Journal of Southeast University (English Edition) Vol. 30, No. 4, pp. 428 - 433 Dec. 2014 ISSN 1003—7985

Load-balancing data distribution in publish/subscribe mode

Li Kai[1,2] Wang Yun[1,2] Yin Yi[1,2,3] Yuan Feifei[1,2]

([1]School of Computer Science and Engineering, Southeast University, Nanjing 211189, China)
([2]Key Laboratory of Computer Network and Information Integration of Ministry of Education, Southeast University, Nanjing 211189, China)
([3]School of Computer Science and Engineering, Nanjing Normal University, Nanjing 210046, China)

Abstract: To improve data distribution efficiency, a load-balancing data distribution (LBDD) method is proposed in publish/subscribe mode. In the LBDD method, subscribers are involved in distribution tasks and data transfers while receiving data themselves. A dissemination tree is constructed among the subscribers based on MD5, where the publisher acts as the root. The proposed method provides bucket construction, target selection, and path updates; furthermore, the property of one-way dissemination is proven. That the average out-going degree of a node is 2 is guaranteed with the proposed LBDD. The experiments on data distribution delay, data distribution rate and load distribution are conducted. Experimental results show that the LBDD method aids in shaping the task load between the publisher and subscribers and outperforms the point-to-point approach.

Key words: data distribution; publish/subscribe mode; load balance; dissemination tree

doi: 10.3969/j.issn.1003 - 7985.2014.04.005

In the publish/subscribe mode, subscribers subscribe to the topics they are interested in and publishers publish relevant data to those subscribers. When publishers have new information, a data distribution procedure is launched to distribute that information to all interested subscribers. The publish/subscribe mode relieves the tight coupling of publishers and their subscribers. With the data distribution service, neither the publisher nor the subscribers need to know the exact locations of the other, which enhances the service's flexibility in adapting to dynamic applications.

To promote transparency between publishers and subscribers, agents are used. These agents play dual roles: on the one hand, they store the global information of both the publishers and subscribers; on the other hand, they actively engage in topic matching and data transfer. Very often, publishers will often need to distribute information to many subscribers. Thus, the multicast technique is suitable for fulfilling this task but, unfortunately, applications are not able to use IP multicast in routers provided by most Internet service suppliers. Furthermore, a publish/subscribe system based on the topics must be in charge of multicast group management. Therefore, the data distribution service adopts a point-to-point approach in order to reliably disseminate data to subscribers.

Due to its simplicity, point-to-point dissemination works relatively well when small amounts of data and few nodes are involved. However, as applications scale up, such processing becomes problematic. With the increase in the number of subscribers and amounts of data, the publisher becomes a bottleneck due to the heavy load and it must send data to subscribers one by one. In addition, most subscribers are in the waiting state (or even starvation) because the publisher sends data sequentially. Taken as a whole, all of this contributes to poor data distribution efficiency.

To solve this problem, this paper proposes a load-balancing data distribution (LBDD) method to disseminate data in parallel by asking subscribers to undertake a part in data transfers. Without extra data transfer costs, all the subscribers receive the topic data and the publisher only needs to send data directly to a small number of the subscribers, thus clearly reducing the sending load.

How to shape the load between publishers and subscribers is the main challenge and will be examined later in this paper. The main contributions of this paper are summarized as follows.

The LBDD method is proposed to support the load balance in a publish/subscribe system. A dissemination tree is organized to guarantee that all the subscribers receive the topic data once and only once. Furthermore, the LBDD is proven to have some important properties, including one-way dissemination and depth control.

Both the empirical and simulation results show that the LBDD method is able to take advantage of the bandwidth among subscribers and to achieve a load balance.

1 Related Work

In a state-of-the-art publish/subscribe system, the overlay network is composed of many specific routers. These routers play the role of node agents; they save subscription information, provide network communication

Received 2014-07-01.

Biography: Li Kai (1979—), male, doctor, lecturer, newlikai@seu.edu.cn.

Foundation item: The National Key Basic Research Program of China (973 Program).

