtual nodes to form a BHC_k.

Theorem 1.

The number of master servers in a BHC_k is M_k, and $M_k = P^K 2^{i+2}$;

The number of worker servers in a BHC_k is W_k, and $W_k = P^{K+1} 2^{i+2}$

Based on the recursively defined structure of BHC, M_k depends on the P and M_{k-1}, and W_k also depends on the P and W_{k-1}. Equations 1 and 2 can be derived as follows:

The number of master servers in a BHC_k is M_K:

$$M_k = P \times M_{k-1} = \prod_{i=1}^{k} P \times M_0 = P^K 2^{i+2} \tag{1}$$

The number of worker servers in a BHC_k is W_K:

$$W_k = P \times W_{k-1} = \prod_{i=1}^{k} P \times W_0 = P^{K+1} 2^{i+2} \tag{2}$$

Theorem 1 shows that the number of worker servers and master servers scales based on the number of switch's ports and the worker unit's level. For example, when $K=3$, $P=6$ and i=2, a BHC_3 have as many as 3456 master servers and 20736 worker servers

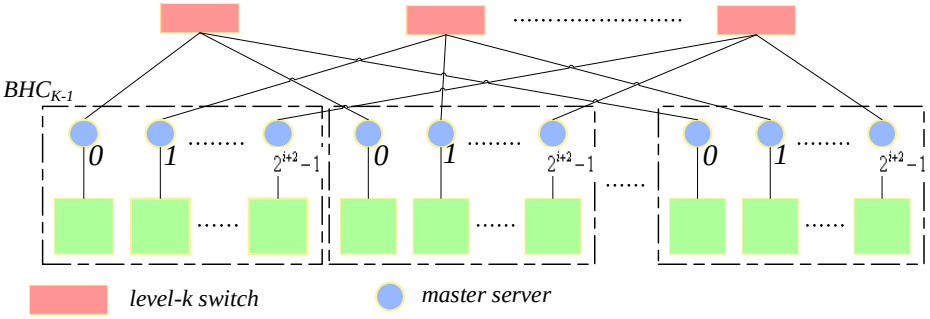

Fig. 4. BHC_k, a BHyberCube network

Bisection width denotes the minimal number of links to be removed to partition a network into parts of equal size. A large bisection width implies high network capacity and a more resilient structure against failures.

Theorem 2. The bisection width of BHC is $P \times 2^{i+1}$.

As the procedure of building a BHC_k addressed above, each of the P *level-k* switches has 2^{i+2} links connected to the *level-k-1* switches. Hence, it is easy to figure out that the bisection width of BHC is $P \times 2^{i+1}$.

Theorem 3. The network diameter of BHC is

$$3 < D_{BHC} < P + 2 \tag{3}$$

The network diameter is the longest path between any two servers. In the uniform traffic model, the maximal number of hops between any two master servers in a BHC_k is P, if they are in hi BHC_0s. For BHC_0, the maximal number of hops is 2. While in the MapReduce application, the master server distributes job chunks to the workers, and the workers process a map task to generate intermediate and report intermediate to the master. The master partitions the intermediate into workers and workers process a reduce task to generate the output values. Hence, three hops is the minimal distance to complete a job. *Theorem 3* is proven.

$Master_{K,j}$ denotes the any master server in a BHC_K, and s denote the sequence of a BHC_k which contains $Master_{K,j}$ in the BHC_K. The value of s is given as follows:

Theorem 4. The sequence of BHC_K $Master_{K,j}$ belongs to in the BHC_K is

$$s = j / P^k 2^{i+2} \tag{4}$$

According to Theorem 1 and Equation 1, the number of master servers is $P^K 2^{i+2}$ in a BHC_K. j is assumed as the sequence of $Master_{K,j}$ in the BHC_K. Therefore, s is the sequence of BHC_K $Master_{K,j}$ belongs to in the BHC_K.

