文章编号:1001-2486(2011)06-0001-06

面向云计算的数据中心网络拓扑研究*

丁泽柳,郭得科,申建伟,罗爱民,罗雪山
(国防科技大学 信息系统工程重点实验室,湖南 长沙 410073)

摘 要:目前对数据中心网络拓扑的研究主要集中在如何提高结构性能上,却忽略了数据中心网络拓扑是否与云计算机制相适应的问题。针对该问题,建立了一种面向云计算的数据中心网络拓扑结构。研究了在具有该拓扑结构的数据中心网络上执行主流云计算机制的方法。分析了该拓扑结构的网络规模、网络直径等性能。仿真结果表明在具有该拓扑结构的数据中心网络上进行云计算是可行的。

关键词:数据中心网络;拓扑;云计算
中图分类号:TP393 **文献标识码**:A

Researching Data Center Networking Topology for Cloud Computing

DING Ze-liu, GUO De-ke, SHEN Jian-wei, LUO Ai-min, LUO Xue-shan
(Science and Technology on Information Systems Engineering Laboratory,
National Univ. of Defense Technology, Changsha 410073, China)

Abstract: Nowadays, the research work about data center networking (DCN) mainly focuses on the improvement of the topological properties of DCN. Unfortunately, it has been ignored whether the topologies of DCN are suitable for the cloud computing mechanisms running on DCN. To address this problem, a novel DCN topology for cloud computing was proposed. The effective method of running a mainstream cloud computing mechanism on the new topology was researched. The size and diameter of the new topology was analyzed. Simulation results demonstrate that the new topology is feasible for cloud computing.

Key words: data center networking (DCN); topology; cloud computing

数据中心是为大型复杂信息系统的海量数据提供分布式存储和计算的运行平台,它是分布式数据处理和云计算不断发展的产物[1]。数据中心网络是指数据中心的网络基础设施,它通过高速的链路和交换机连接着大量的服务器[2]。数据中心网络利用各类数据在服务器间的组织交互,向用户提供各种敏捷和高效的信息服务。

对这些海量数据进行具体操作和管理的是运行在数据中心网络上的数据管理系统和数据处理机制,如GFS[3]、HDFS[4]、Bigtable[5]、Dryad[6]等。其中影响最为深远的当属谷歌公司提出的一种分布式数据处理机制——MapReduce[7],它为云计算提供了良好的控制及执行方式。

近年来,各种新的服务需求的不断涌现对数据中心网络提出了更高的可扩展性、可靠性等结构性要求。针对这些要求,研究者们设计了一些新的数据中心网络拓扑,主要有Fat-tree[8]、DCell[2]、FiConn[9]以及BCube[10],它们各有侧重,从不同方面弥补了传统树形结构的不足。但研究者们都仅仅从结构本身来考虑如何提高网络性能,却忽略了数据中心网络设计的实用性要求,特别是与云计算机制相适应的问题。首先,这些新的网络拓扑都没有明确主服务器与数据服务器之间的连接关系。其次,这些网络的容错能力低,而且主服务器能够控制的数据服务器数量有限。最后,随着网络规模的扩大,它们的网络直径也快速增加,这不利于云计算的数据管理。只有拓扑结构和云计算机制相匹配的数据中心网络才能更好地满足各种新的服务需求。

针对目前数据中心网络拓扑的设计忽略云计算机制的问题,本文基于BCube和Fat-tree结构,设计了一种支持云计算的数据中心网络拓扑——Hyper-Fat-tree Network(HFN),它具有连通性高、直径小、可拓展性强的特点。研究了在HFN上执

* **收稿日期**:2011-03-25
基金项目:国家自然科学基金资助项目(60903206);国家部委基金资助项目;博士后基金资助项目(20100480898);国防科技大学资助项目
作者简介:丁泽柳(1983—),男,博士生。

行 MapReduce 基本步骤的路由方法。分析了 HFN 的运行效果。给出了数据中心网络拓扑设计结合实际应用的一个范例。

1　结构描述

HFN 采用递归层次结构设计,以具备较好的可拓展性。在有效满足数据中心网络基本建设要求的前提下,考虑了 MapReduce 对数据中心网络的结构要求,以此确定数据中心网络的节点连接关系。下面分别从最小递归单元和递归规律两个方面介绍。

