

# Dynamic load balancing based on restricted multicast tree in triplet-based hierarchical interconnection network

Liu Bin<sup>1,2</sup> Shi Feng<sup>1</sup> Gao Yujin<sup>1</sup> Ji Weixing<sup>1</sup> Song Hong<sup>1</sup>

(<sup>1</sup>School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China)

(<sup>2</sup>College of Economics and Management, Hebei University of Science and Technology, Shijiazhuang 050018, China)

**Abstract:** To solve the load balancing problem in a triplet-based hierarchical interconnection network (THIN) system, a dynamic load balancing (DLB) algorithm—THINDLBA, which adopts multicast tree (MT) technology to improve the efficiency of interchanging load information, is presented. To support the algorithm, a complete set of DLB messages and a schema of maintaining DLB information in each processing node are designed. The load migration request messages from the heavily loaded node (HLN) are spread along an MT whose root is the HLN. And the lightly loaded nodes (LLNs) covered by the MT are the candidate destinations of load migration; the load information interchanged between the LLNs and the HLN can be transmitted along the MT. So the HLN can migrate excess loads out as many as possible during a one time execution of the THINDLBA, and its load state can be improved as quickly as possible. To avoid wrongly transmitted or redundant DLB messages due to MT overlapping, the MT construction is restricted in the design of the THINDLBA. Through experiments, the effectiveness of four DLB algorithms are compared, and the results show that the THINDLBA can effectively decrease the time costs of THIN systems in dealing with large scale compute-intensive tasks more than others.

**Key words:** triplet-based hierarchical interconnection network; dynamic load balancing; multicast tree

The triplet-based hierarchical interconnection network (THIN), whose lowest hierarchy is completely connected and other hierarchies have relatively fewer numbers of links, well supporting computing locality, can be used to construct homogeneous parallel processing systems<sup>[1-3]</sup>. To efficiently utilize resources, load balances should be achieved in THIN systems. Compared with static load balancing algorithms, dynamic load balancing (DLB) algorithms can make load migration decisions according to the real-time system load state<sup>[4]</sup>. Experiments suggest that DLB algorithms outperform static ones in the mean response time at most about 30% in common situations<sup>[5]</sup>. THINDLBA, a THIN system-oriented DLB algorithm, utilizing multicast tree technology to

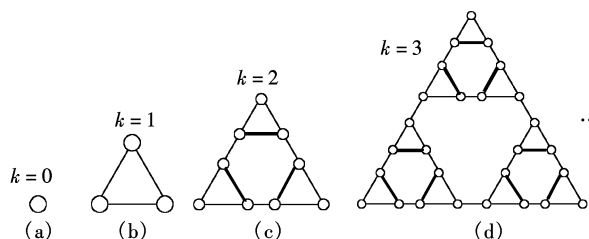
reduce the costs due to interchanging DLB messages among processing nodes, is presented in this paper.

## 1 THIN

As shown in Fig. 1, THIN is a hierarchical and scalable architecture. The level-0 THIN is just a single node. The level-1 THIN, a triangle connecting three nodes with communication links, is the basic component for constructing THIN. In fact, THIN is a fractal object called a Sierpinski Gasket. According to the fractal geometry theory, a Sierpinski Gasket can be produced by the iterator function system (IFS). Suppose that the IFS of THIN is  $IFS\{F_1, F_2, F_3\}$ , and any level THIN can be constructed from level-0 THIN with the IFS. Level- $k$  THIN, represented as  $N(k)$ , is regarded as the result of  $k$  times iterations of the IFS. The construction process of THIN can be represented as

$$N_{k+1} = \bigcup_{i=1}^3 F_i(N_k) \quad (1)$$

Refs. [1–3] presented more introductions about THIN in detail.