Theorem 5. The number of master servers between $Master_{K,x}$ and $Master_{K,y}$ is

$$|x-y| = P^K 2^{i+2} \tag{5}$$

We assume that two master servers, denoted as $Master_{K,x}$ and $Master_{K,y}$, connect to the same switch at $level_{k+1}$, and belongs to a pair of adjacent BHC_ks in a BHC_{k+1}.

$|x-y|$ means the absolute value of x minus y in the Equation 5. Based on recursive rules, in this pair of adjacent BHC_ks, $Master_{K,x}$ and $Master_{K,y}$ are the only two master servers connected to the same switch at $level_{k+1}$. For a BHC_{k+1}, the number of switches at $level_{k+1}$ is $P \times 2^{i+2}$, and other master servers between $Master_{K,x}$ and $Master_{K,y}$, are connected to the other $P \times 2^{i+2} - 1$ switches. So the number of master servers between $Master_{K,x}$ and $Master_{K,y}$ is $P \times 2^{i+2} - 1$, namely $|x-y| = P \times 2^{i+2}$.

4. Routing in a BHC

According to the procedure of MapReduce in the DCN and the roles of the servers in MapReduce, the routing algorithm is designed for four scenarios [31]. The first is the routing algorithm between a master server and its worker servers, used for assigning map and reduce tasks. The second one is the routing algorithm between two worker servers that are controlled by the same master server, used for transmitting intermediate data. The third one is the routing algorithm among master servers, used for assigning jobs. The fourth one is the routing algorithm between two worker servers that belong to different smallest recursive units, used for transmitting the necessary data that are not stored on local disks. Because there are only one or two hops in the second routings, which can be addressed only in the smallest recursive unit, this paper mainly focuses on the third and fourth scenarios.

```
Algorithm 1 :AssigningJobs (int j, int L)
List ServersSought;
for  l=0; l<L; l++
 if Master_{I,y}'s worker severs hold the data for Job_l ;
   assign Job_l to Master_{I,y};
   Master_{I,y}.RoutingPath={ Master_{I,j} };
  Master_{I,y}.RoutingPath=FindRouting1(I- 1,j ,y);
  ServersSought.add(Master_{I,y} );
```

4.1. Master-to-master Routing

The routing algorithm among master servers depends on the job-assigning scheme of MapReduce service [32]. A master server that receives a multijob MapReduce request sends each job to the nearest master server, which controls the worker servers containing the necessary data for the job.

Algorithms 1 and 2 are proposed to implement the master-to-master routing algorithm for assigning MapReduce jobs on BHC. Here $Master_{I,j}$ means the *jth* master server in the BHC_I. $Master_{I,j}$ is supposed to receives a MapReduce service request, which needs to be assigned to L ($L > 1$) master servers. Algorithm 1 demonstrates the algorithm of assigning jobs. In Algorithm 1, Job_l ($0 \leqslant l \leqslant L$) denotes the jobs that need to be assigned to a master server. Algorithm 1 first finds the master server, denoted as $Master_{I,y}$ which controls the worker servers with the data required by Job_l. It then assigns Job_l to $Master_{I,y}$ and finds a routing path from $Master_{I,j}$ to $Master_{I,y}$ by citing Algorithm 2 and adds the routing path to the Path attribute of $Master_{I,y}$. Finally, it adds $Master_{I,y}$ to the object list *ServersSought*.

