Bao WT, Fu BZ, Chen MY *et al.* A high-performance and cost-efficient interconnection network for high-density servers. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 29(2): 281–292 Mar. 2014. DOI 10.1007/s11390-014-1430-0

A High-Performance and Cost-Efficient Interconnection Network for High-Density Servers

Wen-Tao Bao[1,2] (包雯韬), *Student Member, CCF, ACM, IEEE*
Bin-Zhang Fu[1] (付斌章), *Member, CCF, ACM, IEEE*, Ming-Yu Chen[1,2] (陈明宇), *Member, CCF, ACM, IEEE*
and Li-Xin Zhang[1,2] (张立新), *Member, ACM, IEEE*

[1] *State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences Beijing 100190, China*

[2] *University of Chinese Academy of Sciences, Beijing 100049, China*

E-mail: {baowentao, fubinzhang, cmy, zhanglixin}@ict.ac.cn

Received November 14, 2013; revised January 9, 2014.

Abstract The high-density server is featured as low power, low volume, and high computational density. With the rising use of high-density servers in data-intensive and large-scale web applications, it requires a high-performance and cost-efficient intra-server interconnection network. Most of state-of-the-art high-density servers adopt the fully-connected intra-server network to attain high network performance. Unfortunately, this solution costs too much due to the high degree of nodes. In this paper, we exploit the theoretically optimized Moore graph to interconnect the chips within a server. Accounting for the suitable size of applications, a 50-size Moore graph, called Hoffman-Singleton graph, is adopted. In practice, multiple chips should be integrated onto one processor board, which means that the original graph should be partitioned into homogeneous connected subgraphs. However, the existing partition scheme does not consider above problem and thus generates heterogeneous subgraphs. To address this problem, we propose two equivalent-partition schemes for the Hoffman-Singleton graph. In addition, a logic-based and minimal routing mechanism, which is both time and area efficient, is proposed. Finally, we compare the proposed network architecture with its counterparts, namely the fully-connected, Kautz and Torus networks. The results show that our proposed network can achieve competitive performance as fully-connected network and cost close to Torus.

Keywords high-density server, interconnection network, Moore graph, Hoffman-Singleton graph, equivalent partition

1 Introduction

With the rapid expansion of user data, more and more cloud-sea computing applications have emerged in modern data centers. Those applications are usually classified as high-throughput-computing (HTC) applications[1], and highly concerned with the system performance. In order to serve the HTC applications effectively, the new architecture called high-density server consisting of hundreds or thousands of processor cores has been proposed[2-4]. For instance, one Cray Cascade chassis may consist of 128 Xeon processors (1 280 cores if Xeon E7 processor is adopted)[2], one SeaMicro SM10000 server may consist of 512 cores[5], and each IBM Power 775 drawer contains 256 cores[4].

In general, an HTC system consists of a lot of high-density servers connected by electrical or optical cables. Each high-density server can be in the form of chassis, drawer or cabinet consisting of many processor boards connected by the processor backplane or midplane. In addition, a processor board further consists of several switch chips or SoCs (system-on-chips) and each chip (or SoC) connects some processors. Note that the basic unit (or node) to construct the network of a high-density server is the switch chip (or SoC) and we only focus on the intra-server network. In other words, we will not discuss the way to connect separated servers with electrical or optical cables within this paper.

Moreover, a high-density sever is usually prone to adopt a high-performance intra-server interconnection

Regular Paper
This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA06010401, the National Natural Science Foundation of China under Grant Nos. 61202056, 61331008, 61221062, and the Huawei Research Program of China under Grant No. YBCB2011030.
A preliminary version of the paper was published in the Proceedings of HPCC 2013.
©2014 Springer Science + Business Media, LLC & Science Press, China

network to connect a large number of nodes to improve system performance. For instance, Cray Cascade and IBM Power 775 exploit the fully-connected network[2,4], and SeaMicro SM10000 adopts the 3D-Torus network[5]. Obviously, the fully-connected network could obtain the best performance since every two nodes could communicate with each other at any time without any contention. However, the high performance of fully-connected network is at the expense of its high cost. For example, each node at least needs $(n-1)$ ports to construct the n-node-size fully-connected network. Torus network, on the other hand, is highly scalable since the number of ports is determined by the number of network dimensions, but the growing scale of Torus network will result in long diameter, which will degrade the network performance significantly.

To make a trade-off between cost and performance, we exploit the Moore graph[①] to construct the intra-sever interconnection network. As we will extensively discuss in Section 2, a Moore graph is a regular maximal graph with the given degree d and the diameter k. In other words, given the network size and diameter, the Moore graph will get the minimal degree of a node which will potentially minimize the cost of a router and the network. In this paper, we focus on the Moore graph with diameter $k = 2$. Given the diameter of Moore graph, it is natural to ask that how many nodes should be integrated into one server. As we will discuss in Subsection 3.1, the 50-node-size Moore graph called Hoffman-Singleton graph is the most reasonable choice because it can provide enough parallelism for most HTC applications.

In practice, multiple nodes are usually integrated onto one processor board to further increase the server density. As shown by the Hoffman-Singleton graph, we can integrate 5 or 10 nodes onto one processor board depending on the density of each board and the manufacturing technology. To facilitate the batch production, homogenous boards are expected which means that the subgraphs of the Hoffman-Singleton graph should be identical. Furthermore, the subgraph is required to be connected in case that all other boards break down. Note that a connected board indicates that the nodes in a single board can communicate with each other without the help of other boards.

To make the subgraphs homogenous, two partition schemes are proposed in this paper. The first scheme divides the Hoffman-Singleton graph into ten groups, each of which is a 5-node ring. While the other scheme divides the graph into five groups, each of which is a 10-node Petersen graph[6]. Since the ring $(d = 2, k = 2)$ and Petersen $(d = 3, k = 2)$ graph are both Moore graphs, the network of each board is also optimal. Finally, for each partition scheme, a logic-based routing algorithm is proposed.

The main contributions of the paper are as follows:

1) The Moore graph is discussed and utilized to reduce the cost of intra-server network without degrading the network performance.

2) Two partition schemes of Hoffman-Singleton graph are proposed, making the whole solution practical.

3) Two deadlock-free routing algorithms are proposed to fully utilize the path diversity of the network.

4) The traffic characteristics of five typical cloud-sea applications are analyzed in this paper.

The rest of this paper is organized as follows. Section 2 discusses the intra-server networks of high-density servers as well as the Moore graph. Section 3 discusses the network architecture and the two partition schemes. Section 4 shows the two routing algorithms. Section 5 discusses the traffic characteristics of cloud-sea applications. Section 6 compares the proposed network with its counterparts through simulations and analyses. Finally Section 7 concludes this paper.

2 Background

In this section we will discuss the interconnection networks of current high-density servers and then introduce three well-known Moore graphs which are closely related to our work.

2.1 Interconnection Networks of High-Density Servers

First, note that this paper refers to the compute unit, whose components are not connected by cables, as a "high-density" server. For example, a Cray Cascade chassis and a Power 775 drawer are both the "high-density" servers. As mentioned, a high-density server may contain hundreds of processors. The network connecting these processors is called the intra-server interconnection network.