**Fig. 1** Topology of THIN. (a) Level-0; (b) Level-1; (c) Level-2; (d) Level-3

## 2 Multicast Tree

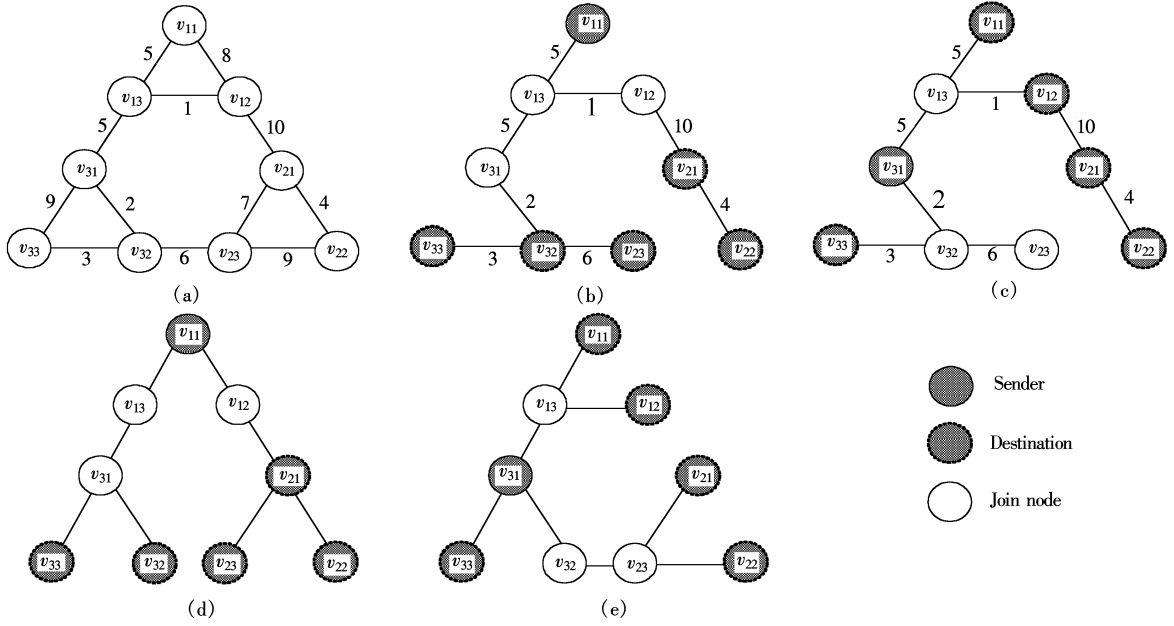
According to the objectives, multicast trees are classified into two categories, the shortest path tree (SPT) and the minimal Steiner tree (MST). The SPT is designed to minimize the message transmit delay between a sender and the destinations; the MST is designed to minimize the network costs consumed by a multicast group<sup>[6-7]</sup>. Let  $v_s$  and  $\{v_{di} \mid 1 \leq i \leq n, n \leq \text{total number of nodes in the system}\}$  represent the sender and the destinations, respectively. The multicast tree can be represented as  $MT(v_s, v_{d1}, v_{d2}, \dots, v_{dn})$ . Fig. 2, where the numbers attached to the edges represent the data transfer costs, gives examples of the SPT and the MST in THIN. In this paper, the data transfer costs of each edge are assumed to be equal. So, the SPT is adopted to decrease the data transfer costs due to interchanging DLB messages.

Received 2007-09-24.

**Biographies:** Liu Bin (1975—), male, graduate, lecturer, geegle@tom.com; Shi Feng (corresponding author), male, doctor, professor, shifengyoujian@tom.com.

**Foundation item:** The National Natural Science Foundation of China (No. 69973007).

**Citation:** Liu Bin, Shi Feng, Gao Yujin, et al. Dynamic load balancing based on restricted multicast tree in triplet-based hierarchical interconnection network[J]. Journal of Southeast University (English Edition), 2008, 24(1): 33–37.



**Fig. 2** Examples of multicast tree. (a) Level-3 THIN; (b)  $MST(v_{11}, v_{21}, v_{22}, v_{23}, v_{32}, v_{33})$ ; (c)  $MST(v_{31}, v_{11}, v_{12}, v_{21}, v_{22}, v_{33})$ ; (d)  $SPT(v_{11}, v_{21}, v_{22}, v_{23}, v_{32}, v_{33})$ ; (e)  $SPT(v_{31}, v_{11}, v_{12}, v_{21}, v_{22}, v_{33})$