Algorithm 2 is designed to find a master-to-master routing path for assigning jobs. Algorithm 2 recursively records each node in the routing path from $Master_{I,j}$ to $Master_{I,y}$ from level I to level *0*, For level I, Algorithm 2 takes $Master_{I,j}$ and $Master_{I,y}$ as the source and destination nodes of the routing path, respectively. It determines if $Master_{I,j}$ and $Master_{I,y}$ connect to the same switch at level I through *Theorem 3*. Otherwise, according to *Theorem 4* and *Equation 3* and *4*, Algorithm 2 records the master server, namely $Master_{I,x}$, which not only connects to the same switch at level I with $Master_{I,j}$, but also belongs to the same BHC_{I-1} with $Master_{I,y}$. Above process is performed again for level *I-1*.with taking $Master_{I,x}$ as the new source node, also denoted as $Master_{I,j}$. This process is performed recursively until $Master_{I,y}$ is taken as the new source node or $Master_{I,j}$ and $Master_{I,y}$ belong to the same BHC_0. For the latter event, if the number of hops from $Master_{I,j}$ to $Master_{I,y}$ is larger than one, minimal master servers are further recorded in order in the routing path from $Master_{I,j}$ to $Master_{I,y}$. Otherwise, Algorithm 2 just records $Master_{I,y}$ as the last node and returns the whole routing path.

```
Algorithm 2 : FindRouting1 (int f , int j , int y)
int k = 0; int x = 0;
for  i = f ; i ⩾ 0; i--
   if  i > 0
     if j/P×2^(i+2)  ≠ y/j/P×2^(i+2);
       int h = (y-j) /P×2^(i+2);
       x = j + h ×P×2^(i+2);
       add Master_I,x to Master_I,y.RoutingPath;
       if  x == y
         return Master_I,y.RoutingPath;
      k = i ;
   break;
   if  i = 0
     if  j-y >2;
       for  x = j - 2; x>y; x-=2
         add Master_I,x to Master_I,y.RoutingPath;
       if  y -j >2
         for  x = j +2; x <y; x+=2
         add Master_I,x to Master_I,y.RoutingPath;
     add Master_I,y to Master_I,y.RoutingPath;
     return Master_I,y.RoutingPath;
FindRouting1 (k,x,y);
```

4.2. Worker-to-worker Routing

A worker server may need the data stored at another worker server, when it is executing a map or reduce task. Based on the master-to-master routing algorithm, Algorithm 3 is proposed as the worker-to-worker routing algorithm in BHC. Algorithm 3 adds two worker servers and the routing path between their master servers. Routing path can be obtained from Algorithm 2. $Worker_{j,m1}$ denotes any worker server which are controlled by $Master_{I,j}$, and $Worker_{y,m2}$ denotes any worker server controlled by $Master_{I,y}$. $Worker_{j,m1}$ and $Worker_{y,m2}$ are assumed not to be controlled by the same master server.

```
Algorithm 3: FindRouting2 (int j , int y , int m1, int m2)
  Worker_{j,m1}.RoutingPath = { Worker_{j,m1}, Master_{I,j} };
  Worker_{j,m1}.RoutingPath = FindRouting1(I-1,j,y);
  add Worker_{y,m2} to Worker_{j,m1}.RoutingPath;
  return Worker_{j,m1}.RoutingPath;
```

5. Map and Reduce on BHC

Based on routing algorithm described above, the jobs of a complex MapReduce are assigned to several master servers. These master servers will control a number of worker servers to execute the received jobs. Map and reduce operations are involved in the execution of each job. This procedure is demonstrated in Figure 5.