最小递归单元给出了网络拓扑的结构单元,确定整个网络拓扑的构建基础。用 $HFN_0(N,M)$ 表示最小递归单元。N 表示 $HFN_0(N,M)$ 中有 N 个主服务器和 N 个交换机按照二分图的形式相互连接。这 N 个主服务器和 N 个交换机分别为二分图中被链路集分开的两个端点集。它们的连接方式为:第 0 个交换机连接第 0 个和第 1 个主服务器;第 n 个交换机($1\leqslant n\leqslant N-2$)连接第 $n-1$ 个、第 n 个、第 $n+1$ 个主服务器;第 $N-1$ 个交换机连接第 $N-2$ 个和第 $N-1$ 个主服务器;各交换机之间以及各主服务器之间不直接建立链接。M 表示每个交换机直接连接着各自的 M 个数据服务器。这种构造类似 Fat-tree 结构中集合层和叶子层的连接方法,区别是把 Fat-tree 集合层上部的交换机换成了主服务器,把叶子层的服务器换成了数据服务器。利用二分图和 Fat-tree 构建最小递归单元的目的是为了提高网络的可靠性。在具有这种结构的最小递归单元中,单个数据服务器故障或主服务器故障不会影响到其他任何节点。

递归规律给出了递归单元的连接方式,确定整个网络拓扑的构建规则。本文采用参考文献[10]中拓扑结构 BCube 的单元互联方式,即超立方体的节点连接关系作为递归规律。由于任意两个同级别单元中对应位置的服务器之间的通信仅有一跳,因此这种递归方式可以减小网络直径,提高通信质量和传输效率。用 $HFN_i(K,(N,M))$ 表示以 $HFN_0(N,M)$ 为最小递归单元的第 i 层递归单元($i\geqslant 1$)。K 表示 HFN_i 中第 i 层交换机的数量,K 等于每个 HFN_{i-1} 中主服务器的数量。一个 HFN_i 由 N 个 HFN_{i-1} 通过这 K 个交换机连接构成,连接方式为每个 HFN_{i-1} 中的第 k 个主服务器都与第 i 层的第 k 个交换机直接相连,其中 $0\leqslant k<K$。根据这样的连接方式可知 HFN_{i-1} 由 N^{i-1} 个 HFN_0 组成,每个 HFN_0 有 N 个主服务器,所以 HFN_{i-1} 中主服务器的数量为 N^i,即 $K=N^i$。$HFN_i(K,(N,M))$ 可表示为 $HFN_i(N,M)$。

为了表述方便,本文用 HFN_0 表示最小递归单元,用 HFN_i 表示第 i 层递归单元。用 $MServer_{i,j}$ 表示 HFN_i 的第 j 个主服务器,则 $MServer_{i,j}$ 可以唯一标识 HFN_i 的每个主服务器。用 $Switch_{i,k}$ 表示 HFN_i 中位于第 i 层的第 k 个交换机,$0\leqslant k<N^i$。

2　在 HFN 上运行 MapReduce 的路由方法

MapReduce 的基本运行过程包括两个阶段,即 Map 阶段和 Reduce 阶段[7]。每个阶段又分别包括多个并行的 Map 任务和 Reduce 任务。一个复杂的 MapReduce 程序会由一系列的工作组成[11],其中每个工作都包括一个 Map 阶段和一个 Reduce 阶段。关于 Map 和 Reduce 任务的详细执行过程可参见参考文献[7]。

在 MapReduce 的运行过程中,HFN 的节点路由方式主要包括三类:数据服务器与控制它的主服务器之间的路由;被同一个主服务器控制的数据服务器之间的路由;主服务器之间的路由;属于不同最小递归单元的数据服务器之间的路由。前两类路由在最小递归单元内部即可实现,比较简单,下面主要介绍后两类路由方法。

先介绍两个相关的定理。设 HFN 的总层次数为 I,按照 HFN 的构造方法可以得到下面两个定理。

定理 1　对于 HFN_I 中的任意主服务器 $MServer_{I,j}$ 及其所属的第 i 层递归单元 HFN_i,设 d 为 HFN_i 在 HFN_I 中的序号,则有:$d=\lfloor j/N^{i+1}\rfloor$。