### 3 THINDLBA

#### 3.1 Load index

The processing ability of the node  $v_i$ , which is also the maximum number of processes processed by  $v_i$  per time unit, is represented as  $P_{\max}(v_i)$ ; the load threshold is represented as  $P_{\text{limit}}(v_i)$ ; the current CPU run queue length is represented as  $Q_{\text{cpu}}(v_i)$ ; then the load state of  $v_i$  can be calculated according to

$$S_{\text{load}}(v_i) = \begin{cases} \text{heavy} & Q_{\text{cpu}}(v_i) > P_{\max}(v_i) + P_{\text{limit}}(v_i) \\ \text{normal} & P_{\max}(v_i) - P_{\text{limit}}(v_i) \leq Q_{\text{cpu}}(v_i) \leq P_{\max}(v_i) + P_{\text{limit}}(v_i) \\ \text{light} & Q_{\text{cpu}}(v_i) < P_{\max}(v_i) - P_{\text{limit}}(v_i) \end{cases} \quad (2)$$

where “heavy” represents heavily loaded, and the number of processes that need to be migrated out is  $P_{\text{out}}(v_i) = Q_{\text{cpu}}(v_i) - P_{\max}(v_i) - P_{\text{limit}}(v_i)$ ; “normal” represents normally loaded, neither migrate-in nor migrate-out processes; “light” represents lightly loaded, and the number of processes can be migrated in is  $P_{\text{in}}(v_i) = P_{\max}(v_i) + P_{\text{limit}}(v_i) - Q_{\text{cpu}}(v_i)$ .

#### 3.2 DLB message

1) RFH (request from HLN-heavily loaded node): ① HLN; ② NRP: The number of excess processes of the HLN. Original sender: The HLN that wants to migrate loads out; Transmitter: Lightly loaded nodes (LLNs) or normally loaded nodes (NLNs); Destination: The LLNs that can migrate processes in from the HLN.

2) RFL (response from LLN): ① HLN; ② LLN; ③ MIP: The number of processes that LLN can migrate in. Original sender: The LLN that responds to RFH; Transmitter: LLNs or NLNs; Destination: The original sender of the RFH.

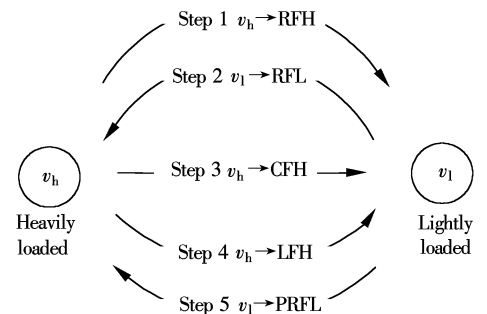
3) CFH (confirm from HLN): ① HLN; ② LLN; ③ DEPN: The number of processes that the HLN decides to migrate out to the LLN. Original sender: The HLN that re-

sponds to the RFL; Transmitter: LLNs or NLNs; Destination: The original sender of the RFL.

4) LFH (load from HLN): ① HLN; ② LLN; ③ PI: Context of the migrated process; ④ RR: Whether to return the processing result of the migrated process should be assigned “T” or “F”. Original sender: The HLN responds to the RFL; Transmitter: NLNs; Destination: The original sender of the RFL.

5) PRFL (processing result from LLN): ① HLN; ② LLN; ③ R: Processing result of the migrated process; ④ LP (last process): Indicates whether  $R$  is the result of the last process returned from LLN to HLN should be assigned “T” or “F”. It decides whether the node in multicast tree should still maintain the routing information of its “prior” and “subsequent” neighbours or not. Original sender: The LLN that is still light or has become normal after migrating processes in; Transmitter: NLNs; Destination: The HLN is still heavy or has become normal after process migration.

The message is represented as  $\text{Message}(\text{Parameter}_1, \text{Parameter}_2, \dots, \text{Parameter}_n)$ ; the hypotaxis relationship between a message and its parameters is represented as “.”; the message originating from  $v_i$  is represented as  $v_i \rightarrow \text{message}$ . The messages interchanged in a successful DLB process is shown in Fig. 3.



**Fig. 3** DLB messages interchanged in the THINDLBA

### 3.3 Information maintained by node

The information maintained by a processing node is as follows:

1) RTN (RFH transfer node): If  $v_i$  is LLN or NLN, it will respond RFH, and RTN records the neighbour which transmits the RFH to  $v_i$ . RTN can be assigned as “Null” or a neighbour.

2) LEP (left excess process): To an HLN, maybe some excess processes have been assigned to some LLNs, but have not been practically migrated. LEP records the number of excess processes that are still not assigned out. If LEP equals 0, the HLN is considered as “bogus heavy”. LEP should be assigned as  $0 \leq \text{LEP} \leq P_{\text{out}}(v_i)$ , and initialized as  $P_{\text{out}}(v_i)$ .