Investigations indicate that most of current high-density servers adopt fully-connected intra-server network such as Cray Cascade and Power 775. In Cray Cascade, as shown in Fig.1(a), the chassis backplane connects 16 blades, each blade contains four nodes, and each node contains two Xeon processors, thus each chassis contains 128 Xeon processors. Since it adopts fully-connected network to connect 16 blades, every two blades in one chassis can communicate with each other directly. In Power 775, as shown in Fig.1(b), the su-

[①] Moore graph. http://en.wikipedia.org/wiki/Moore_graph, Dec. 2013.

pernode can contain up to four drawers, each of which contains up to eight compute nodes. Similarly, the fully-connected network is used to connect compute nodes including both intra- and inter- drawers.

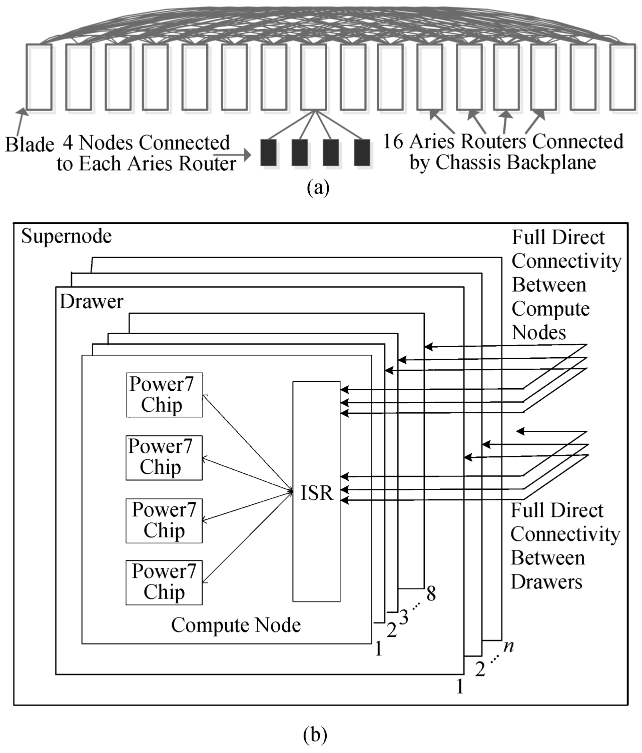

Blade
4 Nodes Connected to Each Aries Router
16 Aries Routers Connected by Chassis Backplane
(a)
Supernode
Full Direct Connectivity Between Compute Nodes
Drawer
Power7 Chip
Power7 Chip
Power7 Chip
Power7 Chip
ISR
Full Direct Connectivity Between Drawers
Compute Node
1
2
3
8
1
2
n
(b)

Fig.1. Architecture of high-density servers. (a) Architecture of a chassis of Cray Cascade system. (b) Architecture of a supernode of IBM Power 775 system.

2.2 Moore Graph

In this subsection we will discuss three typical Moore graphs closely related to our proposed network. The first graph called ring consists of five nodes. Both of the degree and diameter of the 5-node ring are 2. The second one with degree 3 and diameter 2 is called Petersen graph. As is shown in Fig.2(a), a Petersen graph can be formed by embedding a five-point star into a ring. The last one is the Hoffman-Singleton graph as shown in Fig.2(b), which has 50 nodes with degree 7 and diameter 2.

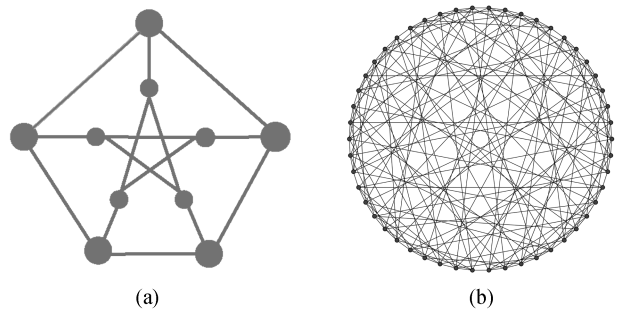


Fig.2. Examples of Moore graphs. (a) Petersen graph. (b) Hoffman-Singleton graph.

As shown in Fig.3, a Hoffman-Singleton graph can be formed by stars and rings. The five-point star is represented by P_j (with $0 \leqslant j \leqslant 4$) and ring is represented by Q_k (with $0 \leqslant k \leqslant 4$). To form the Hoffman-Singleton graph, every node in the five-point star should be connected to one node of every ring. Mathematically, node $P_{j,i}$ should be connected to node $Q_{k,q}$, where $q = (i + j \times k) \bmod 5$. For example, node $P_{1,0}$ should be connected to node $Q_{1,1}$ since $i = 0$, $j = k = 1$ and $q = (0 + 1 \times 1) \bmod 5$.

3 Proposed Network Architecture

In this section, we will first answer the question that why the Hoffman-Singleton graph instead of others should be selected. Secondly, the equivalent-partition schemes, which are used to address the batch production problem, are discussed.

3.1 Why Hoffman-Singleton

Although the network diameter is not the only factor that determines the network performance, we still want to keep it as small as possible to reduce the network

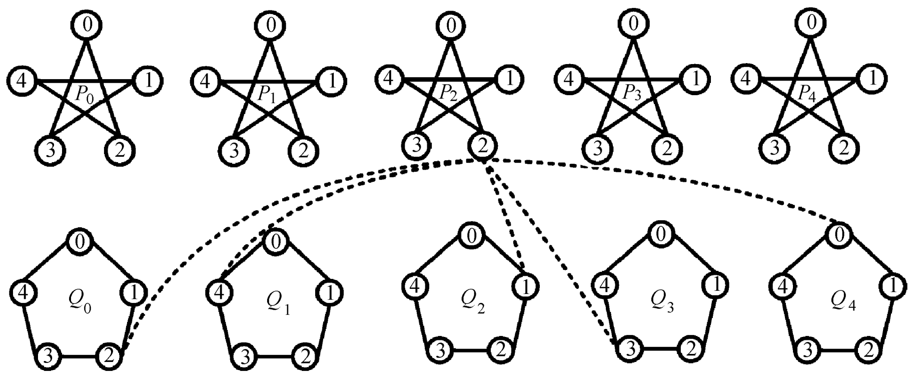


Fig.3. Traditional way to form a Hoffman-Singleton graph.

latency. Since the cost of fully-connected network with diameter 1 is too high, we decide to design an intra-server network with diameter 2.

Once the network diameter is determined, the next job is to determine the network size. In general, the size is expected to be large enough to provide sufficient parallelism for most applications. Therefore, one application is usually mapped onto a single high-density server to fully utilize the benefits of the high-performance intra-server network. Otherwise, inter-server network with relatively low performance may degrade the application performance.

Generally, the parallelism is limited by the Amdahl's law. If P is the portion of an application that can be made parallel, N is the total number of threads, then the speedup of parallelization can be presented as (1). Obviously, given P, the speedup will not increase (noticeably) once N is large enough.

$$\mathit{Speedup} = \frac{1}{(1-P)+P/N}. \tag{1}$$

From this equation, we could find that 8 192 threads are enough for most applications, even for those with $P = 0.99$, i.e., 99% of the applications can be made parallel. Note that we do not consider the applications with $P = 1$, because we can divide those applications, which can be 100% made parallel, into several separated sub-applications. We consider the case that there are many-core processors already taped out, such as Intel's 80-core processor[7] and Tilera's Tile64 processor[8]. Since there may be multiple threads per processor core, it is possible to provide hundreds of threads by a processor. Assuming one many-core processor could provide 160 threads (80 cores × 2 threads per core), it requires 51.2 processors to provide 8 192 threads. Therefore, we choose the Hoffman-Singleton graph with 50 nodes as the intra-server network. We believe that future many-core processors could provide more parallelism, thus integrating 50 nodes into a server will be enough for future applications. Otherwise, if the Petersen graph is chosen, then the total number of threads will be 1 600 which is not enough for applications with $P \geqslant 0.97$.