Citation: Li Kai, Wang Yun, Yin Yi, et al. Load-balancing data distribution in publish/subscribe mode[J]. Journal of Southeast University (English Edition), 2014, 30(4): 428 - 433. [doi: 10.3969/j.issn.1003 - 7985.2014.04.005]

among publishing and subscribing nodes, and conduct reasonable data transfers in order. To locate specific nodes, traversal algorithms (such as the flooding algorithm, the matching algorithm, or the gossip and infection algorithm) are usually used[1]. These algorithms apply broadcast, which is a heavy burden when nodes scale up. Passing messages by application multicasts has also been introduced to publish/subscribe systems[2-3]. JEDI[4] constructs an application multicast tree in which each node agent stores only local topological information; a complete multicast tree is built by all the agents and reduces extra broadcasting costs by choosing the proper path within the tree. During tree construction, many messages are required as agents exchange information. This may make the implementation more complex and necessitate high tree-maintenance costs.

Publiy[5] takes advantage of both P2P content dissemination and the publish/subscribe mode that are based on node agents. With Publiy, a block data dissemination strategy is proposed to improve the efficiency of large data block distribution due to node collaboration. In Publiy, publishing nodes obtain some subscribing node lists from agents in several domains. These subscribing nodes behave as seed nodes to generate copies of the transferring data. Thus, multiple source dissemination is set up. However, it is relatively complicated during agent traversal when determining seed nodes.

In P2P-based publish/subscribe systems, each node is both an agent and a client. Generally, these systems construct structured, logical topologies based on a distributed hash table. SCRIBE[6] sets up relationships among nodes and resources to locate resources efficiently. Nodes are categorized by clusters according to their physical distances. Those in a cluster are organized as a chord[7] ring. The most powerful node in a cluster is selected as that cluster's representative. All these representatives form a super cube. High performance issues in publish/subscribe systems are also addressed[8-11].

2 Model

2.1 System model

In the proposed method, the participants of a publish/subscribe system are categorized into two types: users (including publishers and subscribers) and agents. Users either publish or subscribe to data. Agents are in charge of subscription information maintenance and data transfer. Each node is equipped with an agent and may have several users. Thus, a node behaves as both a user and an agent and a user on a node connects to the local agent. All the users on a node are managed by the same local agent.

A publish/subscribe system has n nodes, denoted by $PS = \{N_1, N_2, \ldots, N_n\}$. N_i maintains all the publishing and subscription information in the system, which is denoted by $P_i = \{p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n\}$, in which $p_j (j \neq i)$ represents a triple $(s_{ij}, \text{pub}_j, \text{sub}_j)$. The parameters s_{ij}, pub_j, sub_j stand for the link between N_i and N_j, the publishing topic set of N_j, and the subscription topic set of N_j, respectively.

All the agents are fully connected. When a user N_i subscribes to a topic t, it sends the subscription message to its local agent. The agent then informs all other agents in a flooding way. All the agents save all the subscription information and maintain links among themselves. An agent determines whether other agents are alive or not depending on periodic heartbeat messages to maintain data consistency. When a user N_i publishes the data of topic t, it also sends that data to its local agent. Upon the determination of the subscriber destination set, the topic data is transferred among the related agents until all the subscribers of that topic receive the data.

2.2 Problem statement

For a given set of one publisher and multiple subscribers, the publisher needs to disseminate data to the subscribers. Therefore, find a solution to effectively alleviate the publisher's task load and to guarantee that all subscribers receive the required data.

3 Load-Balancing Data Distribution

3.1 Overview

To send data to a given set of subscribers with the LBDD, all the subscribers are first mapped to logical buckets. A topic data t flows in terms of the order of these buckets. Therefore, a unidirectional property holds, which guarantees that all the subscribers will receive the data. Globally, a tree is constructed to disseminate t. Therefore, the LBDD is composed of three parts: 1) Bucket construction, which sets up buckets based on a logical distance; 2) Target selection, which provides a way for an agent to locally select its destination; and 3) Path update, which allows an agent to re-compute a path in case of node failure, joining, or leaving.

3.2 Bucket construction

For any node N_i, its IP address is hashed to a 128-bit sequence with MD5. For simplicity, the highest 32 bits are selected as the sequence ID Digesti for N_i. Due to MD5's features, the sequence IDs maintain their randomness. Thus, all nodes are mapped to a new 32-bit address space S. The distance between any two nodes N_i and N_j is calculated as $\text{distance}(N_i, N_j) = \text{digest}_i \oplus \text{digest}_j$.