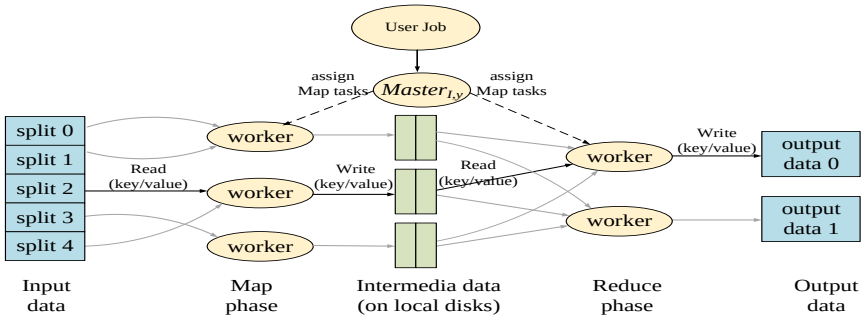

Fig. 5. The procedure of Map and Reduce on BHC

5.1. Map on BHC

Suppose that $Master_{I,y}$ receives a job. The number of map tasks is determined by the number of data chunks that job needs to process. The default mapping approach, which consists of three steps, is one map task for one data chunk. In the first step, $Master_{I,y}$ chooses some idle worker servers, named map worker servers, and assigns a map task to each of them. In the second step, map worker servers divide the corresponding input data into intermediate key/value pairs by means of predefined map programs and store

the intermediate data on local hard disks. In the third step, map worker servers feedback the keys of intermediate data to $Master_{I,y}$ and then send the number of waiting tasks in their local queues, namely their state information, to the corresponding master servers.

5.2. Reduce on BHC

The number of reduce tasks is determined by the types of intermediate data' keys. One reduce task can process one or several types of key/value pairs. But one type of key/value pairs is usually processed by only one reduce task. The default reducing approach consists of four steps. In the first step, $Master_{I,y}$ chooses some idle or not busy worker servers, named reduce worker servers, and assigns a reduce task to each of them. In the second step, according to the types of keys of their received reduce tasks, reduce worker servers fetch the intermediate data from the corresponding map worker servers. In the third step, reduce worker servers merge the same type of key/value pairs by means of predefined reduce programs to generate output values. In the fourth step, reduce worker servers feedback the output values to $Master_{I,y}$. They also send their state information to the corresponding master servers. The output data of some jobs might be the input data of other jobs. When $Master_{I,y}$ has finished its job, it sends the result directly to the master server that receives the next job, namely the next object in the object list FindedServers, which is derived from Algorithm 1. The routing algorithm between $Master_{I,y}$ and that master server can be obtained by means of Algorithm 2. The master server that executes the final job forwards its result to $Master_{I,j}$ through the routing path recorded in its Path attribute.

6. Properties and Simulation

In this section, a comparison of properties of BHC and BCube is demonstrated. Additionally, the performance of two scenarios: different numbers of jobs and different probability of communications in worker servers is shown respectively.

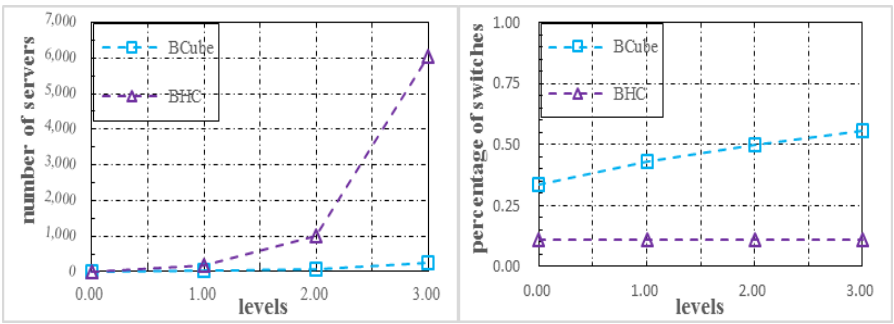

Fig. 6. The network size and percentage of switches of BCube and BHC in case that $N = 4$ servers in $BCube_0$ and $P = 8$ and $i = 0$ in BHC_i.

6.1. Comparison of Properties

Based on the Theorem 1, the network size and percentage of switches can be figured out with the level increasing. The comparison between BCube and BHC is illustrated by Figure 6.

Figure 6 shows the number of servers versus the number of levels in the network. The scalability of BHC is better than the BCube structure, when the level is higher than 3.

Figure 6 also shows the percentage of switches in BCube and BHC. The result of BHC is lower than BCube, which implies that BHC needs much less switches than BCube while the same number of servers can be connected.