3.2 Partition Schemes

To facilitate the batch production, two partition schemes are proposed in this subsection. Based on the original construction[②] of the Hoffman-Singleton graph, the first scheme is simply to transform each P_j into a ring and the nodes of P_j are re-indexed from 0 to 4 clockwise. After rearrangement, the ten groups are identical since P_j is the same with Q_k. Thus, for every two nodes in P_j, their relative position has been changed as shown in Fig.4.

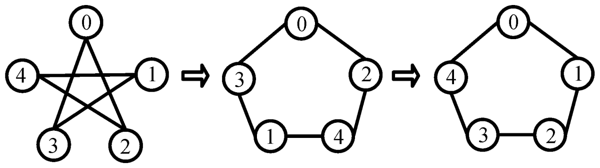


Fig.4. Transforming a five-point star into a ring.

Additionally, the connection rule between P and Q also should be changed: for the node i in a P_j, the only adjacent node it has in Q_k is the node labeled $((i + \lfloor \frac{j}{2} \rfloor \times \lfloor \frac{k}{2} \rfloor) \times 3) \bmod 5$. The whole construction is shown in Fig.5. To better understand the specific architecture, we present it in Fig.6. There are ten processor boards (labeled Blade_0 to Blade_9) on the processor backplane and each processor board contains five nodes forming a ring. One possible mapping from P_i and Q_j ($0 \leqslant i, j \leqslant 4$) to Blade_k ($0 \leqslant k \leqslant 9$) is that $\mathit{Blade}_k = P_{k/2}$ if $k\%2 = 0$, otherwise $\mathit{Blade}_k = Q_{k/2}$. In this way, Blade_i is connected to Blade_j, where $i\%2 \neq j\%2$. The second scheme is to divide the graph into five groups, and each group forms a Petersen graph

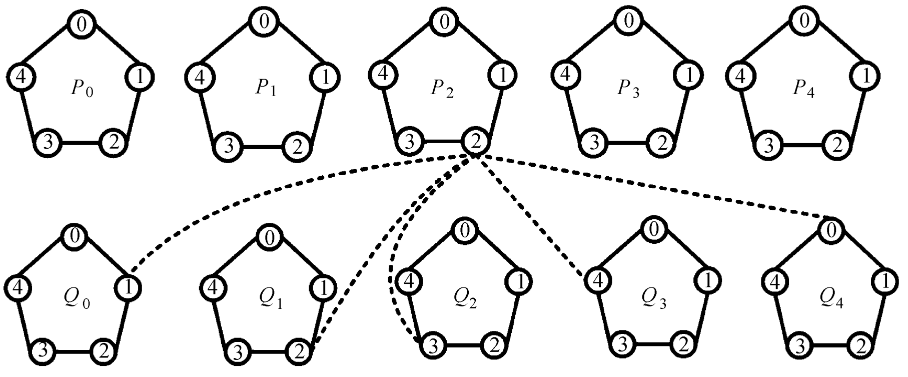


Fig.5. Way to form a Hoffman-Singleton graph with rings.

[②]Hoffman-Singleton graph. http://mathworld.wolfram.com/Hoffman-SingletonGraph.html, Dec. 2013.

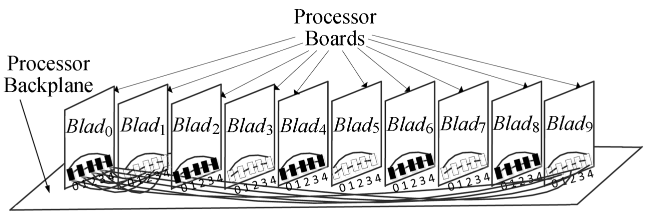


Fig.6. Architecture of the high-density server consisting of 10 ring-connected boards.

as shown in Fig.2(a). Unlike the first partition scheme, we combine P_i with Q_i ($0 \leqslant i \leqslant 4$) to form the Petersen graph (labeled T_i) as shown in Fig.7. Obviously, T_0, T_1 (or T_4) and T_2 (or T_3) are heterogeneous because the relative position of every two nodes in each group is different. For example, node 0 of the outer ring is adjacent to node 0 of the inner star in T_0 while adjacent to node 4 of the inner star in T_1. Thus it is necessary to re-index the nodes to keep all Peterten graphs the same with T_0. Furthermore, in order to simplify the routing algorithm, we should index the nodes in a processor board T_i with 0 to 9 where the nodes in the outer ring and inner star labeled with 0 to 4 and 5 to 9 respectively.

Since their relative positions have been changed, similar to the first scheme, we create a new connection rule to connect different groups that is proposed as follows: Generally, a node $T_{j,i}$ is connected to the node $T_{k,q}$, where $q = (i + (j - k) \times k) \bmod 5 + 5$, if $i \in [0, 4]$ and $j > k$; otherwise, it is connected to $T_{k,m}$, where $m = (i - 5 - (k - j) \times j) \bmod 5$ if $i \in [5, 9]$ and $j > k$. The whole construction is shown in Fig.8. We also present the specific architecture as shown in Fig.9. There are five processor boards (labeled $Blade_0$ to $Blade_4$) in the processor backplane. Each processor board $Blade_i$ contains ten nodes (labeled 0 to 9) forming a Petersen graph. One possible mapping from T_i to $Blade_j$ is that $Blade_i = T_i$.

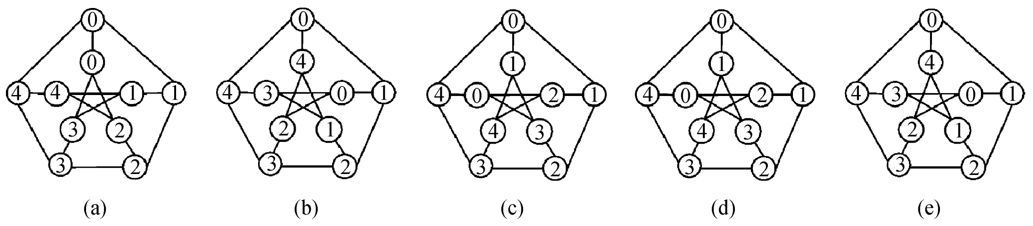


Fig.7. Combinations of P_i and Q_i. (a) T_0. (b) T_1. (c) T_2. (d) T_3. (e) T_4.

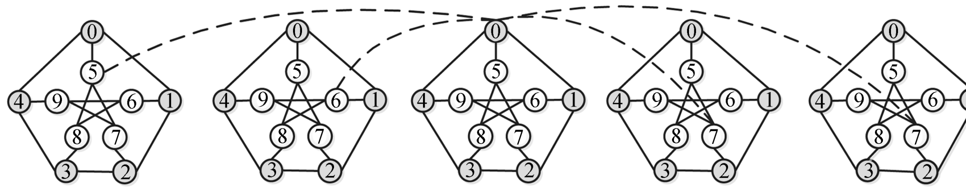


Fig.8. Way to form a Hoffman-Singleton graph with Petersen graphs.