Considering all the nodes publishing and subscribing to topic t, we assume that the source node is in bucket 0. Any node N_i is in bucket m if and only if the distance between N_i and the source node is in the intersection of $[2^{m-1}, 2^m)$. N_i locally puts all the nodes subscribing to t into buckets. N_i constructs an ordered sequence of buck-

ets, i. e., Bucket = {bucket$_1$, bucket$_2$, …, bucket$_x$}, with the distance between them monotonically increasing. N_i itself is in the bucket SelfNo ($0 \leqslant \text{SelfNo} \leqslant \log_2 n$). A node N_j will be inserted into the local bucket sequence in N_i if and only if bucketNo (the bucket rank of N_j) > SelfNo holds. According to the proposed method, only nodes with a longer distance to the source node can be potential candidates for N_i's next hop; this is reflected in the above-mentioned rules regarding which nodes are inserted into the local bucket sequence in N_i. The processing procedure in N_i is listed in Algorithm 1.

Algorithm 1 BucketConstruction(P_i, t)
Input: P_i; topic t.
Output: The bucket rank of N_i.
SelfNo = ComputeBucketNo(s_Digest, SelfDigest); //computing bucket rank for itself
for (peer ∈ P_i) {
if peer. TestSubscribe(t) = = true{
BucketNo = ComputeBucketNo(s_Digest, peer. Digest)//filter the nodes not subscribing to t
if BucketNo = = SelfNo{ //in the same bucket as N_i
if peer. digest < SelfDigest //update the bucket rank of N_i
rank + +; }
else if BucketNo > SelfNo //insert the new node into a bucket with larger rank
InsertPeerToBucket(peer, BucketNo); }
}
return rank;

3.3 Target selection

Using local information, N_i independently computes and puts subscribers for topic t into corresponding buckets. In this section, we set up some rules for N_i's selection of nodes to go into the next bucket as its Dest$_i$ for the next hop. N_i is in the bucket SelfNo.

Algorithm 2 SelectTargets (SelfBucket)
Input: The specific bucket.
Output: The arranged intersection of nodes as the destination nodes schedule.
TargetBucket. sort(); //nodes in the bucket are ordered in increasing order
Schedule = nil; //initiate the arrangement
ratio = TargetBucket. size()/SelfBucket. size();
for rank = 1 to SelfBucket. size()
{LowBound = ratio × (rank − 1) + 1;
UpBound = ratio × rank; //determine the boundary of an intersection
Add the intersection to the schedule; }
End for
return schedule;

More often than not, more than one node is in the bucket SelfNo. Nodes in the bucket SelfNo are responsible for transferring data to a subset of Dest$_i$. Therefore, a mapping relationship f is formed between the nodes in the buckets SelfNo and Dest$_i$. That is, f : rank→Dest$_i$. Usually, nodes evenly allocate Dest$_i$. The processing procedure for this is described in Algorithm 2.

3.4 Path update

Usually, a node may join or leave the subscription set for topic t. Any such change leads to path updates. Each node periodically sends its heartbeat message to the others. If N_i checks a heartbeat message timeout, it then updates its local P_i and deletes the registration and subscription information of the failed node. A node failure is equal to that node unsubscribing from all topics. Therefore, a failed node is removed from all paths and all paths have to be restructured to maintain connectivity.

If a new user wants to join, his/her agent broadcasts its information to all other agents. Each agent executes Algorithm 3 to update its own path.

Algorithm 3 PathUpdate(Peer$_{\text{fail}}$)
Input: The failed peer.
BucketNo = ComputeBucketNo(source_Digest, peer$_{\text{fail}}$. Digest);
If BucketNo > SelfNo
{Delete peer$_{\text{fail}}$ in BucketNo;
SelectTargets(BucketNo);
return;

3.5 Load-balancing property

In the LBDD method, the subscribers also participate in data transfers as intermediate nodes. Each node is able to find a path from the source node. Clearly, path depth and node degrees of intermediate nodes affect data dissemination efficiency.

3.5.1 One-way dissemination

Theorem 1 All paths form a tree with the root node of a publisher.

Proof 1) There is a path between the publisher and any node N_i in bucket m.

This is true because there are finite buckets x ($0 \leqslant x < m-1$, $x \in \mathbf{Z}$) between the publisher and N_i. One and only one node in each bucket is selected to be an ancestor of N_i. All such nodes plus the publisher and N_i form a path from the publisher to N_i.

2) There is no loop in any path.