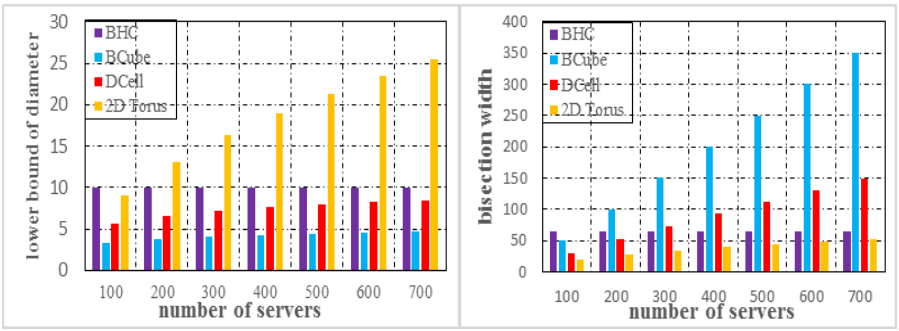

Fig. 7. The comparison of bisection width and diameter in BCube, DCell, 2D Torus and BHC in case that $N = 4$ servers in $BCube_0$ and $DCell_0$ and $P = 8$ and $i = 2$ in BHC_i.

Based on the Theorem 2 and 3, the diameter and bisection width can be figured out with the number of servers increasing. The comparison of bisection width is illustrated by Figure 7.

Figure 7 illustrates the comparison of bisection width in BCube, DCell, 2D Torus and BHC. BHC has the highest bisection width with about 100 servers. However, as network scale growing, the BCube's bisection is the highest and BHC still keeps a fixed value, and is just better than 2D Torus.

The comparison of network diameter illustrates the comparison of diameter in BCube, DCell, 2D Torus and BHC. The diameter of BHC is close to BCube and DCell, but 2D Torus gets worse when the network size grows.

Table 1. Parameters in simulations

Parameter	Value
Traffic pattern	Uniform
Switching mechanism	Wormhole
Packet length(flits)	3
Flit length(bits)	256
Cycle period(ns)	50
Number of Virtual channels	8
Offered load(flits/cycle/node)	0.01~0.4

6.2. Simulations on BHC

In this section, a simulator based on OPNET is built to evaluate the performance of BHC on MapReduce. Every simulated node is configured to use wormhole switching mechanism. The routing algorithm presented in section 4 is implemented in the simulations. We set $P = 4$, $i = 0$ and $K=2$ to build a BHC_2 with 64 master servers and 256 worker servers. Our results quantify two metrics: ETE (End to End) delay and throughput. The ETE delay is the elapsed time (in ns) between the generation of a packet at a source host and its delivery at a destination host. The throughput sum of the data rates (in Gaps) that are delivered to all terminals in a network. The simulation parameters are set as Table 1. Offered load denotes the traffic injection of each node in per cycle.

Performance of different numbers of jobs. The more jobs are injected in the network in the same time, the heavier pressure is exerted on the network. It is essential to test different numbers of jobs injected in the same time on BHC. BHC_0 is the unity to be injected. Hence, BHC_0 is chose as our test unity.

Figure 8 plots the throughput and ETE delay in different numbers of jobs. The single job's saturation point is offered load = 0.2 and the dual jobs' and four jobs' saturation point is about offered load = 0.05, which is about 1/4 of single job's saturation point. This verifies the theoretical value. The results also imply that BHC has a graceful performance even all of the master servers are injected jobs at the same time.