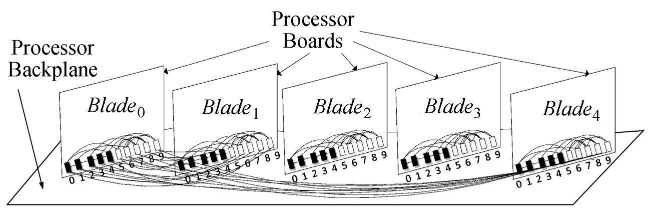


Fig.9. Architecture of the high-density server consisting of 5 Petersen-graph-connected boards.

The characteristics of the two partition schemes are concluded as follows. First, the homogenous partition facilitates the batch production. Second, either the ring or the Petersen graph is proven to be a Moore graph. Thus, the network of each processor board is also optimal. Third, we can expand the processor board one by one without any re-cabling operation or degradation of network performance.

4 Routing Mechanisms

In general, there are two popular ways to implement routing algorithms: table-based solution and logic-based solution. Table-based implementation is flexible, but may cause large space and time overhead. Thus we adopt the logic-based solution for the two partition schemes.

4.1 R_{ring}: Routing Algorithm for Ring-Based Partition

Note that the ring-based partition solution has two significant characteristics. First, nodes in P_i (Q_i) are not connected to any nodes in P_j (Q_j), where $i \neq j$. Second, every node in P_i (or Q_i) has a connection with only one node in Q_j (or P_j).

The main idea of this routing algorithm is to find the shortest route. Given the source and destination node, there are three possible cases that should be discussed.

1) They are in the same processor board.

2) The source node is in P_j (or Q_j) and the destination node is in P_k (or Q_k), where $j \neq k$.

3) The source node is in P_j (or Q_j) and the destination node is in Q_k (or P_k).

For case 1, we only need to forward the packet to the corresponding port according to the routing algorithm inside the processor board.

For case 2, the packet must be forwarded to an intermediate node. We denote an intermediate node as $Blade_{e,f}$ where e represents the group number and f represents the node number inside this group. Similarly, we denote the source node as $Blade_{a,b}$ and the destination node as $Blade_{c,d}$. Since a, b, c and d are already known, we use them to calculate e and f. We first calculate e according to the parity of a where even represents that the source node is in $P_{\frac{a}{2}}$, or vice versa in $Q_{\frac{a}{2}}$. If a is even, e is calculated as (2) where v is variable, and then $f = ((b + \lfloor\frac{a}{2}\rfloor \times \lfloor\frac{e}{2}\rfloor) \times 3) \bmod 5$. Otherwise, e is calculated as (3) where v is variable and $f = ((2 \times b - 6 \times \lfloor\frac{a}{2}\rfloor \times \lfloor\frac{e}{2}\rfloor)) \bmod 5$. Now we have finished finding the intermediate node. Note that the way to choose the variable v is determined by the sign of the denominator. If the denominator is negative (or positive), v will minus 1 (or add 1) from 0, and then we will choose the value that first makes the numerator divisible by the denominator.

$$e = \frac{(d - b + 5 \times v) \times 2}{\left\lfloor\frac{a}{2}\right\rfloor - \left\lfloor\frac{c}{2}\right\rfloor + 1}, \tag{2}$$

$$e = \frac{(d - b + 5 \times v) \times 2}{3 \times \left(\left\lfloor\frac{c}{2}\right\rfloor - \left\lfloor\frac{a}{2}\right\rfloor\right)}. \tag{3}$$

For case 3, the source and destination nodes may not be adjacent, because every node in P_j is connected to only one node in Q_k directly. Therefore there are also three cases as shown in Fig.10.

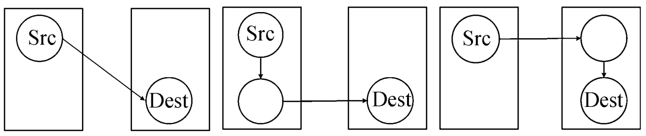


Fig.10. Routing examples based on different distributions of the source node (Scr) and destination node (Dest).

1) The source and the destination nodes are adjacent.

2) The source node first forwards the packet to the intermediate node in its own processor board.

3) The source node first forwards the packet to the intermediate node in the processor board where the destination node is.

For case 1, the source node forwards the packet to the destination directly. However, as for the following cases, we should find the processor board where the intermediate node locates. Since the diameter of the graph is 2, the intermediate node is adjacent to either the source node or the destination node.

Without loss of generality, we assume that it belongs to case 3 where the intermediate node is adjacent to the destination node. So we should find out the intermediate node which is not only adjacent to the source node but also in the processor board where the destination node locates. First, we denote $Blade_{a,b}$, $Blade_{c,g}$ and $Blade_{c,d}$ as the source node, the intermediate node and the destination node respectively. We need to calculate g with given values of a, b and c. The calculation is also discussed with two categories according to the parity of a. If a is even, g is calculated as (4). Otherwise, g is calculated as (5).

$$g = \left(\left(b + \left\lfloor\frac{a}{2}\right\rfloor \times \left\lfloor\frac{c}{2}\right\rfloor\right) \times 3\right) \bmod 5, \tag{4}$$

$$g = \left(2 \times b - 6 \times \left\lfloor\frac{a}{2}\right\rfloor \times \left\lfloor\frac{c}{2}\right\rfloor\right) \bmod 5. \tag{5}$$

If $Blade_{c,g}$ is adjacent to $Blade_{c,d}$, the intermediate node is $Blade_{c,g}$. Otherwise, we can infer that it must belong to case 2. Therefore, the intermediate node must be in the processor board where the source node is and is adjacent to the destination node. Hence, we can use the equation above to find the intermediate node by simply replacing a with c, b with d, c with a, because the source node is $Blade_{c,d}$, the intermediate node is $Blade_{a,g}$ and the destination node is $Blade_{a,b}$. Note that all cases are checked in parallel by hardware. The basic routing procedure is shown in Fig.11.

4.2 R_{Peterson}: Routing Algorithm for the Petersen-Based Partition

The basic idea of R_{Peterson} is the same with that of R_{ring}, but we utilize the R_{InnerP} algorithm[9] to route packets within each processor board. As shown in Fig.12, the procedure of R_{Peterson} is the same with that of R_{ring} except that $Blade_{e,f}$, $Blade_{c,g}$ and $Blade_{a,g}$ are calculated in a different way.

