ISSN:0975-9646

A New Approach to Event Dissemination in Distributed Systems

Anson Antony Fertal[#1], P. Priya Ponnusamy[#2]

[#]*Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology L&T Bypass Road, Coimbatore, India*

***Abstract*— An approach for efficient event dissemination in a distributed system for communication between the entities of the system. The publish/subscribe systems are widely used for event dissemination, which provide loose coupling between publishers and subscribers, but the routing of events to corresponding subscribers involves the issues of scalability and load balancing with a large number of publishers and subscribers. The approaches that have been adopted to deal with these issues brings with it the need for addition or utilization of more resources such as memory, bandwidth, etc. The proposed system is an attempt to balance the utility and utilization of resources thereby providing maximum benefits with minimum utilization of resources using techniques such as attribute and locality based grouping of publishers and peer-to-peer model based on ring and binary-tree structures for scalability and load balancing.**

***Keywords*— Publish/subscribe, peer-to-peer, broker-less, binary tree structure, cloud computing.**

I. INTRODUCTION

Communication between entities in a distributed system is one of the most important aspects that keep it alive and consistent. With the emergence of cloud computing, the communication and coordination among entities have become ever more a deciding factor in the quality of service provided. Among the various communication paradigms, publish/subscribe communication paradigm has proved to be very effective due to its inherent decoupling of publishers from subscribers in terms of time, space, and synchronization. Publishers are a group of entities that generate events in the system, which are of interest to another group called subscribers. The events are forwarded to interested subscribers. The decoupling is evident in that the publishers and subscribers are unaware of each other. The number of publishers and subscribers could be a huge number and they can join the system, perform functions, and leave the system without being aware of each other. While this lends a great flexibility, it brings with it the issue of routing the events to the interested subscribers. A single broker obviously would bog down as the number of entities on either side increases. Also, the single broker is a potential single-point of failure. To overcome this, in later works, a network of brokers has been employed for load balancing and fault tolerance. This approach brings with it the problems of security and overhead of introducing another layer of resources in between to reap the benefits of the pub/sub paradigm.

Further improvements have made use of broker-less environments, where subscribers form a peer-to-peer overlay to handle the dissemination of events through the network. Various protocols and structures have been followed to arrange the peers to handle event dissemination in a scalable, load-balanced, and fault tolerant manner, which involves additional communication overhead between the peers in addition to event forwarding.

This paper tries to address these problems in an efficient manner by providing maximum benefit with very little additional overhead in terms of memory and bandwidth utilization. Publishers and subscribers are grouped based on attributes, which in the proposed system has been adopted as the 'event type' for simplicity. Furthermore, the publishers and subscribers are separated based on locality into user-defined regions. Publishers and subscribers first register with a local server, which assigns them to groups based on their attributes and publisher groups and subscriber groups are matched with a one-to-one relation. After registrations, members of each publisher group are made aware of a finite set of subscribers whose group match with the corresponding publishers.

For event dissemination, the event destined for a particular group is forwarded to the finite set of subscribers assigned to a publisher that form a ring-like structure around the event being generated. These subscribers then form a binary tree of subscribers with them acting as root nodes and each node in the tree is responsible for forwarding the events to only two children.

Fault detection and recovery is addressed by heartbeat among neighbouring nodes at each level, which follows a clockwise direction and introducing an additional path of event dissemination, which is the left child of the failed node in case of clockwise detection and right child in case of anticlockwise detection. Event dissemination on this additional path triggers event forwarding by the node that received the event to the neighbouring node to the right in addition to normal forwarding to its children.

II. SYSTEM ARCHITECTURE

The main components in the proposed system architecture are:

1. Publisher Groups.
2. Subscriber Groups.
3. Registration and Membership/Group Generation Servers.

Another concept used is the notion of separation of these three basic components based upon user-defined regions. The process of communication between these user-defined regions is not addressed in this paper. The high-level architecture is as shown in Figure 1. The publishers and subscribers register with the local server in their respective regions whose IPs are publicized to them.

