Proceedings of the 5th WSEAS International Conference on Applied Computer Science, Hangzhou, China, April 16-18, 2006 (pp451-457)

Clustering mechanism for content-based system in DHT-based overlay networks

EMILIANO CASALICCHIO[1], FEDERICO MORABITO[2], FABRIZIO DAVIDE[2], GIOVANNI CORTESE[2]
University of Roma "Tor Vergata", Computer Science Department[1]
Via del Politecnico 1 - Rome[1]
Telecom Italia Learning ServicesDepartment[2]
Viale Parco de Medici 37, Rome[2]
ITALY

Abstract: A generic pub/sub communication system (often referred to in the literature as *Event Service* or *Notification Service*) is composed of a set of nodes distributed over a communication network. Clients to the systems are divided according to their role into publishers, which act as producers of information, and subscribers, which act as consumers of information. Clients are not required to communicate directly among themselves but they are rather *decoupled*: the interaction takes place through the nodes of the pub/sub system. There are several architectural options and subscription models for publish-subscribe communication. We focus on content-based publish subscribe, and we explore an architectural options for realizing the broker overlay, namely peer-to-peer structured. We propose a novel adaptive content-based subscription management system, relying on a Distributed Hash Table routing infrastructure. We define a model for the event space guaranteeing the expressiveness for any application domain. Also we provide mechanism to dynamically identify groups of users with similar preferences (multicast group) based on clustering algorithms for the users preferences.

Key-Words: - Content-based Publish/Subscribe, Multicast, Peer-to-Peer system, DHTs, clustering

1 Introduction

Content-based Publish/Subscribe (CBPS) interaction paradigm is suitable for a variety of large scale dynamic applications: news delivery, stock quoting, air and metropolitan traffic control, on-line games, dissemination of multimedia contents, dissemination of auction bids, services and resources discovery, remote control of critical infrastructures and management of large scale systems. In contrast to their flexibility and expressiveness, scalable CBPS systems are difficult to implement and the proposed solutions are not again mature. The publish/subscribe interaction paradigm provides subscribers with the ability to express their interest in an event or a pattern of events, in order to be notified subsequently of any event, generated by a publisher, that matches their registered interest. The different ways of specifying the events of interest have led to several subscription schemes, the channel-based, the topic-based (or subject-based), and the content-based.

The publish/subscribe scheme consists of a set of nodes that asynchronously exchange notifications decoupled by a notification service that is interposed between them [1]. When a notification is produced by a subscriber, it expresses node' interest in receiving an event, or a pattern of events. The publisher produces an event, that is asynchronously propagated to all subscribers that registered interest in that given event. The pub/sub paradigm provides 1) to the subscribers 1a) the ability to express their interest (subscription) and 1b) to be notified of any event that matches the subscription and 2) to the publishers to publish information (event) that will be efficiently distributed (notification) to whose nodes that are interested in receiving.

The main characteristic of this event-based interaction style lies in the full decoupling in time, space and synchronization between publishers and subscribers. The publishers do not need to address the subscribers and vice versa, facilitating to add, mode or remove new subscribers and publishers. The communication among the nodes is inherently asynchronous, decoupling the production and consumption of information, with a relevant increase in terms of scalability and ability of locally computing (subscribers can perform concurrent jobs, while waiting for an event). Publishers and subscribers do not need to be available at the same time: a subscription causes notifications to be delivered even if producers join after the subscription was issued. The communication initiated by the effective publishers of information and it is full event/information-driven.