3) LPM (left place for migration): LPM records the number of processes that an LLN can accept except for the processes that it has answered to accept but still not received. If LPM equals 0, the LLN is considered as “bogus light”. LPM should be assigned as  $0 \leq \text{LPM} \leq P_{\text{in}}(v_i)$ , and initialized as  $P_{\text{in}}(v_i)$ .

4) NRT (neighbour routing table): Table model (OS-original sender of RFH or RFL; DN-destination of RFH or RFL; PN-prior node that sends RFH or RFL to  $v_i$ ; SN-subsequent node that  $v_i$  sends RFH or RFL to). During load migration and result return of the migrated process, NRT helps  $v_i$  select its neighbour to transmit a DLB message.

### 3.4 Algorithm descriptions

**Algorithm 1** Search the LLNs for load migration

**Input:** ①  $v_h$ : An HLN that wants to migrate loads out; ②  $T_{\text{wait}}$ : Upper time threshold of waiting for RFL. It can be set as twice the communication delay between  $v_h$  and the farthest node.

**Output:** A set of LLNs which are covered by a multicast tree and can migrate in the loads from  $v_h$ .

**Step 1**  $v_h$  calculates  $P_{\text{out}}(v_h)$ .

**Step 2**  $v_h$  sends RFH( $v_h, P_{\text{out}}(v_h)$ ) to all of its neighbours.

**Step 3** Within  $T_{\text{wait}}$ , if  $v_h$  receives no RFLs (RFL. HLN =  $v_h$ ), go to step 1; otherwise, algorithm 2 is executed, anytime  $v_h$  changes to normal.

**Step 4** End.

**Algorithm 2** Deal with DLB messages

**Input:** ①  $v_i$ : The node that just receives a message; ② ms: The message transmitted from  $v_j$ ; ③  $v_j$ : A neighbour of  $v_i$ .

**Output:** Strategy for dealing with ms.

**Step 1** If  $v_i$  is light, go to step 2; if normal, go to step 3; heavy, go to step 4.

**Step 2** If ms is RFH, go to step 2.1; if RFL or PRFL, go to step 2.2; if CFH, go to step 2.3; if LFH, go to step 2.4.

**Step 2.1** If  $v_i$  is “bogus light” or has responded to other HLN’s CFH (RTN is not “null”), or it does not respond to the ms, go to step 5; otherwise, set RTN = RFH. HLN and  $v_i$  sends RFL (RFH. HLN,  $v_i$ , LPM( $v_i$ )) to  $v_j$ , and ① If LPM( $v_i$ )  $\geq$  RFH. NRP, a new record (RFH. HLN,  $v_i$ ,  $v_j$ , null) is inserted into NRT; ② Otherwise, RFH. NRP should

be modified as RFH. NRP-RFH. NRP-LPM( $v_i$ ), and RFH is multicast to all the  $v_i$  neighbours except  $v_j$ , the corresponding number of new records-(RFH. HLN, null,  $v_j$ ,  $v_i$ ’s neighbour) are inserted into NRT, go to step 5.

**Step 2.2** If ms is RFL or PRFL, it means that  $v_i$  has transmitted RFH to  $v_j$  in the past as described in step 2.1. Locate record  $r$  in NRT with the restricted condition:  $r$ . SN =  $v_j$  and  $r$ . OS = ms. HLN, ① PRFL. PI = “T”, delete  $r$  from NRT after transmitting the PRFL to  $r$ . PN; ② Otherwise, update  $r$  as (ms. HLN, ms. LLN,  $r$ . PN,  $v_j$ ) and transmit the ms to  $r$ . PN, go to step 5.

**Step 2.3** If CFH. LLN is  $v_i$ , set RTN = null and LPM( $v_i$ ) = LPM( $v_i$ ) - CFH. DEPN; otherwise, locate record  $r$  in NRT with the restricted condition:  $r$ . PN =  $v_j$  and  $r$ . OS = CFH. HLN,  $v_i$  transmits CFH to  $r$ . SN, go to step 5.

**Step 2.4** If LFH. LLN is  $v_i$ , take out and deal with the migrated process. When finished dealing with the migrated process, locate record  $r$  in NRT with the restricted condition:  $r$ . PN =  $v_j$  and  $r$ . OS = LFH. HLN, if LFH. RR = “T”, send back PRFL( $r$ . HLN,  $r$ . LLN,  $R$ , LP) to  $v_j$ . After dealing with all the migrated loads from LFH. HLN, delete  $r$  from NRT.