A. *Attribute-Based Pub/Sub Grouping*

Attribute-based grouping uses common attributes of entities to form groups. The set of attributes for an entity may be represented by the set S = {A_1, A_2, …, A_n}, where each Ai represents an attribute. In the proposed system, for simplicity, event of interest for subscribers and type of event injected into the system by publishers are matched for attribute-based grouping. Publishers are grouped according to the type of events to be generated by them and subscribers by the events to be consumed by them. Let E be the event space represented by {e_1, e_2, ..., e_m}, where e_i represents each unique event. Publisher and subscriber groups are matched with each other based upon the events to be generated and consumed. In the simplest case, each publisher generates exactly one type of event and each subscriber consumes exactly one type of event. In this case, the number of groups generated will be the same as the number of events in the event space, i.e., the set of groups, G = {G_1, G_2, ..., G_m}. For each group, there are a set of publishers and subscribers represented by {P_1, P_2, ..., P_n} and {S_1, S_2, ..., S_n}. The number of groups that could potentially be formed depends on the event space and the subsets that could be formed out of a given event space. The number of groups could be very large since given an event space of 'm' elements, the number of possible subsets that could be formed excluding the null set is 2^{m-1}. However, this creates only the overhead of generating groups, but the number of entities will remain the same always and the aim of this paper is not to devise an efficient algorithm for generating groups but to handle any number of entities participating in the system and disseminate events in a lightweight, structured, and fault-tolerant fashion. Also, the aim of grouping is to avoid unnecessary messages to groups that are not interested in those messages.

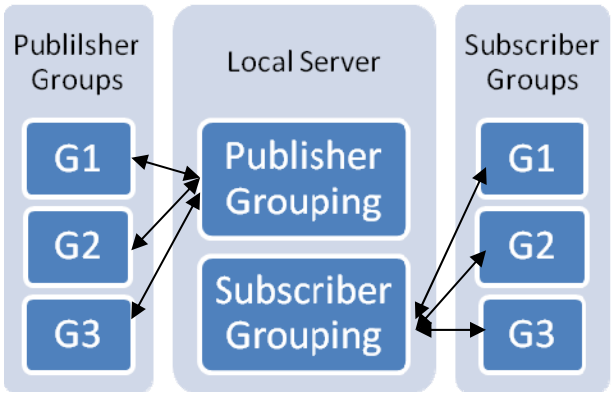

Fig. 1. Pub/Sub Registration and Grouping

B. *Publisher and Subscriber Group Allocation*

During registration, the local server assigns the publishers and subscribers to their respective groups. Once adequate number of registrations has been made in a subscriber group corresponding to a finite number greater than 'm', the first 'm' members will be made known to the matching publishers. The number 'm' is determined as m = $\log_2(n)$, where 'n' is the number of publishers. So, if there are 16 publishers, m will be equal to 4. These nodes will form a ring around the event as shown in Figure 2.

Each entity is uniquely identified by (IP,Port) combination with grouping based on event type. So, a publisher entity P could be represented by {P:((IP,Port),event_type,region))}.

C. *Peer-to-Peer Assignment Strategy*

Publishers are assigned neighbors to whom events are to be forwarded at the first ring or level equal to the number m=$\log_2(n)$, where n is the number of publishers in a particular group. This is shown in Figure 2.

Each subscriber in the ring around the published event is assigned two children from their respective group recursively to form a binary tree and each node in the tree is responsible for forwarding the event only to its two immediate children. This is shown in Figure 3.