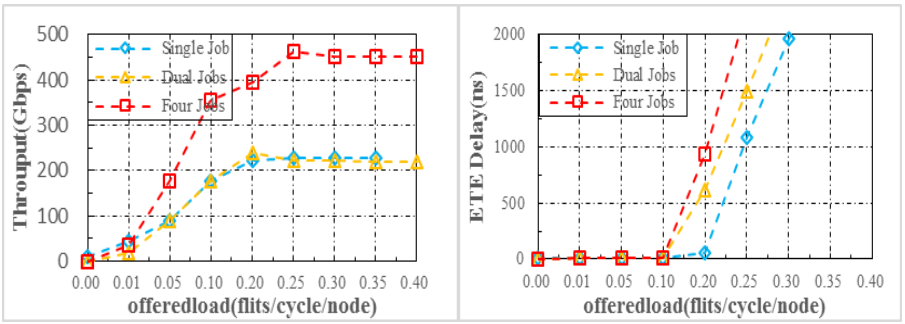

Fig. 8. The comparison of throughput and ETE delay in different numbers of jobs

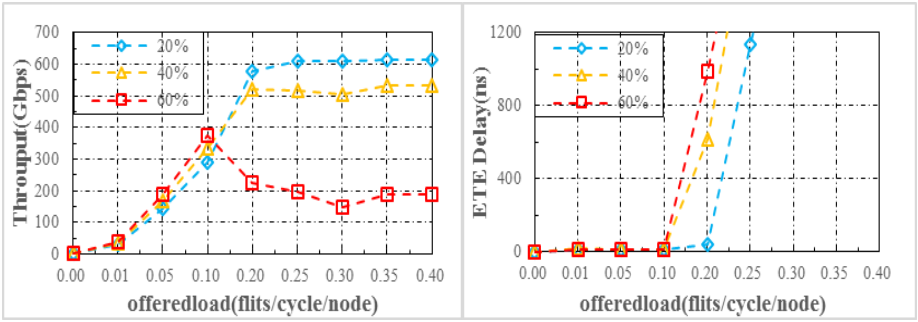

Fig. 9. The comparison of throughput and ETE delay in different *P*

Performance of different probability of communications in worker servers. As proposed in section IV, when a worker server is executing a map or reduce task, it may need the data stored at another worker server. *P* is supposed as the probability that a worker server needs the intermediate data stored at another worker server. If the *P* is getting higher, the more communications will happen in the worker servers. Hence, the simulation results under different *P* can be one of the metrics of worker-to-worker communication performance.

Figure 9 plots the throughput and ETE delay in different *P* of workers servers' communications. With the P growth, the performance of BHC is deteriorating. For example, when offered load is 0.3, the throughput with *P = 20%* is three times of the throughput with *P = 60%*. When *P* is getting higher, it is more probable that the a worker server, executing a map or reduce task, needs the data stored at the another worker server, and this dependency relationship will reduce the efficiency of handling a job, even turn into congestion to deteriorate the performance. Hence, this dependency relationship should be cut down as much as possible in practical, and we can improve the routing algorithm of BHC in future work.

7. Conclusion

Several new DCN architectures have been proposed to improve the topological properties of data centers, however, they do not match well with the specific requirements of some dedicated applications. This paper presents a MapReduce-supported DCN network, named BHC. Through comprehensive analysis and evaluation, BHC is a scalable topology with excellent topological properties and communication performance. It is proven that BHC is competent for MapReduce under different traffic characteristics. The simulation results show that BHC has a graceful performance in multi-job injection. But when the worker servers have a high probability (60% or higher) of dependency relationship, the performance is deteriorating because the efficiency of handling a job is dropping, even resulting in congestion. Hence, this dependency relationship should be cut down as much as possible in practical.

Acknowledgments. This work was supported by the National Science Foundation of China Grant No.61472300, the Fundamental Research Funds for the Central Universities Grant No. JB150318, the 111 Project Grant No.B08038

References

1. Dede, E., Fadika, Z., Govindaraju, M., Ramakrishnan, L.: Benchmarking MapReduce implementations under different application scenarios. Future Generation Computer Systems, 36, 389-399. (2014)
2. Klašnja-Milićević, A., Vesin, B., Ivanović, M., Budimac, Z.: E-Learning personalization based on hybrid recommendation strategy and learning style identification. Computers & Education, 56(3), 885-899. (2011)
3. Zdravkova, K., Ivanović, M., Putnik, Z.: Experience of integrating web 2.0 technologies. Educational Technology Research and Development, 60(2), 361-381. (2012)