证明　一个 HFN_i 包括 N^{i+1} 个主服务器,j 为 $MServer_{I,j}$ 在 HFN_I 中的序号。所以用 j 除以 N^{i+1} 再取整,即得到 $MServer_{I,j}$ 所属的 HFN_i 在 HFN_I 中的序号。

定理 2　设两个主服务器 $MServer_{I,x}$ 和 $MServer_{I,j}$ 分别属于同一个 HFN_{i+1} 中两个相邻的 HFN_i,若 $MServer_{I,x}$ 和 $MServer_{I,j}$ 连接到 HFN_{i+1} 中第 $i+1$ 层的同一个交换机,则有 $|x-j|=N^{i+1}$。其中 $|x-j|$ 表示 x 和 j 差值的绝对值。

证明　根据递归规律和已知条件,HFN_{i+1} 的第 $i+1$ 层交换机数量为 N^{i+1},两个相邻的 HFN_i 中只能有 $MServer_{I,x}$ 和 $MServer_{I,j}$ 连接到同一个 $i+1$ 层交换机,且位于 $MServer_{I,x}$ 和 $MServer_{I,j}$ 之间的主服务器分别连接着另外 $N^{i+1}-1$ 个交换机,所以 $MServer_{I,x}$ 和 $MServer_{I,j}$ 之间有 $N^{i+1}-1$ 个主服务器,因此 $|x-j|=N^{i+1}$。

从定理 2 不难得出推论 1。

推论 1　设 $MServer_{I,x}$ 和 $MServer_{I,j}$ 分别属于同一个 HFN_{i+1} 中的任意两个 HFN_i,且 $MServer_{I,x}$ 和 $MServer_{I,j}$ 连接到 HFN_{i+1} 中第 $i+1$ 层的同一个交换机,若这两个 HFN_i 在 HFN_{i+1} 中的序号分别为 d_1 和 d_2,则有 $|x-j| = |d_1-d_2|N^{i+1}$。

2.1　主服务器之间的路由方法

主服务器之间的路由取决于主服务器对 MapReduce 任务的分配方式。为了降低各服务器的负荷、减少网络通信量并保证数据局部性[12],本文采用的分配方式是把一个 MapReduce 的各项工作分别分配给多个主服务器来完成。即某个主服务器在收到 MapReduce 服务请求后,把每个工作分配给最近(起点到终点的链路跳数最小)的主服务器,且该主服务器控制的数据服务器存储着工作所需的数据。根据分布式文件系统[3-4]的数据管理机制,可认为所有的数据都存储在数据服务器上。

按照上述定理和服务分配方式,将 MapReduce 服务请求在 HFN 中的路由和分配方法表示为算法 1 和算法 2。算法 1 用于分配 MapReduce 的各项工作。算法 2 用于获取 $MServer_{I,j}$ 到 $MServers_{I,y}$ 的路由。设 $MServer_{I,j}$ 获得了用户的 MapReduce 服务请求,该服务的工作需要被分配给 L 个主服务器来执行。用 $Job_l(0 \leqslant l < L)$ 表示需要分配给某个主服务器的工作。FindedServers 为对象链表,按顺序记录获得工作的每个主服务器。Path 属性记录获得服务请求的主服务器到每个获得工作的主服务器的路径。

Algorithm 1: AssignJobs (int j, int L)

```
1: objectList FindedServers;
2: for  l=0; l<L; l++
3:   if  MServers_{I,y}.DServers.Data = Job_l.Data; //表示 MServers_{I,y} 控制的数据服务器存有 Job_l 需要的数据;
4:       MServers_{I,y}.Path = {MServer_{I,j}; };
5:       MServers_{I,y}.Path = CreatRouting1(I-1,j,y);
6:       assign Job_l to MServer_{I,y};
7:       add MServer_{I,y} to FindedServers;
```