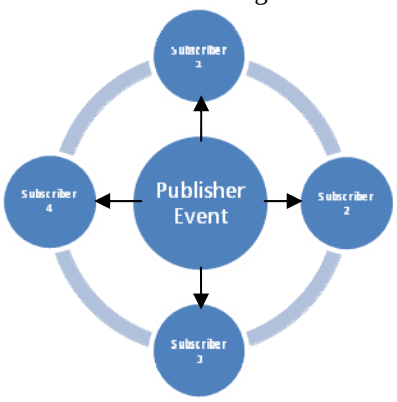

Fig. 2. Event Dissemination Initiation

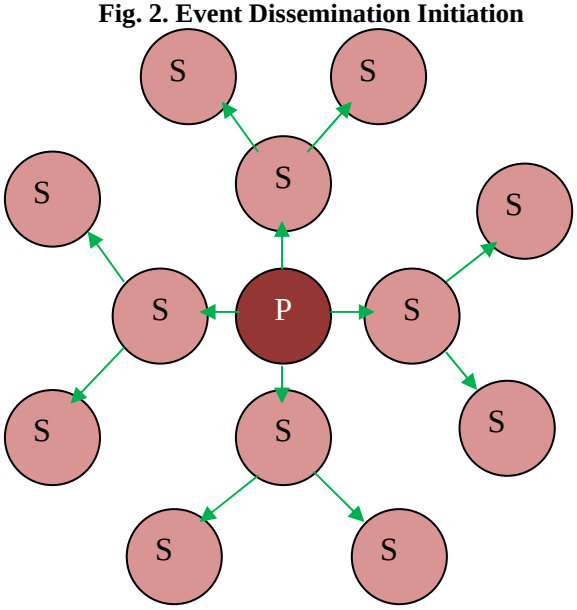

Fig. 3. Event Dissemination Model

III. Event Dissemination Model

When an event is generated, the node generating the event passes it on to its neighbors in the ring around it. These neighbor nodes pass it on to its immediate children. This event dissemination process continues at all nodes in each ring following the binary tree structure until the leaf nodes or the nodes in the outermost ring are reached. This is depicted in Figure 3.

A. Fault Detection and Recovery

The aliveness property of nodes in the system is kept track of by heartbeat messages by a node to another node in the same ring following a clockwise direction.

An alternate path for event forwarding is taken by the detecting node on behalf of the failed node. The event dissemination is maintained without failure by forwarding the event by the detector node to the left child of the failed node, which is in the next ring, in addition to its child nodes.

For added fault tolerance capabilities, a second scenario is also considered. Each node knows about its left and right neighbours in the same ring. If a node does not receive a heartbeat from the immediate neighbour on the left side, it follows the same strategy as in the first scenario except that it forwards the event to the right child of the failed node towards its left in its ring.

Each node also knows about its parent. So, in the event of receipt of a message from a node other than the parent, it also forwards the message to the neighbouring node towards its right in the clockwise direction; if the message has not already been received, the message is accepted, otherwise rejected.

B. Maintenance of the System Structure

The local server in each region maintains a list of subscribers for each group, which reflects the system structure. The list is updated according to the failure detection messages from the nodes in the system. A node which fails creates an empty position in the list and the first newly joining node acquires that position. When a failed node rejoins at a later time, it is given a leaf node position.

IV. Approach Overview And Analysis

The approach followed for the proposed system is a combination of techniques of clustering based on locality and attributes to provide a clear and structured way to separate dissimilar entities and group similar entities and peer-to-peer topology and binary tree structures for event dissemination. Approach for fault detection is the use of heartbeat messages with neighbouring nodes. The approach has been proposed with a dynamic and wireless environment in mind, though it is applicable to a wired environment with well-defined structure like data centre environments providing cloud computing services. A step by step analysis of each of the goals of the approach is given in the following paragraphs.

A. Restricting Messages within a Group and Avoiding Single Point of Failure

With publisher and subscriber groups matched, it is guaranteed that events are forwarded only to matching groups. Since grouping and assignment of neighbors is done by the server local to the respective publisher and subscriber regions, there is no overhead on the part of publishers and subscribers for this function. These entities just register with the local server with their corresponding attributes. Although a complete decoupling between publishers and subscribers is not achieved, still a publisher in a group only needs to know about $\log_2(n)$ subscribers in the group, where 'n' is the total number of publishers in the group. The number of nodes in a ring also avoids a single point of failure and prevents complete loss of a message.