With Algorithm 1, N_i only concerns nodes with larger bucket ranks. Such nodes are potential destination nodes for N_i. This requires the LBDD method to maintain a unidirectional property, i. e., from a smaller bucket rank to a larger bucket rank.

All nodes except the publisher have only one father node.

According to Algorithm 2, Dest$_i$ is cut into several disjoint subsets. Each node in the bucket SelfNo is mapped to only one subset. Therefore, a node in Dest$_i$ has only one father node.

In summary, all paths share the publisher as the source node, and all paths form a tree.

Corollary 1 The data dissemination is one-way because all the nodes form a dissemination tree.

3.5.2 Depth control

Theorem 2 With the LBDD method, the average out-going degree of a node is 2.

Proof The IP address of a node is hashed to a 32-bit new address space by MD5. Thus, the distance from N_i to the source node, denoted by distance (source, N_i) is also a 32-bit random number. Without the loss of generality, we assume that distance (source, N_i) $= a_{31}a_{30}\ldots a_0$, in which $a_k = 0$ or 1 and $0 \leqslant k \leqslant 31$. Since $a_{31}a_{30}\ldots a_0$ is a random number, for $\forall k \in [0, 31]$, there is $P(a_k = 1) = 0.5$.

According to the relationship between distance (source, N_i) and bucket rank, N_i is in the bucket with a rank of $n(0 \leqslant n \leqslant 32)$ if and only if the highest non-zero bit is a_{n-1} in $a_{31}a_{30}\ldots a_0$.

Therefore, the probability of N_i being in the bucket n is as follows:

$$P_n = P(a_{n-1} = 1)\prod_{k=n}^{31} P(a_k = 0) \quad (1)$$

The mathematical expectation of the number of nodes in bucket n is as follows:

$$E_n = NP_n = NP(a_{n-1} = 1)\prod_{k=n}^{31} P(a_k = 1) \quad (2)$$

For the same reason, the mathematical expectation of the number of nodes in bucket $(n+1)$ is as follows:

$$E_{n+1} = NP(a_n = 1)\prod_{k=n}^{31} P(a_k = 1) \quad (3)$$

$E_{n+1}/E_n = 2$ holds, which means that, theoretically, the number of nodes in bucket $(n+1)$ is two times that in bucket n. Thus, the average out-going degree of a node in bucket n is 2.

4 Experiments and Analysis

4.1 Experimental settings

The experimental environment is composed of twelve PCs and two routers. Each PC is a Lenovo Yangtian T2900d, equipped with Pentium(R) Dual-Core E6700 @ 3.20 GHz and Marvell Yukon 88E8057 PCI-E Gigabit Ethernet controller. Each router is a D-Link DES1008A with 24-Port Gigabit Ethernet Switch. Furthermore, each PC is equipped with an agent. Applications connect to their local agents. For simplicity, there is one PC acting as publisher in the environment. Other PCs work as subscribers. Therefore, a one-to-many data distribution structure is set up.

Two methods are explored in the experiment. One is LBDD. The other is point-to-point. The data amount for both is 8 GB. The data is sent in slices with a slice size of 64 KB. Data distribution rate, delay, and load distribution are investigated.

4.2 Experimental results and analysis

1) Data distribution delay

Twelve PCs are involved in the experiment. Their IP addresses are *.74, *.97, *.234, *.90, *.98, *.64, *.61, *.184, *.72, *.88, *.44 and *.68, respectively, where * stands for the common IP prefix 10.3.17.

All nodes are fully connected. In the point-to-point approach, the network topology is shown in Fig. 1(a). In the LBDD method, after operating Algorithms 1 and 2, the network topology is set up as shown in Fig. 1(b).

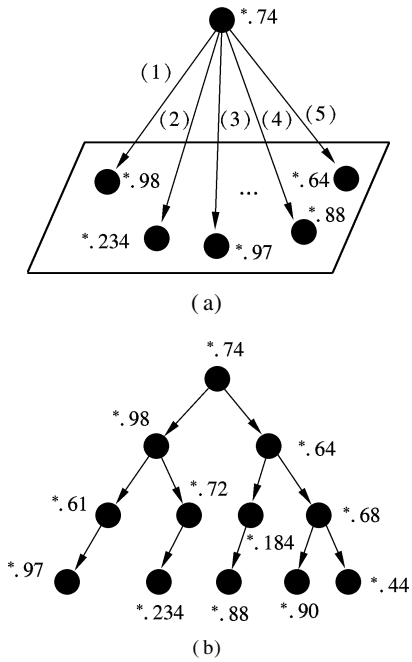

Fig. 1 Topology. (a) In point-to-point mode; (b) In load-balancing mode

In the point-to-point approach, the data distribution delay from the publisher to all subscribers lasts 7 810 s; in the LBDD method, the delay is 3 020 s. Since more subscribers are involved in coordinating data dissemination and improving parallelism in the LBDD method, the delay is significantly reduced.