2 Problem Formulation

Publish/subscribe systems exhibit a lot of interesting challenges, but a fundamental aspect in any publish/subscribe system is the expressiveness of the notification selection, i.e., how consumers specify subscriptions. Choosing the notification selection mechanism is perhaps the most important (and most difficult) choice to be made when developing a notification service. Exists different ways of specifying the events of interest that have led to several subscription schemes, that are topic-based and content-based publish/subscribe subscription scheme. In the topic-based, participants can subscribe themselves to individual topics which are

identified by keywords. Topics are similar to the notion of groups. An example of such systems is Scribe. Scribe [2] is an event notification infrastructure for topic-based publish-subscribe applications implemented on the top of Pastry [3], a DHT infrastructure. The topic, identified by a key called the *topicId*, represents a group of some subscribers that want to receive all generated events for the topic. The multicast tree, rooted at the rendez-vous point, is created using a scheme similar to reverse path forwarding and permits to disseminate the events published in the topic. Any node with the appropriate credentials can publish events that will be disseminated to all the topics subscribers along the multicast tree associated to the topic. The main drawback of topic-based is the lack of expressiveness. The content-based is more powerful than the topic-based, giving the users the ability to specify their subscription expressing their interest. In the content-based, the subscription schema is based on the actual content of the considered events and not according to some pre-classified and existing criterion (the topic-name). The participants can subscribe to logical combinations of elementary events and are only notified about composite pattern of events. SIENA [4] is a wide-area event notification service, employing an overlay network of event brokers, which support rich subscription languages. Also Griphon [5] is a content-based system, obtained with a similar mechanism. However, they commonly have the two drawbacks as follows. First, state of the art systems have static overlay networks consisted of reliable brokers under the administrative control, or assume that a spanning tree of entire brokers is known beforehand. Clearly, this is not feasible when a system involves an enormous number of brokers, which join and leave the overlay network dynamically. Second, a broker keeps a large amount of routing states and its control message overhead is huge. This is because every broker can be an intermediate router on the paths of an event dissemination tree. The main difficult in the design of content-based is to address the scalability property, that is the ability to provide the event service across a whole wide-area networks, while maintaining the expressiveness of flexibility in the subscription schema. The main issues in design content-based publish/subscribe systems are the following:

1) to maximize the expressiveness, that is the ability of the event notification service to provide powerful data model able to capture information about the events, able of expressing filter and pattern on the notifications of interest.
2) To guarantee the scalability that is the ability to provide the event service across a whole wide-area networks, while maintaining the expressiveness of flexibility in the subscription schema.
3) To balance the load among peers in the system.
4) To adapt the network topology and the groups of common interest subscriptions to the dynamics of the system.

In this paper, we propose an approach that explicitly addresses the system adaptability problem and that represents a trade-off solution between the first two issues.

3 Problem Solution

As previously mentioned, our paper aims to solve three main issues:

1) to realize a system with an high degree of expressiveness.
2) To guarantee the scalability with respect to the number of subscribers, publishers and with respect the size of the event space.
3) To adapt the network topology and the groups of common interest subscriptions to the dynamics of the system.

The first problem is addressed proposing a flexible application domain data model that can be applied to any specific application domain. Limiting the number of multicast groups is one of the main characteristic of the proposed data model, exploiting the intrinsic scalability provided by the application layer multicast (DHT-based) to solve the problem. The adaptability of our system is guaranteed by a mechanism that automatically evaluates the efficiency of the actual configuration and starts the multicast groups reconfiguration process.

Although, two hard problems are still unresolved in the proposed approach:

1) the optimization of the multicast tree reorganization process;
2) the realization of a fully distributed solution without introducing the specialized nodes.

3.1 DHT-based overlay network

In general, publish/subscribe systems are build on top static network infrastructures, and indeed many routing and forwarding algorithms rely on this static nature. Vice versa, many applications may benefit from more dynamically network infrastructures, allowing to scale with respect to nodes and data. As overlay network, we use the solution basis on the Distributed Hash Tables (DHT) infrastructure. Distributed hash tables have emerged as infrastructures for efficient, scalable resource lookup in large peer-to-peer distributed networks. Such systems are decentralize scalable, and self-organizing (i.e. often as well, they automatically adapt to the arrival, departure and failure of nodes in the network). Such characteristics make DHTs attractive for building distributed applications [8], [9].