4. McKusick, K., Quinlan, S.: GFS: evolution on fast-forward. Communications of the ACM, 53(3), 42-49. (2010)
5. Ghemawat, S., Gobioff, H., Leung, S. T.: The Google file system. In ACM SIGOPS operating systems review (Vol. 37, No. 5, pp. 29-43). ACM. (2003)
6. Burrows, M.: The Chubby lock service for loosely-coupled distributed systems. In Proceedings of the 7th symposium on Operating systems design and implementation (pp. 335-350). USENIX Association. (2006)
7. Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Gruber, R. E.: Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS), 26(2), 4. (2008)
8. Baker, J., Bond, C., Corbett, J. C., Furman, J. J., Khorlin, A., Larson, J., Yushprakh, V.: Megastore: Providing scalable, highly available storage for interactive services. In CIDR (Vol. 11, pp. 223-234). (2011, January)
9. Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., Wilkes, J.: Omega: flexible, scalable schedulers for large compute clusters. In Proceedings of the 8th ACM European Conference on Computer Systems (pp. 351-364). ACM. (2013,)
10. Wang, Y., Lin, D., Li, C., Zhang, J., Liu, P., Hu, C., Zhang, G.: Application Driven Network: providing On-Demand Services for Applications. In Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference (pp. 617-618). ACM. (2016)
11. Xu, K., Qu, Y., Yang, K.: A tutorial on the internet of things: from a heterogeneous network integration perspective. IEEE Network, 30(2), 102-108. (2016)
12. Fehér, P., Asztalos, M., Vajk, T., Mészáros, T., Lengyel, L. Detecting subgraph isomorphism with MapReduce. The Journal of Supercomputing, 1-42.
13. Guo, C., Wu, H., Tan, K., Shi, L., Zhang, Y., Lu, S.: Dcell: a scalable and fault-tolerant network structure for data centers. In ACM SIGCOMM Computer Communication Review (Vol. 38, No. 4, pp. 75-86). ACM. (2008)
14. Li, D., Guo, C., Wu, H., Tan, K., Zhang, Y., Lu, S.: FiConn: Using backup port for server interconnection in data centers. In Infocom 2009, ieee (pp. 2276-2285). IEEE. (2009)
15. Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Lu, S.: BCube: a high performance, server-centric network architecture for modular data centers. ACM SIGCOMM Computer Communication Review, 39(4), 63-74. (2009)
16. Liu, B., Huang, K., Li, J., Zhou, M.: An incremental and distributed inference method for large-scale ontologies based on mapreduce paradigm. IEEE transactions on cybernetics, 45(1), 53-64. (2015)
17. Mohammed, E. A., Far, B. H., Naugler, C.: Applications of the mapreduce programming framework to clinical big data analysis: current landscape and future trends. BioData Mining, 7(1), 22. (2014).
18. Al-Fares, M., Loukissas, A., Vahdat, A.: A scalable, commodity data center network architecture. ACM SIGCOMM Computer Communication Review, 38(4), 63-74. (2008).
19. Ding, M., Tian, H.: PCA-based network Traffic anomaly detection. Tsinghua Science and Technology, 21(5), 500-509. (2016)
20. Kandula, S., Sengupta, S., Greenberg, A., Patel, P., Chaiken, R. The nature of data center traffic: measurements & analysis. ACM SIGCOMM Conference on Internet Measurement Conference (Vol.9, pp.202-208). ACM. (2009).
21. Guo, D., Chen, T., Li, D., Liu, Y.: BCN: Expansible network structures for data centers using hierarchical compound graphs. INFOCOM 2011. IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 10-15 April 2011, Shanghai, China (Vol.21, pp.61-65). DBLP. (2011).
22. Wang, L., Tao, J., Ranjan, R., Marten, H., Streit, A., Chen, J., et al.: G-hadoop: mapreduce across distributed data centers for data-intensive computing. Future Generation Computer Systems, 29(3), 739–750. (2013).