Algorithm 2: CreateRouting1 (int f, int j, int y)

```
1: int g=0; int x=0;
2: for  i=f; i≥0; i--
3:   if  i>0
4:     if  j/N^{i+1} ≠ y/N^{i+1}
5:       int h=(y-j)/N^{i+1}; //"/"表示相除后取整数部分
6:       x=j+h×N^{i+1};
7:       if  (j+1)%N==0
8:         x=j-1; // (j+1)%N 表示 j+1 除以 N 取余,如果该值为 0,说明 MServer_{I,j} 是它所在 HFN_0 的最后一个主服务器。
9:       else x=j+1;
10:      add MServer_{I,x} to MServer_{I,y}.Path;
11:      g=i; break;
12:  if  i=0
13:    if  j-y>2
14:      for  x=j-2; x>y; x-=2
15:        add MServer_{I,x} to MServer_{I,y}.Path;
16:    if  y-j>2
17:      for  x=j+2; x<y; x+=2
18:        add MServer_{I,x} to MServer_{I,y}.Path;
19:    add MServer_{I,y} to MServer_{I,y}.Path;
20:    return MServer_{I,y}.Path;
21: CreateRouting1 (g,x,y);
```

2.2　数据服务器之间的路由方法

不同最小递归单元存储不同类型的数据,以保证数据局部性。因此,传递不同类型的数据需要利用数据服务器间的路由。算法 3 在算法 2 的基础上给出了分属不同 HFN_0 的数据服务器之间的路由方法。$WServer_{j,m1}$ 和 $WServer_{y,m2}$ 分别表示 $MServer_{I,j}$ 和 $MServers_{I,y}$ 控制的任意数据服务器。

Algorithm3:CreateRouting2(int j, int y, int $m1$, int $m2$)

```
1: WServer_{j,m1}.Path = {WServer_{j,m1}; MServer_{I,j};};
2: WServer_{j,m1}.Path = CreatRouting1(I-1,j,y);
3: add WServer_{y,m2} to WServer_{j,m1}.Path;
4: return WServer_{j,m1}.Path;
```

3　评 价

3.1　拓扑性能

下面把 HFN 的拓扑性能与 FiConn 和 BCube 这两个典型的递归定义的数据中心网络结构的拓扑性能进行比较。

3.1.1　网络规模

数据中心的网络规模取决于可容纳的最大服务器数量,反映了网络的可拓展性。根据 HFN 的最小递归单元结构和递归规律,可得到 $HFN_i(N,M)$ 中主服务器的数量为 N^{i+1},数据服务器的数量

为 $M \times N^{i+1}$。所以一个 HFN_i 可连接的服务器总数量为 $(M+1)N^{i+1}$。同层次 FiConn 的服务器数量为 $2^{i+2} \times (N/4)^{2}$[9]。同层次 BCube 的服务器数量为 N^{i+1}[10]。图 1 显示，随着层次数 i 的增加，HFN 的网络规模要远大于 FiConn 和 BCube 的规模。

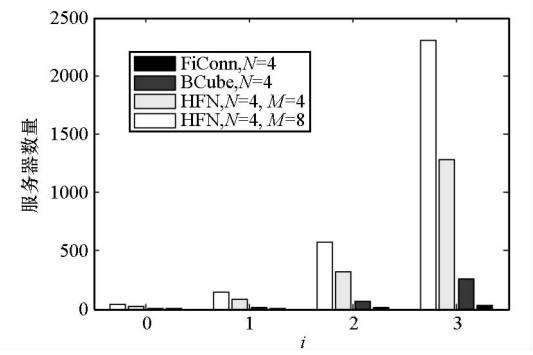

图 1　网络可容纳的最大服务器数量
Fig. 1　The maximum number of servers

3.1.2　网络直径

数据中心网络的直径是指网络中任意两个服务器之间最短距离的最大值。直径越短，网络各节点之间的数据交换就越便捷。如果把所有的主服务器和数据服务器抽象成同等的节点，则 HFN_i 的直径为 $i+1+N/2$。其中 $N/2$ 取整数部分，表示最小递归单元内两个数据服务器之间的最大跳数。同层次 FiConn 的直径为 $2^{i+1}-1$[9]。同层次 BCube 的直径为 $i+1$[10]。图 2 显示，随着层次数 i 的增加，FiConn 的直径要远大于 HFN 和 BCube 的直径。当 N 取值不大时，HFN 和 BCube 的直径相差不大。