In our solution, we have used a DHT infrastructure based on Pastry [3], a scalable, distributed object location and routing substrate for wide-area peer-to-peer applications. Pastry is a self-organizing overlay network of nodes, where each node routes client requests and interacts with local instances of one or more applications. The multicast communication can be implemented in two flavours,

network supported and application level multicast. First one is based on IP multicast model, that allows scalable and efficient multi-party communication, particularly for groups of large size. However, the deployment of IP multicast group requires substantial infrastructure modifications and is hampered by a host of open problems such as reliability, flow, and congestion control, security and access control. Because of the problems facing the deployment of a network level multicast service, we have adopted an application-level multicast solution on peer-to-peer overlays based on tree building. In the tree-based approach, a tree for each group is built. Multicast messages related to a group are propagated through its associated forwarding tree. This form of application-level multicast is leveraging the object location and routing properties of the overlay network to create groups and join groups. The application then creates and manages the tree and uses it to propagate messages. As application-level multicast infrastructure, we use Scribe [2], an event notification infrastructure for topic-based publish-subscribe applications implemented on the top of Pastry. Briefly, any node may create a topic (or group). Each topic is identified by a *key* called the *topicId*, obtained by the hashing of the group's textual name, for example. The topic represents a group of some subscribers that want to receive all generated events for the topic. The node responsible for the namespace segment containing the *topicId* is the root of the tree or the rendezvous point. The multicast tree, rooted at the rendez-vous point, is created using a scheme similar to reverse path forwarding and permits to disseminate the events published in the topic. To subscribe to a topic, a node routes a message through the overlay to the rendez-vous point. Any node with the appropriate credentials can publish events that will be disseminated to all the topic's subscribers along the multicast tree associated to the topic. Scribe scales well because of its decentralized algorithm and the randomization of overlay addresses ensures that the tree is well balanced. Scribe also tolerates node and root failures ensuring fault-tolerance property to the notification mechanism.

3.2 Problem notation

Here we introduce specific notation we will use in the rest of the paper.

- Let N denote the number of attributes of the application domain schema.
- Let Ω define the event space (an N-dimensional cartesian space).
- Each event being published within the system can be uniquely described with a single value ω.
- Let V be the set of nodes of the network.
- Let V_S be the set of nodes hosting subscribers.
- Let us assume that each subscriber v_i has a set of r_i subscription preference expressed by $I_i = \{b_{ij}, j=1 \ldots r_i\}$. Each b_{ij} is an aligned rectangle in space Ω.
- Let us define I to be the set including all subscription rectangles.

3.3 Application domain data model

For each application domain, the publishers and subscribers use a data model representation based on a set of application dependent attributes. We indicate this set of attributes as the *application schema*. Attributes are characterized by their *type*, *name*, and *constraints* on possible values, specifying the general format of data and their possible values (within each application domain). Each application domain has its own schema, thus multiple domain schemas can be handled simultaneously by the same application (and also different applications may run on the same network). For example, a complex system management tool may handle simultaneously three domain: a distributed system management domain, a power grid monitoring domain and resource discovery domain. The proposed application domain data model allows subscribers to specify the subscription preferences, indicating the attributes and the related range values; and it allows publishers to disseminate events to the interested users. The proposed data model, inspired by [6], permits to represent the application domain through an N-dimensional cartesian space Ω in which each event can be uniquely described with a single multidimensional element ω. The axes of the event space represent the attributes and they are labelled with the *name* field contained in the schema, while the ranges of the axes are specified into the *values* field, as shown in Figure 3. Using the proposed abstraction, each subscription preference of the form [*name attribute 1, range value 1], [name attribute 2, range value 2], [name N attribute, range value N]* is represented as an hyper-rectangle into the event space. In Figure 1, an example of schema for the resource discovery domain, for simplicity composed only by three attributes, is reported. The graphical representation of the associated event space Ω is shown in Figure 3.