Particularly, to determine $Blade_{e,f}$, we can use (6) to get e.

```
1:  **Require**: source node: $Blade_{a,b}$; destination node: $Blade_{c,d}$
2:    **if** $Blade_{a,b} == Blade_{c,d}$ **then**
3:        Consume the packet
4:    **else**
5:        **if** $a == c$ **then**
6:            Call $R_{\text{basic\_unit}}$
7:        **else**
8:            **if** $a\%2 == c\%2$ **then**
9:                Find the middle node $Blade_{e,f}$
10:           **else**
11:               **if** $Blade_{a,b}$ and $Blade_{c,d}$ are neighbours **then**
12:                   Forward the packet to $Blade_{c,d}$ directly
13:               **else**
14:                   Calculate $Blade_{c,g}$ adjacent to $Blade_{a,b}$
15:                   **if** $Blade_{c,g}$ is adjacent to $Blade_{c,d}$ **then**
16:                       Forward the packet to $Blade_{c,g}$
17:                   **else**
18:                       Calculate $Blade_{a,g}$ adjacent to $Blade_{c,d}$
19:                       Forward the packet to $Blade_{a,g}$
20:                   **end if**
21:               **end if**
22            **end if**
23        **end if**
24:   **end if**
```

Fig.11. Routing algorithm for the ring-based partition.

```
1:  **Require**: source node: $Blade_{a,b}$; destination node: $Blade_{c,d}$
2:    **if** $Blade_{a,b} == Blade_{c,d}$ **then**
3:        Consume the packet
4:    **else**
5:        **if** $a == c$ **then**
6:            Call $R_{\text{Inner}P}$
7:        **else**
8:            **if** ($b > 4$ and $c > 4$) or ($b < 5$ and $c < 5$) **then**
9:                Find the middle node $Blade_{e,f}$
10:           **else**
11:               **if** $Blade_{a,b}$ and $Blade_{c,d}$ are neighbours **then**
12:                   Forward the packet to $Blade_{c,d}$ directly
13:               **else**
14:                   Calculate $Blade_{c,g}$ adjacent to $Blade_{a,b}$
15:                   **if** $Blade_{c,g}$ is adjacent to $Blade_{c,d}$ **then**
16:                       Forward the packet to $Blade_{c,g}$
17:                   **else**
18:                       Calculate $Blade_{a,g}$ adjacent to $Blade_{c,d}$
19:                       Forward the packet to $Blade_{a,g}$
20:                   **end if**
21:               **end if**
22:           **end if**
23:       **end if**
24:   **end if**
```

Fig.12. Routing algorithm for the petersen-based partition.

$$\begin{cases} \dfrac{d-b+5\times v}{a-c}, & b \geqslant 5, \\ \dfrac{(b-d+5\times v)}{a-c}+a+c, & b < 5. \end{cases} \tag{6}$$

Likewise, to determine $Blade_{c,g}$, we use (7) to calculate g.

$$\begin{cases} (b-5-(c-a)\times a) \bmod 5, & b \geqslant 5, \\ (b+(a-c)\times c) \bmod 5+5, & b < 5. \end{cases} \tag{7}$$

As for calculating $Blade_{a,g}$, we also use (7) but replace a with c, b with d, c with a.

5 Network Traffic Characteristics

In this section, we discuss the traffic characteristics of cloud-sea applications, which have a great influence on the simulation and evaluation of our proposed network. Particularly, by analyzing the following five typical applications, we want to answer the following questions in this section: 1) what are the synthetic traffic patterns? 2) what are the aggregate characteristics of packet size?

The five applications include the web search[10] and four Hadoop applications: TeraSort[11] (big data application), K-Means[12] (data mining application), Join and Aggregation[13] (data base application). As for the web search application, a Faban server is utilized to simulate the real world clients, a Tomcat server to simulate the frontend, and a Nutch server to simulate the search engine. We deploy the Hadoop applications on a cluster with one master and eight slaves. The version of Hadoop and JDK is 1.0.2 and 1.6.0, respectively. Particularly, each slave node has 16 map task slots and 12 reduce task slots, and 2 GB Java heap is assigned to each map and reduce task. The size of dataset of TeraSort, Hive, and K-Means is 100 GB, 150 GB and 150 GB, respectively.

As for the synthetic traffic patterns, all four Hadoop applications generate the all-to-all traffic pattern among the slave nodes since most of the data is exchanged during the shuffle phase. Taking the TeraSort as an example, Fig.13 shows that each slave node communicates with all other slave nodes with a similar possibility. Node 4 and node 5 receive packets with a little higher possibility due to the uneven distribution of data. Join, Aggregation and K-Means generate the similar traffic pattern, but the synthetic traffic patterns are omitted due to the limited space. As for the web search application, one-to-many or many-to-many traffic pattern is generated depending on whether multiple web servers are utilized to balance traffic load[10].

Beside the synthetic traffic pattern, the size of packets also strongly affects the network design. Large packets help to improve the utilization of bandwidth, but they also increase the possibility of congestion. Small packets, on the other hand, are difficult to fill the router pipeline. Thus, advanced techniques, such as sophisticated arbiters and speedup techniques, are required to achieve the idea throughput. As shown in Fig.14, the packet sizes of all Hadoop applications exhibit a similar pattern, where about 20%∼30% of packets are small ones, 70%∼80% are MTU packets, and other size packets are very few. However, for the web search application, 80% of the packets are smaller than 600 bytes.

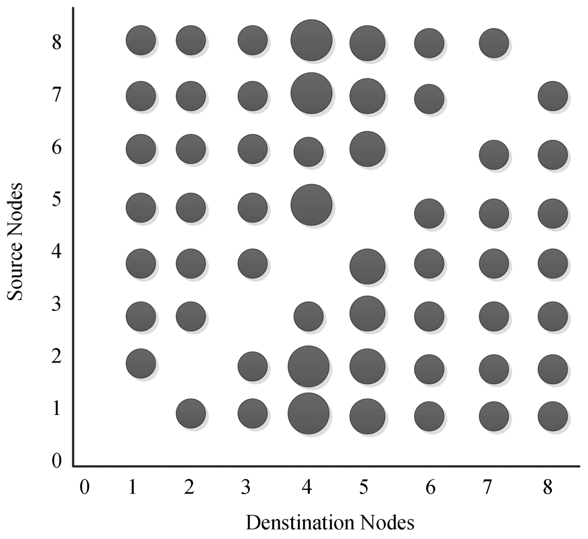


Fig.13. Traffic matrix of TeraSort. The y-axis represents the source node, x-axis represents the destination node, the size of balls represents the possibility of communication, 0 is the master node and 1∼8 are the slave nodes.

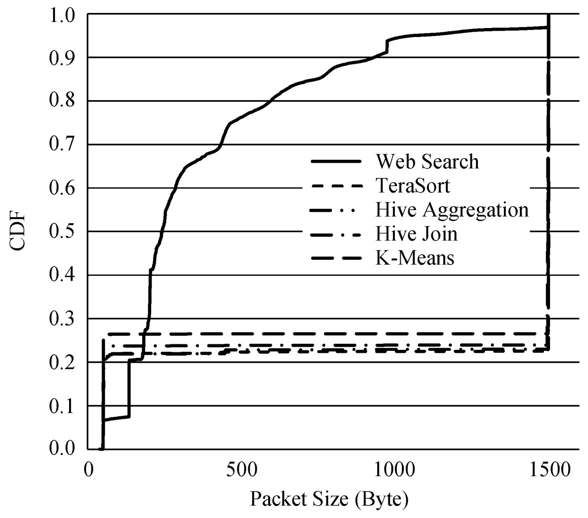


Fig.14. CDF of distribution of the size of packets.

6 Simulation and Analysis

So far we have already proposed the two different partition schemes and the corresponding routing mechanisms for Hoffman-Singleton network. In the reminder of this paper, we compare the performance of Hoffman-Singleton network with its counterparts, namely the fully-connected③, Kautz④ and 3D-Torus[14] networks. All networks are simulated with an event-driven simulator, namely the BookSim simulator[15].