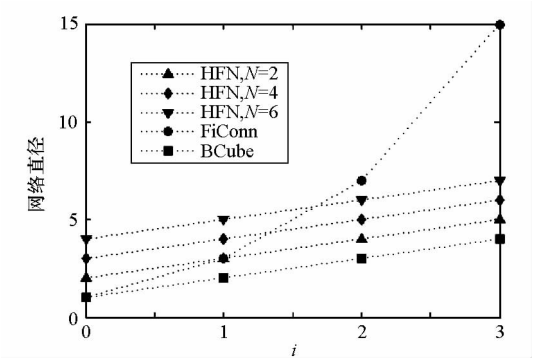

图 2　网络直径的变化情况
Fig. 2　The variation of network diameter

3.2　执行时间

设到达一个数据服务器的 Map 和 Reduce 任务的数量服从泊松分布，在单位时间内到达的平均数量分别为 λ_1 和 λ_2。设一个数据服务器执行 Map 和 Reduce 任务的时间服从负指数分布，执行的平均时间分别为 $1/\mu_1$ 和 $1/\mu_2$。

定理 3　令单个工作的平均执行时间为 t，则有

$$t = \frac{1}{\mu_1 - \lambda_1} + \frac{1}{\mu_2 - \lambda_2} \tag{1}$$

证明　根据网络排队理论可得，单个 Map 任务从分配到执行完毕的平均总时间为 $1/(\mu_1 - \lambda_1)$，包括任务的等待时间和执行时间。Map 步骤的多个 Map 任务是并行执行的，所以 Map 步骤的平均时间也为 $1/(\mu_1 - \lambda_1)$，同理 Reduce 步骤的平均时间为 $1/(\mu_2 - \lambda_2)$。因此得证。

令 C 为一个 MapReduce 包括的工作数量。令 $U = (u_{x,y})$ 表示这些工作的邻接矩阵，

$$u_{x,y} = \begin{cases} 1 & Job_y \text{ 是 } Job_x \text{ 的下一个工作} \\ 0 & Job_y \text{ 不是 } Job_x \text{ 的下一个工作} \end{cases} \tag{2}$$

其中 $0 \leqslant x < C$，$0 \leqslant y < C$。

令 $V = (v_{x,y})$ 表示这些工作的可达矩阵，$v_{x,y}$ 表示从 Job_x 到 Job_y 的工作流程中所有工作的数量。算法 4 基于 Floyd 算法[13]，给出了计算 $V = (v_{x,y})$ 的方法。

Algorithm 4 CreateMatrixV

```
1: for z = 0; z < C; z++
2:     for x = 0; x < C; x++
3:         for y = 0; y < C; y++
4:             v_{x,y} = u_{x,y} = max{u_{x,y}, u_{x,z} + u_{z,y}};
```

定理 4　令 T 表示一个 MapReduce 在 HFN 上运行的平均时间，则

$$T = \max_{\substack{0 \leqslant x < C \\ 0 \leqslant y < C}} \{v_{x,y}\} \times t \tag{3}$$

证明　$\max\{v_{x,y}\}$ 记录了 MapReduce 所有工作流程中的最大工作数量。MapReduce 的运行时间等于这些工作的执行时间之和。又因为每个工作的平均时间是 t，因此得证。

3.3　考虑节点故障的执行时间

设 HFN 中的主服务器、数据服务器和交换机在运行一个 MapReduce 的过程中发生故障的概率分别为 P_M、P_D 和 P_S。

定理 5　令主服务器在执行一个 MapReduce 过程中保持正常工作的概率为 P_1，则有

$$P_1 = \begin{cases} 1 - P_M & i = 0 \\ (1 - P_M) \times \left\{1 - \sum_{n=1}^{i} \left[\binom{i}{n} \times P_S^n\right]\right\} & i > 0 \end{cases} \tag{4}$$