2) Data distribution rate

The data distribution rate is evaluated by the average amount of data sent in one second. As the number of subscribers increases in the point-to-point approach, the data distribution rate decreases because messages are sequentially sent by the publisher to one subscriber at a time. With the LBDD method, the data distribution rate remains around 24 Mbit/s, as shown in Fig. 2.

3) Load distribution

In the point-to-point approach, it is the publisher's task to send the data to all subscribers. Therefore, the distribution load is on the publisher. As shown in Fig. 3, when

there are six subscribers, the network load for the publisher reaches 100%.

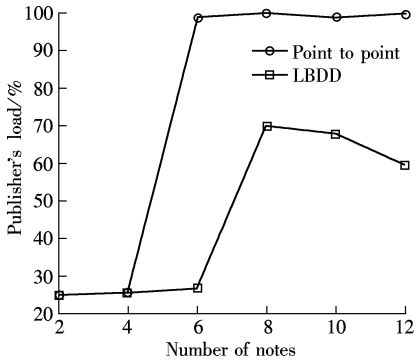

Fig. 2 Data distribution rate

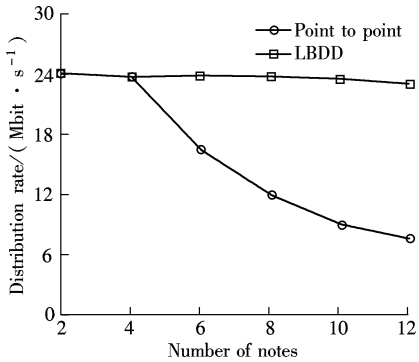

Fig. 3 Publisher's load distribution

With the LBDD method, the network load of the publisher increases still as the number of subscribers increases. However, the network load is always under 70%, which shows that the network load is effectively controlled, thus lightening the publisher's load.

4.3 Simulation test

To investigate LBDD on a large scale, a simulation test is conducted. Several hundred IP addresses are randomly generated. The publisher sends 8 GB of data to the subscribers.

When n is 500, the subscribers are mapped to the buckets indexed from 24 to 32; this reflects the depth of the paths. Based on simulation results when n is from 100 to 500, as shown in Tab. 1, the depth of a data distribution tree is at the intersection of [5, 9], which is a reasonable depth value.

Tab. 1 Path depth and number of subscribers

Subscribers	Path depth
100	5
150	5
200	6
250	7
300	8
350	8
400	8
450	9
500	9

As n subscribers need to receive specific topic data, the minimum n copies of the data have to be sent. With the constructed dissemination tree, exactly n copies are sent. According to the copies a node sends, the contribution of N_i can be easily calculated as follows: Contribution rate$_i$ $= 1/n \times$ the number of copies sent by N_i.

Fig. 4 shows the contribution rates of the publisher and subscribers. As the number of subscribers increases, the load of data distribution undertaken by the publisher almost stays the same and, thus, occupies a lower percentage of the contribution. At the same time, the subscribers collaborate and complete the data distribution. The task load is decomposed by the subscribers, which prevents the publisher from becoming a bottleneck during processing.

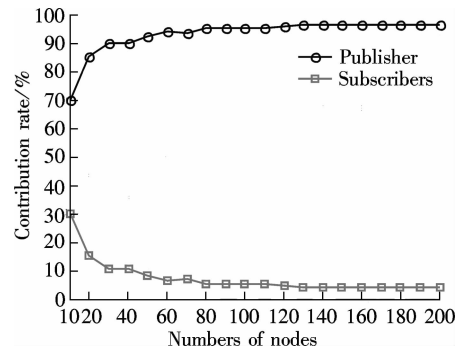

Fig. 4 Contributions by publishers and subscribers

We should point out that, if the number of subscribers is less than six, the point-to-point approach is able to maintain reasonable good data distribution efficiency. However, its efficiency decreases rapidly as the number of subscribers increases.