For fair comparison, we should make the configurations of all networks as same as possible. In our experiment, the simulation parameters are shown in Table 1. It can be inferred that all of these networks' sizes are 50 except Kautz, whose size is determined by $d^{n-1} + d^n$, where d is the degree and n is the diameter[16]. Therefore, we choose $K(7, 2)$ and the resultant network size is 56. We believe that the small difference does not affect the simulation results. For 3D-Torus, in order to get the same network size, we choose the one with every dimension $(x, y, z) = (5, 5, 2)$. To equalize the capability of storing and forwarding in every node, the same number of buffers is allocated to each input port. In the end, internal speedup of 2 is chosen to achieve a high fraction of the ideal throughput.

Table 1. Simulation Parameters

Topology	Network Size	Number of Virtual Channels	Virtual Buffer Size	Internal Speedup
Fully-connected	50	1	128	2.0
Hoffman-Singleton	50	2	64	2.0
Kautz	56	2	64	2.0
Torus	50	2	64	2.0

According to the extensive analysis of traffic characteristics in Section 5, we conduct the experiments under two environments: different synthetic traffic patterns and different cloud-sea applications. As for the first environment, we consider three synthetic typical traffic patterns, uniform, shuffle, and transpose. On the other hand, as for the second environment, we use the new distribution of packet size instead of constant packet size to mimic these real cloud-sea applications and adopt uniform to simulate the all-to-all communication.

6.1 Throughput

The throughput comparisons are shown in Fig.15 and Fig.16 where x-axis represents the injection rate and y-axis represents the throughput in the number of flits accepted per cycle per node.

As shown in Fig.15, it is obviously that different traffic patterns have significantly different influences on the

③Complete graph. http://en.wikipedia.org/wiki/Complete_graph, Dec. 2013.

④The Kautz graph. http://pl.atyp.us/wordpress/?p=1275, 2007, Dec. 2013.

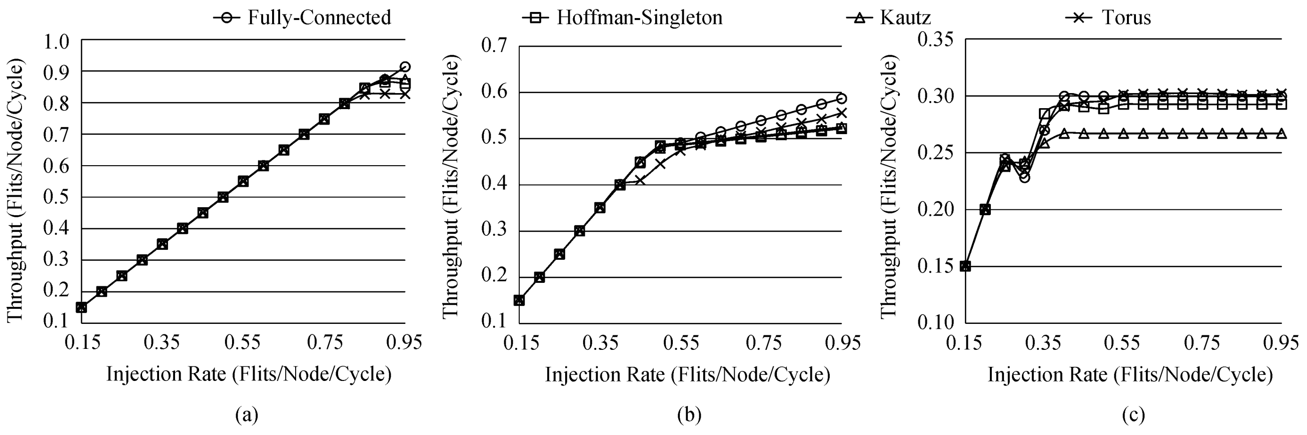


Fig.15. Network throughput under synthetic traffic patterns. (a) Uniform. (b) Shuffle. (c) Transpose.

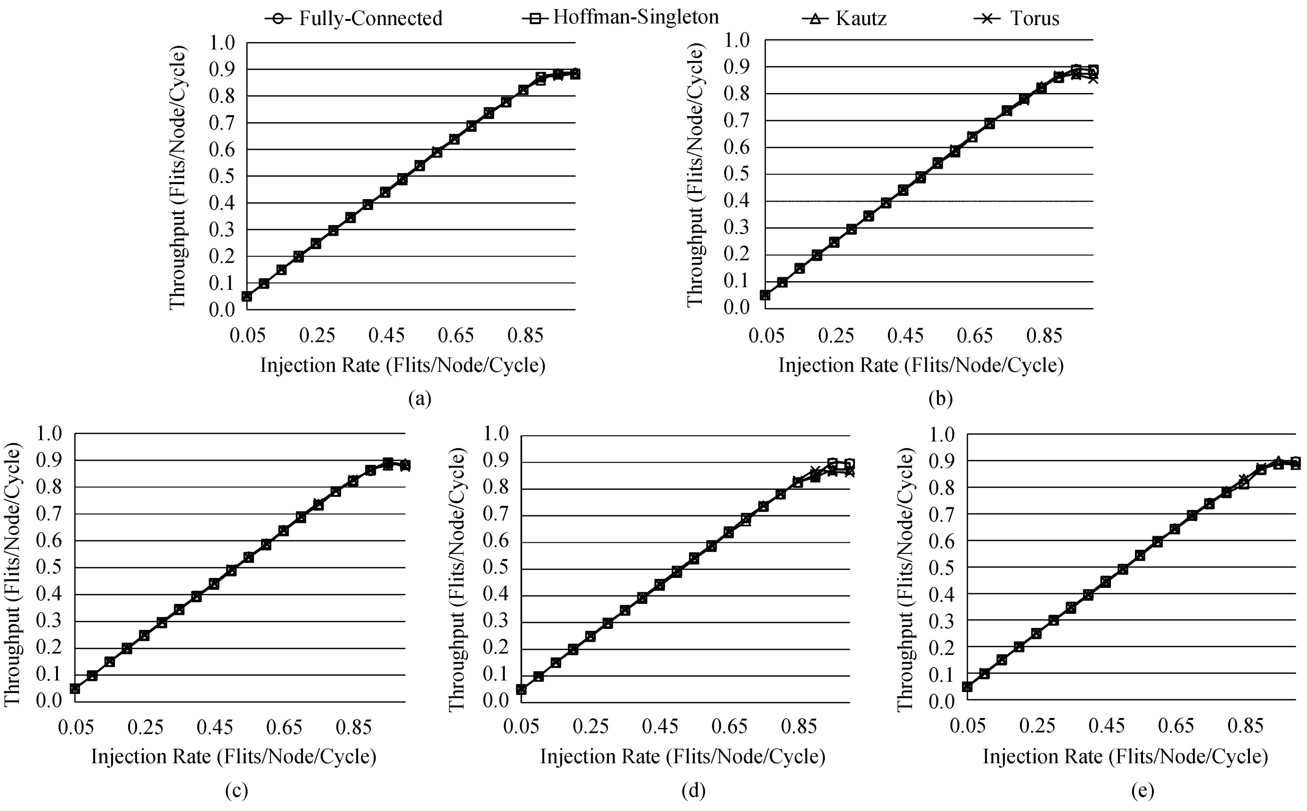


Fig.16. Network throughput under 5 cloud-sea applications. (a) TeraSort. (b) Aggregation. (c) Join. (d) K-Means. (e) Web search.

performance of network. First, regardless of the network, the uniform traffic always shows the best performance, while the transpose traffic always shows the worst performance. Secondly, the fully-connected network performs the best regardless of the traffic pattern, because every node is connected to each other directly. Thirdly, although the Hoffman-Singleton network performs worse than the fully-connected network, it shows relatively good performance.