**Step 3** If ms is RFH, go to step 3.1; if RFL or PRFL, go to step 3.2; if CFH or LFH, go to step 3.3.

**Step 3.1** If RTN equals null, let RTN =  $v_j$ , RFHs are multicast to all the  $v_i$ ’s direct neighbours except  $v_j$ ; if the corresponding number of new records (RFH. HLN, null,  $v_j$ ,  $v_i$ ’s neighbour) is inserted into NRT, go to step 5.

**Step 3.2** Locate record  $r$  in NRT with the restricted condition:  $r$ . SN =  $v_j$  and  $r$ . OS = ms. HLN. ① If PRFL. PI = “T”, delete  $r$  from NRT after transmitting the PRFL to  $r$ . PN; ② Otherwise, update  $r$  as (ms. HLN, ms. LLN,  $r$ . PN,  $v_j$ ); ③ And transmit the ms to  $r$ . PN, go to step 5.

**Step 3.3** Locate record  $r$  in NRT with the restricted condition:  $r$ . PN =  $v_j$  and  $r$ . OS = ms. HLN. ① Update  $r$  as (ms. HLN, ms. LLN,  $v_j$ ,  $r$ . SN); ② And transmit the ms to  $r$ . SN, go to step 5.

**Step 4** If ms is RFL, go to step 4.1; if PRFL, go to step 4.2.

**Step 4.1** If RFL. HLN is  $v_i$ , ① When LRP( $v_i$ )  $\geq$  RFL. MIP, set LRP( $v_i$ ) = LRP( $v_i$ ) - RFL. MIP and send CFH( $v_i$ , RFL. LLN, RFL. MIP) to  $v_j$ ; ② When LRP( $v_i$ ) < RFL. MIP, set LRP( $v_i$ ) = 0 and send CFH( $v_i$ , RFL. LLN, LRP( $v_i$ )) to  $v_j$ . Execute algorithm 3 and go to step 5.

**Step 4.2** If PRFL. HLN is  $v_i$ , take out  $R$  and go on processing the local loads.

**Step 5** End.

**Algorithm 3** HLN migrates loads out to an LLN

**Input:** ①  $v_h$ : An HLN that wants to migrate excess loads out; ②  $v_l$ : The LLN that migrates in the loads from  $v_h$ ; ③  $v_{\text{in}}$ : The first neighbour of  $v_h$  in the migration route originating from  $v_h$ ; ④ mpn: The number of the migrated processes.

**Output:** A set of LFHs from  $v_h$ .

**Step 1** The HLN randomly selects a process  $p$  from the CPU run queue and capsules the context of  $p$  to LFH. PI, sends LFH( $v_h, v_l, \text{PI}, \text{RR}$ ) to  $v_{\text{in}}$ . If the process is the last migrated one, LFH. RR is set to “T”, otherwise “F”.