5 Conclusion

Data distribution services are widely applied in publish/subscribe systems. Regarding service efficiency, the point-to-point approach is not acceptable if there are many subscribers for a specific topic. The main reason for this is that the publisher has to send data to subscribers sequentially, which requires that a portion of the subscribers wait until they are able to contribute. Thus, a method allowing subscribers to be involved in distribution is proposed and the LBDD is explored. The experimental results show that the LBDD aids in shaping the task load between the publisher and subscribers. In future work, we will explore a load-balancing strategy, in which node load is dynamically and dramatically changed.

References

[1] Boyd S, Ghosh A, Prabhakar B, et al. Gossip algorithms: design, analysis and applications[C]//*Proc of INFOCOM*. Miami, USA, 2005: 1653 – 1664.

[2] Fateri S, Ni Q, Taylor G A, et al. Design and analysis of multicast-based publisher/subscriber models over wireless

platforms for smart grid communications[C]//*Proc of IEEE* 11*th International Conference on Trust, Security and Privacy in Computing and Communications* (*TrustCom*). Liverpool, UK, 2012: 1617 - 1623.

[3] Cui J, Xiong N, Park J H, et al. A novel and efficient source-path discovery and maintenance method for application layer multicast[J]. *Computers & Electrical Engineering*, 2013, **39**(1): 67 - 75.

[4] Cugola G, Nitto E D, Fuggetta A. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS[J]. *IEEE Transactions on Software Engineering*, 2001, **27**(9): 827 - 850.

[5] Kazemzadeh R S, Jacobsen H. Publiy +: a peer-assisted publish/subscribe service for timely dissemination of bulk content[C]//*Proc of IEEE* 32*nd International Conference on Distributed Computing Systems*. Macau, China, 2012: 345 - 354.

[6] Rowstron A, Kermarrec A M, Castro M, et al. SCRIBE: the design of a large-scale event notification infrastructure [C]//*Proc of the Third International COST*264 *Workshop, NGC* 2001. London, UK, 2001: 30 - 43.

[7] Stoica I, Morris R, Karger D, et al. Chord: a scalable peer-to-peer lookup service for internet applications[C]// *Proc of ACM SIGCOMM*. San Diego, CA, USA, 2001: 149 - 160.

[8] Esposito C, Cotroneo D, Russo S. On reliability in publish/subscribe services[J]. *Computer Networks*, 2013, **57**(5): 1318 - 1343.

[9] Zhao Y, Wu J. Building a reliable and high performance publish/subscribe system[J]. *Journal of Parallel and Distributed Computing*, 2013, **73**(4): 371 - 382.

[10] Diallo M, Sourlas V, Flegkas P, et al. A content-based publish/subscribe framework for large-scale content delivery[J]. *Computer Networks*, 2013, **57**(4): 924 - 943.

[11] Shen L, Shen H, Sapra K. RIAL: resource intensity aware load balancing in clouds[C]//*Proc of IEEE INFOCOM*. Toronto, Canada, 2014: 1294 - 1302.

发布/订阅模式下面向负载均衡的数据分发

李 凯[1,2] 汪 芸[1,2] 殷 奕[1,2,3] 袁飞飞[1,2]

([1] 东南大学计算机科学与工程学院,南京 211189)

([2] 东南大学教育部计算机网络与信息集成重点实验室,南京 211189)

([3] 南京师范大学计算机与技术学院,南京 210046)

摘要: 为了提高数据分发效率,在发布/订阅模式下提出了一个面向负载均衡的数据分发方法 LBDD. 在 LBDD 方法中,订阅方既接收数据,又承担数据转发工作. 采用 MD5 算法,在发布方和订阅方间建立一棵分发树,其中发布方是根节点. 给出了桶建立、目标选择以及路径修正方法,并进一步证明了数据单向分发性质. LBDD 方法可保证分发树中任意一个节点的平均出度为 2. 针对数据分发延迟、数据分发速率和负载分布进行了实验. 实验数据表明,LBDD 方法能够有效地均衡发布方和订阅方的负载,分发效率高于点到点分发方式.

关键词: 数据分发;发布/订阅模式;负载均衡;分发树

中图分类号: TP391