23. Morla, R., Gonçalves, P., Barbosa, J. G.: High-performance network traffic analysis for continuous batch intrusion detection. Journal of Supercomputing, 72(11), 1-22. (2016).
24. Cohen, J.: Graph twiddling in a mapreduce world. Computing in Science & Engineering, 11(4), 29-41. (2009).
25. Fadika, Z., Dede, E., Govindaraju, M., Ramakrishnan, L. Mariane: using mapreduce in hpc environments. Future Generation Computer Systems, 36(3), 379-388. (2014)
26. Slagter, K., Hsu, C. H., Chung, Y. C., Zhang, D.: An improved partitioning mechanism for optimizing massive data analysis using mapreduce. The Journal of Supercomputing, 66(1), 539-555. (2013)
27. Jiang, H., Chen, Y., Qiao, Z., Li, K. C., Ro, W., Gaudiot, J. L.: Accelerating mapreduce framework on multi-gpu systems. Cluster Computing, 17(2), 293-301. (2014)
28. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113. (2008).
29. Kaashoek, F., Morris, R., Mao, Y.: Optimizing mapreduce for multicore architectures. (2010).
30. Yu, Z., Xiang, D., Wang, X.: Balancing virtual channel utilization for deadlock-free routing in torus networks. The Journal of Supercomputing, 71(8), 3094-3115. (2015)
31. Lin, X. Y., Chung, Y. C.: Master–worker model for mapreduce paradigm on the tile64 many-core platform. Future Generation Computer Systems, 36(3), 19–30. (2014).
32. Chen, R., Chen, H., Zang, B.: Tiled-MapReduce: optimizing resource usages of data-parallel applications on multicore with tiling. International Conference on Parallel Architectures and Compilation Techniques (pp.523-534). IEEE Computer Society. (2010).

Tao Jiang received the B.E. degree in Electronics and Communications Engineering from Xidian University in 2015. Now he is doing the M.E. Programme in Telecommunication and information system in the State key lab of ISN, Xidian University. His main research interests are related to optical interconnected networks and data center networks.

Huaxi Gu received B.E., M.E., and Ph.D. in Telecommunication Engineering and Telecommunication and Information Systems from Xidian University, Xidian in 2000, 2003 and 2005 respectively. He is a Full Professor in the State Key Laboratory of ISN, Telecommunication Department, Xidian University, Xidian, China. His current interests include interconnection networks, networks on chip and optical intrachip communication. He has more than 100 publications in refereed journals and conferences. He has been working as a reviewer of IEEE Transaction on Computer, IEEE Transactions on Dependable and Secure Computing, IEEE System Journal, IEEE Communication Letters, Information Sciences, Journal of Supercomputing, Journal of System Architecture, Journal of Parallel and Distributed Computing, Microprocessors and Microsystems etc.

Kun Wang received the B.E. degree and M.E. degree in Computer Science and Technology from Xidian University, Xi'an in 2003 and 2006 respectively. Now she is a lecturer in the Dept. of Computer Science, Xidian University, Xi'an China. Her Current interests include high performance computing and cloud computing, the network vitulization technology.

Xiaoshan Yu received the M.E. degree in Electronics and Communications Engineering from Xidian University in 2013. Now he is doing the Ph.D. Programme in Telecommunication and information system in the State key lab of ISN, Xidian University. His main research interests are related to optical interconnected networks, data center networks.

Yunfeng Lu received the bachelor's degree in Information Engineering from Jilin University in 2016. Now he is doing the M.S. degree in Telecommunication and information system in the State key lab of ISN, Xidian University. His main research interests are related to optical interconnected networks, high performance computing.

Received: February 2, 2017; Accepted: June 15, 2017.