**Step 2** If the number of the migrated processes does not go beyond  $mpn$ , go to step 1.

**Step 3** End.

#### 4 Experimental Results and Analysis

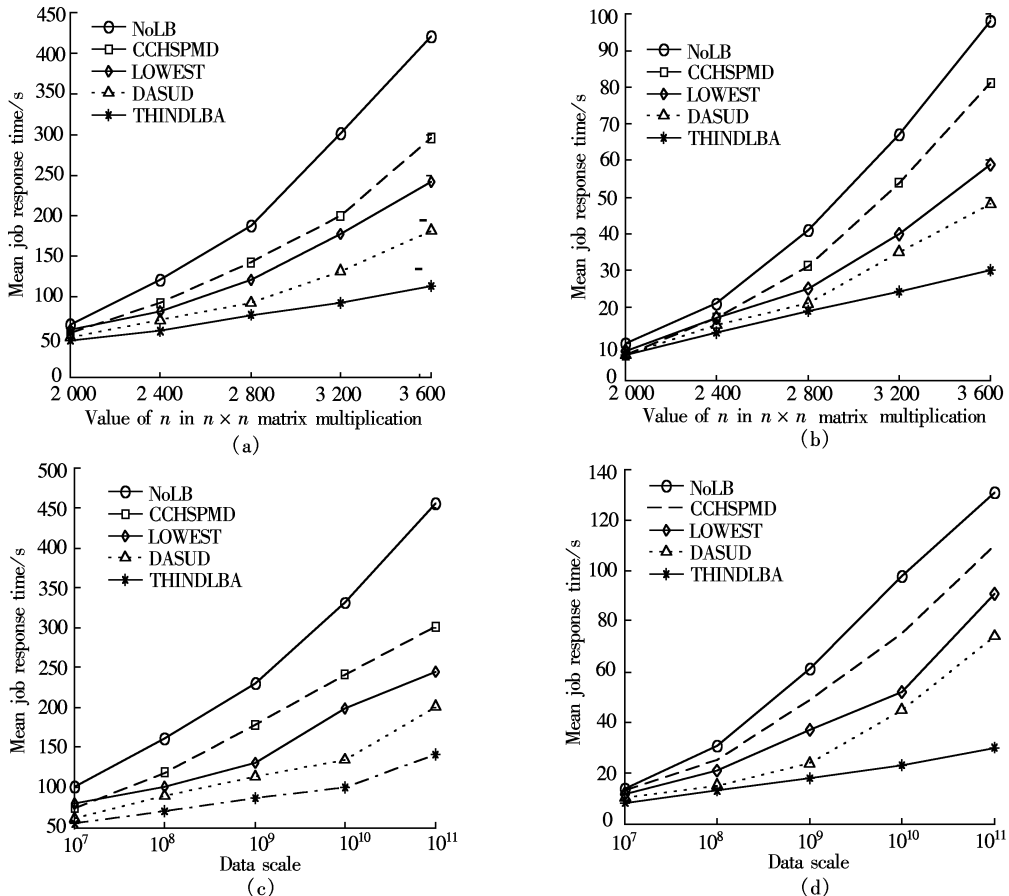
One boundary case (NoLB), the THINDLBA and other three DLB algorithms are simulated as follows: ① NoLB: No load balancing; ② CCHSPMD: It is centralized controlled and adaptive to compute-intensive tasks<sup>[8]</sup>; ③ LOWEST: It is distributed controlled and sender-initiated. The LLN migrating loads in is the lightest one among a fixed number of randomly chosen LLNs that satisfy specified restrictions<sup>[9]</sup>; ④ DASUD: It is distributed controlled and sender initiated. When an HLN searches LLNs, the range is first limited to its neighbours, if no LLNs are found, the range is expanded layer by layer until an LLN is found<sup>[10]</sup>.

**Experiment 1** Matrix computation is representative in a high performance computing field, and it also has a lot of significance in scientific research and engineering applications. So, a parallel matrix multiplication experiment is executed. The results are shown in Fig. 4(a) and Fig. 4(b).

**Experiment 2** Finding prime numbers in large-scale data is an important scientific field and can produce practical benefits in computer security applications. When initialized, each node is equally assigned tasks, and the linear interpolation algorithm is used as the fitting algorithm. Because the iterative cycle procedures are independent and each time the iterative cycle result is different, the experiment can prefera-

bly simulate the load unbalance phenomena. The results are shown in Fig. 4(c) and Fig. 4(d).

According to Fig. 4, we can find that: ① When the computation scale is relatively small, CCHSPMD outperforms LOWEST, but it is inferior to DASUD and THINDLBA; when the computation scale is enlarged, LOWEST, DASUD and THINDLBA gradually outperform CCHSPMD. It shows that the distributed controlled DLB algorithm outperforms the centralized controlled DLB algorithm in THIN systems. ② In LOWEST, to minimize the load influence to an LLN due to the migrated processes, the lightest one is chosen, but the migration distance is ignored. Unlike LOWEST, in DASUD and THINDLBA, the request message derived from an HLN is transferred layer by layer, and the first located LLN is the nearest. So, DASUD and THINDLBA outperform LOWEST. ③ The common feature of LOWEST and DASUD is that only one LLN, either the lightest or the nearest, is chosen for load migration, and the number of migrated processes is not considered. So, they will be executed many times if the LLN cannot accept all the excess processes. Unlike them, in THINDLBA, the route of searching for an LLN (branch of the multicast tree) can be expanded when the searched LLN cannot accept all the overloaded processes. This assures that the excess processes can be migrated to the LLNs in a one time execution of THINDLBA as many as possible. Meanwhile, multicast tree is adopted to decrease the costs due to interchanging DLB messages. So, THINDLBA outperforms DASUD.