B. Load Balancing

This system is inherently load balanced, as each node in a ring is responsible only for forwarding the event to two child nodes when in normal operation. In case of a failure of a neighbouring node, a node will be responsible for forwarding the event to a maximum of only one more mode, namely the left child of the failed node following the clockwise direction or the right child of the failed node following the anti-clockwise direction. Similarly, a node receiving an event from a node other than its parent is responsible for forwarding the event to a maximum of only one another node, namely its nearest neighbour towards the right.

C. Scalability

Scalability in terms of the load incurred on a particular node due to growth in the number of entities joining the system is not an issue, as irrespective of the number of nodes joining the load on a node in the rings is constant at a maximum of 3. In terms of the time taken for the event to reach the leaf nodes, it increases only logarithmically with increase in the number of nodes, which again is well known to be good in terms of scalability.

D. Fault Tolerance

The approach for fault tolerance is of heartbeats for fault detection and uninterrupted message forwarding is achieved through selection of alternate paths. Fault detection is tried to be kept as simple as possible with each node detecting the failure of its immediate neighbouring nodes and recovery path being the left child of the failed node in case of clockwise detection and right child of the failed node in case of anti-clockwise detection. As an additional step to achieve a higher degree of fault tolerance, any node that receives an event from another node other than its parent, forwards the event also to its nearest neighbour towards the right. These three steps can be seen to provide fault tolerance even in a scenario where more than 50% of the nodes in a ring fail, which can be considered a rare occurrence.

The three main aims of the approach, namely scalability, load balancing, and fault tolerance is addressed in the algorithms and analysis shows that these benefits are achieved without much overhead in terms of memory or processing power at each node. With the approach taken for fault tolerance, there is an issue of duplication of messages, which is addressed by ordering of messages through sequencing and each node keeping track of messages received recently, which allows it to reject duplicate messages.

V. Conclusions And Future Work

In this paper, the proposed system is presented as an efficient way of disseminating events to entities waiting for those events in a scalable, load-balanced, and fault-tolerant fashion in a distributed environment like a data centre, a sensor network, etc. Analysis shows that the system is highly scalable in terms of the time required for disseminating the events to all the nodes in the system as well as the load on each entity in the system and bandwidth consumption. The fault tolerance strategy adopted is seen from the analysis to tolerate a complete failure in a particular ring except for a single node, which is near to ideal. The idea of grouping based on attributes and locality poses some challenges since attributes may vary depending upon what entities are involved in the distributed system and the purpose of the system.

In future work, it is planned to address these issues and devise efficient algorithms for event dissemination and fault tolerance at the desired levels of performance proposed in the paper.

References

[1] Ming Li, Fan Ye, Minkyong Kim, Han Chen, and Hui Lei. BlueDove: A Scalable and Elastic Publish/Subscribe Service.

[2] Xiaoyu Yang, Yingwu Zhu, and Yiming Hu. Scalable Content-Based Publish/Subscribe Services over Structured Peer-to-Peer Networks.

[3] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In SIGCOMM ’01.

[4] The Gnutella Protocol Specification v0.4, Document Revision 1.2.

[5] Robbert van Renesse, Yaron Minsky, and Mark Hayden. A Gossip-Style Failure Detection Service.

[6] Abhinandan Das, Indranil Gupta, and Ashish Motivala. SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol.

[7] J. Nagarajarao, R.E. Strom, and D.C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In ICDCS ’99, 1999.

[8] B. Y. Zhao, J. D. Kubiatowicz, and A.D. Joseph. Tapestry: An infrastructure for fault-tolerance wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley, April 2001.

[9] O.D. Sahin, A. Gupta, D. Agarwal, and A.E. Abbadi. Meghdoot: Content-based publish/subscribe over p2p networks. In ACM/IFIP/USENIX 5th International Middleware Conference, Toronto, Ontario, Canada, Oct. 2004.

[10] K. P. Birman. The process group approach to reliable distributed computing. Communications of the ACM, 36(12):36-53, Dec. 1993.

[11] G. Perng, C. Wang, and M.K. Reiter. Providing content-based services in a peer-to-peer environment. In Proceedings of the third International Workshop on Distributed Event-Based Systems (DEBS), pages 74-79, Edinburgh, Scotland, UK, May 2004.