As shown in Fig.15(a), the differences come to occur when the injection rate is larger than 0.85: the Hoffman-Singleton network performs better than Torus and shows the similar result with Kautz under the uniform traffic pattern. As shown in Fig.15(b), the Hoffman-Singleton network performs slightly worse than Tours but similar with Kautz under the shuffle traffic pattern. As for the transpose traffic pattern in Fig.15(c), Kautz performs the worst while the other networks show similar performance. In summary, the fully-connected network has the best performance while the Hoffman-Singleton network is second to it from an overall perspective.

Furthermore, we use the traffic patterns of five typical cloud-sea applications to evaluate the networks. Fig.16 shows similar simulation results under the five different applications. That is because the distribution of packet size of different applications has no significant difference. From these results, we can clearly find that all networks perform similarly with injection rate increasing. Only when the injection rate exceeds 0.85, some slight differences occur. We can find that the throughput of Tours is slightly worse than others while the fully-connected network performs the best. Furthermore, the performance of the Hoffman-Singleton and Kautz networks is almost consistent.

6.2 Latency

Similar to the analysis of throughput, we firstly compare the simulation results under different traffic patterns regardless of the specific network. The fully-connected network performs the best because its latency always increases more slowly than the other networks. This phenomenon is reasonable since the diameter of the fully-connected network is 1 that is shorter than the others. Now, we focus on the comparison of different networks under same synthetic traffic patterns. As for the result of the uniform traffic pattern shown in Fig.17(a), the average latency of the fully-connected network remains below 20 cycles until the injection rate exceeds 0.9. For the Torus network, the latency increases rapidly when the injection rate exceeds 0.8. In addition, the curves of the Hoffman-Singleton network and Kautz network almost overlap and between the other two networks' curves. As for the comparison of the shuffle traffic shown in Fig.17(b), Torus performs the worst since it has the longest diameter, while the other networks are nearly consistent. Lastly, all networks show poor performance with slight differences as shown in Fig.17(c). Their latency tends to go up once injection rate merely exceeds 0.25. The Hoffman-Singleton network performs better than Kautz but worse than the fully-connected and Torus networks. In the end, we can conclude that the fully-connected network performs the best and the Hoffman-Singleton network is only second to it from an overall perspective.

As shown in the comparison of throughput for the five typical applications above, there are no significant differences between these graphs in Fig.18. Thus we only need to have an explicit description about the results as shown in Fig.18(a). We can find that all curves have the same tendency with a slight difference. It should be pointed out that the curves of the Hoffman-Singleton network and Kautz network are almost overlapped. On the other hand, the latency of Torus is slightly higher than the others while the fully-connected network has the lowest latency. These results are mainly due to the length of diameter where Torus's diameter is much longer than the others' and the fully-connected network has the shortest diameter.

6.3 Cost

In order to get a visual representation of the cost, we simply take the number of links and ports of every node into consideration. As shown in Table 2, the cost of the fully-connected network is extremely high since its link number and port number are both seven times of those of the Hoffman-Singleton network. High cost makes the fully-connected network not acceptable even though it shows extremely good performance on throughput and latency. On the contrary, Torus costs the least while its performance is the worst compared with the others. To balance the performance and the cost, the Hoffman-Singleton network and Kautz network are the most desirable candidates since their performance is relatively good and the cost is not very high. However, in terms of cost of them, the cost of the Kautz network is four times of that of the Hoffman-Singleton network. Considering every aspect, the Hoffman-Singleton network is the most suitable one that can be applied to the intra-server interconnection network of high-density servers.

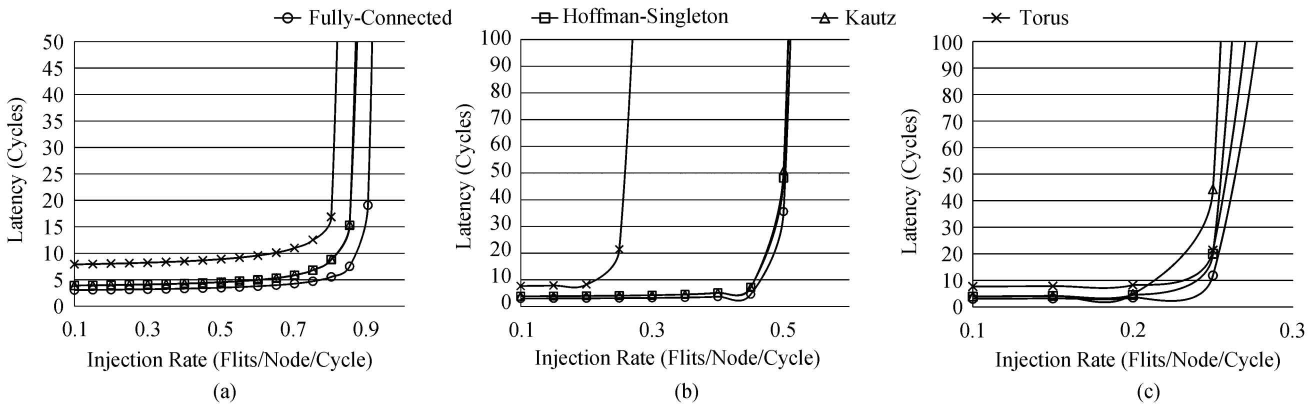


Fig.17. Average packet latency under synthetic traffic patterns. (a) Uniform. (b) Shuffle. (c) Transpose.