**Fig. 4** Time curves of algorithms. (a) Matrix multiplication in level-2 THIN system; (b) Matrix multiplication in level-3 THIN system; (c) Finding prime number in level-2 THIN system; (d) Finding prime number in level-3 THIN system

## 5 Conclusion

We present a THIN system-oriented DLB algorithm, THINDLBA, which is distributed controlled, sender-initiated, and adopts multicast tree technology to decrease the time cost due to interchanging DLB messages. Through experiments, the effectiveness of THINDLBA in comparison with three other DLB algorithms is verified. The results show that THINDLBA can help THIN systems achieve higher performance in dealing with large scale compute-intensive tasks.

## References

- [1] Shi Feng, Ji Weixing, Qiao Baojun, et al. A triplet based computer architecture supporting parallel object computing [C]//*Proceedings of IEEE the 18th International Conference on Application-Specific Systems, Architectures and Processors*. Montreal, Canada, 2007: 192 – 197.
- [2] Shi Feng, Ji Weixing, Qiao Baojun, et al. A new non von Neumann architecture TriBA [J]. *Transactions of Beijing Institute of Technology*, 2006, **26**(10): 847 – 849. (in Chinese)
- [3] Ji Weixing, Shi Feng, Qiao Baojun, et al. Study on an interconnection network for complex embedded systems [J]. *Chinese High Technology Letters*, 2007, **17**(9): 886 – 890. (in Chinese)
- [4] Dhakal S, Hayat M M, Pezoa J E, et al. Dynamic load balancing in distributed systems in the presence of delays: a regeneration-theory approach [J]. *IEEE Transactions on Parallel and Distributed Systems*, 2007, **18**(4): 485 – 497.
- [5] Kameda H, Fathy El-Z S, Ryu I, et al. A performance comparison of dynamic vs. static load balancing policies in a mainframe- personal computer network model [C]//*Proceedings of the 39th IEEE Conference on Decision and Control*. Sydney, 2000: 1415 – 1420.
- [6] Yang Ming, Yang Yuanyuan. Constructing minimum cost dynamic multicast trees under delay constraint [C]//*Proceedings of the 14th International Conference on Computer Communications and Networks*. San Diego, 2005: 133 – 138.
- [7] Adjih C, Georgiadis L, Jacquet P, et al. Multicast tree structure and the power law [J]. *IEEE Transactions on Information Theory*, 2006, **52**(4): 1508 – 1521.
- [8] Lee Busung. Dynamic load balancing in a message passing virtual parallel machine [R]. Singapore: Division of Computer Engineering of School of Applied Science of Nanyang Technological University, 1995.
- [9] Eager D L, Lazowska E D, Zahorjan J. Adaptive load sharing in homogeneous distributed systems [J]. *IEEE Transactions on Software Engineering*, 1986, **12**(5): 662 – 675.
- [10] Cortes A, Ripoll A, Senar M A, et al. On the performance of nearest-neighbors load balancing algorithms in parallel systems [C]//*Proceedings of the Seventh Euromicro Workshop on Parallel and Distributed Processing*. Funchal, 1999: 170 – 177.

# 基三分层互连网络中基于受限多播树的动态负载平衡

刘 滨<sup>1,2</sup> 石 峰<sup>1</sup> 高玉金<sup>1</sup> 计卫星<sup>1</sup> 宋 红<sup>1</sup>

(<sup>1</sup> 北京理工大学计算机科学技术学院, 北京 100081)

(<sup>2</sup> 河北科技大学经济管理学院, 石家庄 050018)

**摘要:**为了解决基三分层互连网络(THIN)系统中的负载平衡问题,提出一种采用多播树技术提高节点间交换负载信息效率的动态负载平衡(DLB)算法——THINDLBA. 设计了一套完整的 DLB 消息和各节点处的信息维护机制以辅助算法实现. 重载节点的负载迁移请求消息沿着一棵以该节点为根的多播树传播,被该树覆盖的轻载节点均成为负载迁移的候选目标节点,可以沿着该树和重载节点交互负载信息,从而使重载节点能够在算法的一次执行中外迁最多的过载进程,尽快改善自身负载状态. 算法设计中约束了多播树的构造过程,以避免因树间覆盖造成的消息误传或冗余. 通过实验对比了 4 种 DLB 算法的性能,结果证明 THINDLBA 能更有效地缩减 THIN 系统处理计算密集型任务的时间.

**关键词:**基三分层互连网络;动态负载平衡;多播树

**中图分类号:**TP311