证明　$1-P_M$ 为主服务器不发生故障的概率。当 $i=0$ 时主服务器不与层次交换机相连,$1-P_M$即为主服务器保持正常工作的概率。当 $i>0$ 时,$\binom{i}{n}\times P_S^n$ 为与一个主服务器相连的 i 个层次交换机中有 n 个发生故障的概率。则 $1-\sum_{n=1}^{i}\left[\binom{i}{n}\times P_S^n\right]$表示与 主服务器直接相连的所有层次交换机都不发生故障的概率。只有当主服务器和与它相连的每个层次交换机都不出现故障,主服务器才能正常工作。因此得证。

定理 6　令数据服务器在执行一个 MapReduce 的过程中保持正常工作的概率为 P_2,则有

$$P_2=(1-P_D)\times(1-P_S) \tag{5}$$

证明　$1-P_D$ 为数据服务器不发生故障的概率。$1-P_S$ 为交换机不发生故障的概率。每个数据服务器仅连接到一个交换机。只有当交换机和数据服务器都无故障,数据服务器才能正常工作。因此得证。

定理 7　如果考虑节点的故障率,令单个工作的平均执行时间为 t',则

$$t'=\frac{t}{P_2} \tag{6}$$

证明　设一个工作有 M 个 Map 任务被分配给 M 个数据服务器来执行。由于数据服务器保持正常工作的概率为 P_2,这 M 个任务中有 $M\times(1-P_2)$个会被重新执行。这 $M\times(1-P_2)$个被重新执行的任务中,又有 $M\times(1-P_2)^2$ 个任务会被执行第 3 次。以此类推,根据定理 3,在单个工作中,Map 阶段的平均执行时间 为 $M\times\sum_{g=0}^{\infty}(1-P_2)^g\times\frac{1}{\mu_1-\lambda_1}\times\frac{1}{M}$, 即$\frac{1}{\mu_1-\lambda_1}\times\frac{1}{P_2}$。同理,Reduce 阶段的平均执行时间为$\frac{1}{\mu_2-\lambda_2}\times\frac{1}{P_2}$。因此,由式(1)得单个工作的平均执行时间为$t/P_2$。

定理 8　如果考虑节点的故障率,令 MapReduce 在 HFN 上运行的平均时间为 T',则有

$$T'=\max_{\substack{0\leqslant x<C\\0\leqslant y<C}}\{v_{x,y}\}\times\frac{t'}{P_1} \tag{7}$$

证明　主服务器的故障率为 P_1,$\max\{v_{x,y}\}$ 个工作中有 $\max\{v_{x,y}\}\times(1-P_1)$个会被重新执行。与定理 7 的证明相同,推得 MapReduce 的平均运行时间为 $\max\{v_{x,y}\}\times t'\times\sum_{g=0}^{\infty}(1-P_1)$, 即 $\max\{v_{x,y}\}\times(t'/P_1)$。

把式(1)代入式(6),再代入式(7)可得

$$T'=\max_{\substack{0\leqslant x<C\\0\leqslant y<C}}\{v_{x,y}\}\times\left(\frac{1}{\mu_1-\lambda_1}+\frac{1}{\mu_2-\lambda_2}\right)\times\frac{1}{P_1\times P_2} \tag{8}$$

3.4　案例

以参考文献[11]中用 MapReduce 计算一个由节点和边组成的图中的三角形个数为例子。该过程的输入数据为图的每个节点和每条边,包括 4 个顺序执行的工作:工作 1,计算图中每个节点的度;工作 2,计算每条边的两个端点的度;工作 3,找出所有相邻的两条边,且这两条边的公共端点是它们端点中度最小的那个;工作 4,找出每对相邻边的不相邻端点间是否存在一条边,即算出三角形。

由于 4 个工作都是顺序执行的,根据式(2)和算法 4,可得 $\max\{v_{x,y}\}=4$。若以一个一层的 HFN 来执行,设主服务器、数据服务器以及交换机的故障率分别为 0.01、0.02 和 0.01。根据式(4)和(5)得 $P_1=0.98$,$P_2=0.97$。代入式(8),以 s 为时间单位,总执行时间 T'的变化情况如图 3 所示,如果以 $\mu_2-\lambda_2$ 为横坐标,会得到相同的结果。