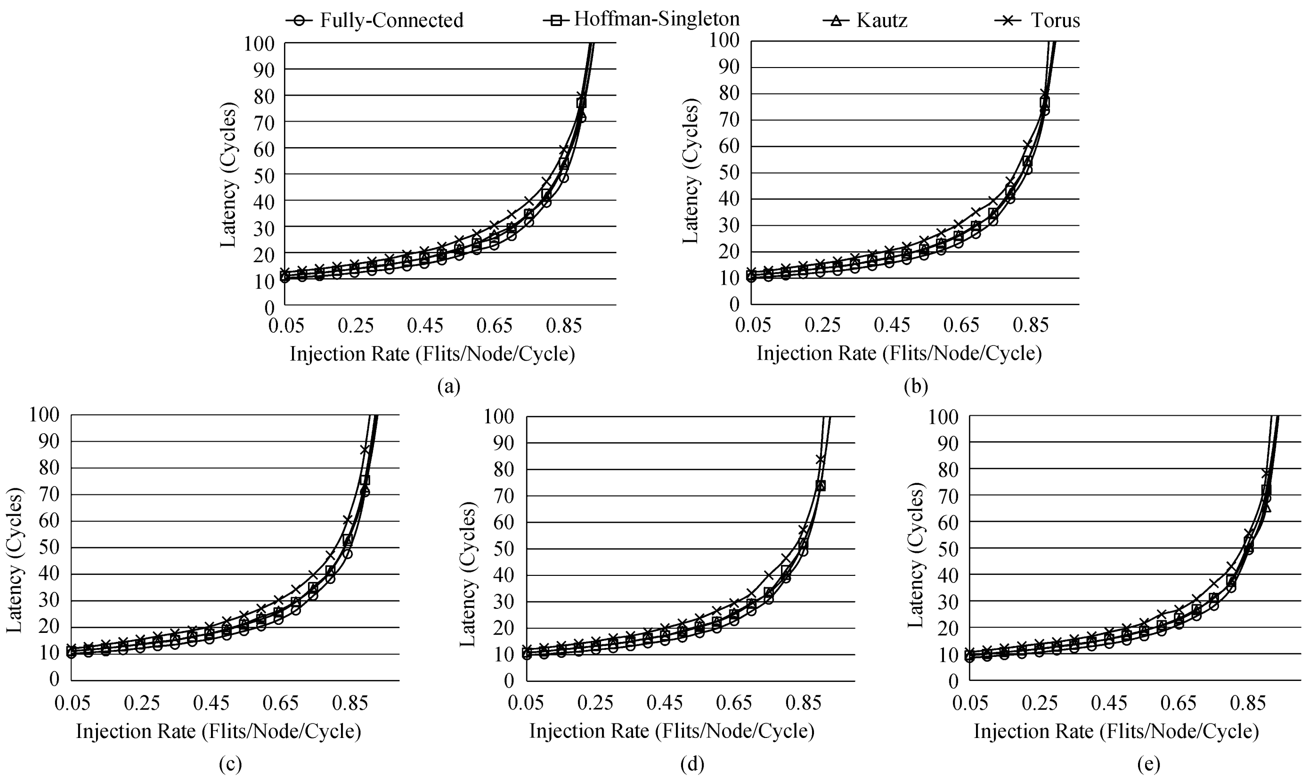


Fig.18. Average packet latency under 5 cloud-sea applications. (a) TeraSort. (b) Aggregation. (c) Join. (d) K-Means. (e) Web search.

Table 2. Cost Comparison

Topology	Number of Links	Number of Ports
Fully-connected	1 225	49
Hoffman-Singleton	175	7
Kautz	784	14
Torus	150	6

7 Conclusions

The intra-server interconnection network largely determines the performance and the cost of high-density servers. Inspired by the Moore graph theory, we exploited the Hoffman-Singleton graph to construct an intra-server interconnection network in this paper. To facilitate the batch production, the Hoffman-Singleton graph should be equivalently divided into connected subgraphs. To address this problem, two partition schemes were presented. Furthermore, two routing mechanisms were proposed for the two corresponding partition schemes. Finally, a series of experiments were taken to compare our proposed network architecture with its counterparts, namely the fully-connected, Kautz and Torus networks. The simulation results show that our proposed network could attain competitive performance as the fully-connected network and cost close to Torus. Therefore, the proposed network architecture is quite suitable for modern and future high-density servers.

References

[1] Montero R S, Huedo E, Llorente I M. Benchmarking of high throughput computing applications on grids. *Parallel Computing*, 2006, 32(4): 267-279.

[2] Faanes G, Bataineh A, Roweth D, Court T, Froese E, Alverson B, Johnson T, Kopnick J, Higgins M, Reinhard J. Cray cascade: A scalable HPC system based on a Dragonfly network. In *Proc. the International Conference for High Performance Computing, Networking, Storage and Analysis* (*SC2012*), November 2012, Article No.103.

[3] Rao A. SeaMicro technology overview. Technical Report, AMD, January 2012. http://www.seamicro.com/sites/default/files/SM_TO01_64_v2.5.pdf, December 2013.

[4] Rajamony R, Stephenson M C, Speight W E. The power 775 architecture at scale. In *Proc. the 27th International ACM Conference on International Conference on Supercomputing* (*ICS2013*), June 2013, pp.183-192.

[5] Rao A. SeaMicro SM10000 system overview. Technical Report, AMD, June 2010. http://www.tiger-optics.ru/ download/seamicro/SM_TO02_v1.4.pdf, December 2013.

[6] Hoffman A J, Singleton R R. On Moore graphs with diameters 2 and 3. *IBM J. Research and Development*, 1960, 4(5): 497-504.

[7] Mattson T G, Van der Wijngaart R, Frumkin M. Programming the Intel 80-core network-on-a-chip terascale processor. In *Proc. the International Conference for High Performance Computing, Networking, Storage and Analysis* (*SC2008*), Nov. 2008, Article No.38.

[8] Bell S, Edwards B, Amann J *et al.* TILE64-processor: A 64-core SoC with mesh interconnect. In *Proc. International Solid-State Circuits Conference* (*ISSCC2008*), February 2008, pp.88-89.

[9] Seo J, Lee H, Jang M. Optimal routing and Hamiltonian cycle

in Petersen-Torus networks. In *Proc. the 3rd International Conference on Convergence and Hybrid Information Technology* (*ICCIT2008*), November 2008, pp.303-308.

[10] Barroso L A, Dean J, Hölzle U. Web search for a planet: The Google cluster architecture. *IEEE Micro*, 2003, 23(2): 22-28.

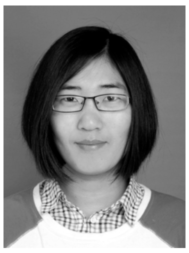
[11] O'Malley O. TeraByte sort on Apache Hadoop. Technical Report, Yahoo!, May 2008. http://sortbenchmark.org/Yahoo-Hadoop.pdf, December 2013.

[12] Esteves R M, Pais R, Rong C. K-Means clustering in the cloud – A Mahout test. In *Proc. the 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications* (*WAINA2011*), March 2011, pp.514-519.

[13] Thusoo A, Sarma J, Jain N *et al.* Hive: A warehousing solution over a map-reduce framework. In *Proc. the 35th International Conference on Very Large Data Bases* (*VLDB2009*), August 2009, pp.1626-1629.

[14] Adiga N R, Blumrich M A, Chen D *et al.* Blue Gene/L torus interconnection network. *IBM Journal of Research and Development*, 2005, 49(2): 265-276.

[15] Nan J, Becker D U, Michelogiannakis G *et al.* A detailed and flexible cycle-accurate Network-on-Chip simulator. In *Proc. IEEE International Symposium on Performance Analysis of Systems and Software* (*ISPASS2013*), April 2013, pp.86-96.



Wen-Tao Bao received the B.S. degree from Jilin University, Changchun, in 2012. Now she is pursuing her M.S. degree in Institute of Computing Technology, Chinese Academy of Sciences, Beijing. Her research interests include high-performance and high-reliable interconnection networks.

Bin-Zhang Fu received his B.Eng. degree in both electronics and information engineering, and computer science and technology from Huazhong University of Science and Technology, Wuhan, in 2004, and Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, in 2011. He is currently an associate professor at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include high-performance and high-reliable interconnection networks.

Ming-Yu Chen received his B.S. degree from University of Science and Technology of China in 1994 and M.S. and Ph.D. degrees in computer science from Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS) in 1997 and 2000, respectively. He is a processor in ICT, CAS. His research interests include computer architecture, operating system, and algorithm optimization for high performance computer.

Li-Xin Zhang received his B.S. degree in computer science from Fudan University and his Ph.D. degree in computer science from the University of Utah. He was a research associate from 1999 to 2003 at the University of Utah. He was a member of the Novel Systems Architecture group in the IBM Austin Research Lab from 2003 to 2010. Dr. Li-Xin Zhang is a professor at Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His research interests include data center computing systems, advanced cache/memory systems, architectural simulators, distributed/parallel computing, performance evaluation, and workload characterization.