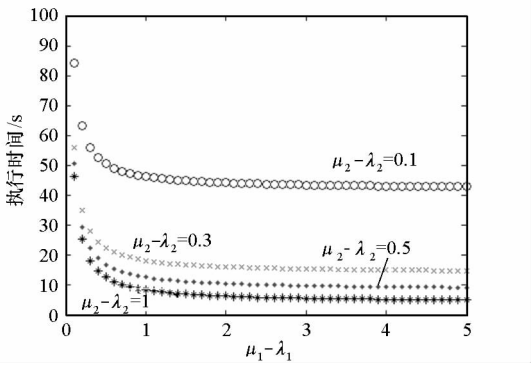

图 3　总的执行时间

Fig. 3　The total execution time

根据 3.2 节中的定义,μ_1 和 μ_2 分别表示单个数据服务器在单位时间内能够处理的 Map 和 Reduce 任务的数量。λ_1 和 λ_2 的值取决于用户对 MapReduce 服务请求的频率。如果 μ_1 与 λ_1 的差值或者 μ_2 与 λ_2 的差值较小,表明当前的服务器处理能力已经不能满足用户的需求,数据中心网络需要添加更多的服务器。图 3 显示,利用 HFN 执行 MapReduce,当 μ_1 与 λ_1 的差值以及 μ_2 与 λ_2 的差值在 1 以上时,T'的值就能保持在相对较小的范围内。

可见在层次数较低和考虑节点故障的情况下,利用 HFN 数据中心网络拓扑进行云计算能够

有效满足用户的时间要求和工作量要求。

4 小结

本文提出了一种支持云计算的数据中心网络拓扑结构(HFN)。它具有连通性高、直径小、可靠性好、可拓展性强的特点。给出了在 HFN 上利用 MapReduce 的基本步骤进行云计算的具体方法。分析了在节点故障情况下利用 HFN 执行 MapReduce 的效果。本文的研究目的在于使数据中心网络的拓扑结构和数据处理机制相适应,使得数据中心网络结构能够面向实际应用和解决实际问题。

参考文献:

[1] Greenberg A, Hamilton J R, et al. VL2: A Scalable and Flexible Data Center Network [C]//ACM SIGCOMM Computer Communication Review,2009,39(4):51 -62.

[2] Guo C, Wu H, et al. DCell: A Scalable and Fault-tolerant Network Structure for Data Centers[C]//ACM SIGCOMM, 2008:75 -86.

[3] Ghemawat S, Gobioff H, Leung S T. The Google File System [C]//Proc. 19th ACM Symposium on Operating Systems Principles, 2003:29 -43.

[4] Borthakur D. The Hadoop Distributed File System: Architecture and Design[R/OL]. [2011 -02 -20]. http://hadoop. apache. org/core/docs/current/hdfsdesign. pdf

[5] Chang F, Dean J,et al. Bigtable: A Distributed Storage System for Structured Data[C]//Proc. 7th Symposium on Operating Systems Design and Implementation (OSDI), 2006: 205 -218.

[6] CloudStore. Higher Performance Scalable Storage [R/OL]. [2011 -02 -20]. http://kosmosfs. sourceforge. net/

[7] Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters[J]. Communications of the ACM, 2008,51 (1):107 -113.

[8] Al-Fares M, Loukissas A, Vahdat A. A Scalable, Commodity Data Center Network Architecture [C]//Proc. ACM SIGCOMM, 2008: 63 -74.

[9] Li D, Guo C, et al. FiConn: Using Backup Port for Server Interconnection in Data Centers[C]//Proc. IEEE INFOCOM, 2009: 2276 -2285.

[10] Guo C, Lu G, et al. BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers[C]// Proc. ACM SIGCOMM, 2009: 63 -74.

[11] Cohen J. Graph Twiddling in a MapReduce World[J]. IEEE Computing in Science and Engineering, 2009, 2(4): 29 -41.

[12] Bastoul C, Feautrier P. Improving Data Locality by Chunking[J]. Springer Lecture Notes in Computer Science, 2003, 2622: 320 -334.

[13] Weisstein E. Floyd-warshall Algorithm [R/OL] Wolfram MathWorld, 2009 [2011 -02 -20]. http://mathworld. wolfram. com/Floyd -WarshallAlgorithm. html