Name	Type	Values
Available_Resource_Capacity	percentage (%)	0..100
Cost	dollars ($)	10..1000
Reputation	a-dimensional (-)	1..10

Figure 1 The schema for resource discovery application domain

Figure 2 shows an example of preference subscription b, represented also as horizontal plane in Figure 3.

Name	Type	Values
Available Resource Capacity	$	50..85
Cost	$	550..650
Reputation	-	5

Figure 2 An example of subscription preference description

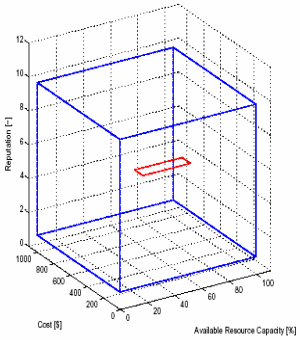

Figure 3 Graphical representation of the event schema

4 System architecture

Our solution is based on a two layers architecture. As shown in Figure 4, on the top of the transport layer we build a DHT-based multicast application layer based on Scribe and Pastry. The *DHT multicast layer* embeds the application layer multicast protocol and the DHT-based overlay network, providing to the upper layer, the following primitives:

1) overlay network management primitives (e.g. *join(), subscribe(), unsubscribe(), lookup(), route()*);
2) multicast primitives (e.g. *join(), create(), leave(), multicast()*).

The content-based pub/sub functionalities are implemented by the following three processes:

a. Multicast Group Identification (MGI) process that associates a subscription b_{ij}, to one or more multicast trees.
b. Multicast Group Matching (MGM) process that identifies the multicast groups to which deliver an event ω.
c. Multicast Group Creation (MGC) process that re-organizes the multicast groups when the systems state change compromising the efficiency.

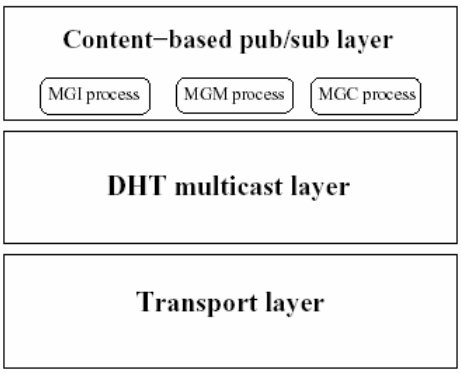

Figure 4 Layers

We propose a solution to create the multicast groups, based on the dynamical clustering of the subscribers' preferences, that builds groups composed of nodes with similar interests.

4.1 System state description

For each application domain, the publishers and subscribers use a data model representation based on a set. The identification, creation and reorganization of multicast groups and the solution of the matching problems, require some knowledge about the system state. The system state can be descried by the following set:

1) the set of overall subscription *I*. This set changes during the time and an overall view of the subscriptions is needed to dynamically compute at run time the multicast groups that better represent the system.
2) The set of overall multicast group *M*. This information is needed to solve the multicast group matching problem, and the multicast groups identification problem.
3) The set of overall events generated in a chosen temporal windows *E*. This information could be useful to optimize the multicast group creation process and the multicast groups identification problem. Note that the set of overall events *E* should be maintained updated when the system is running.

4.2 Clustering based solution

We group the different pattern of subscriptions into clusters, using the Minimum Spanning Tree [7] algorithm. We used the Euclidean distance between centroid of hyper-rectangles expressing the subscriptions, as distance function to be minimized The set *I* of all subscriptions is partitioned into clusters of different volume. Each cluster c_i is a rectangle in the Ω space of centroid $cen(i)=(x_1, x_2, \ldots x_N)$. Note that the coordinates of the centroid are not fixed but they dynamically change depending on the the pattern of subscriptions. The clustering based solution associates a multicast group *MG(i)* to each cluster c_i and then the nodes in the multicast group *MG(i)* are organized in the multicast tree *MTree(i)* where the centroid *cen(i)* is the root.

MTree(i) is responsible for the events and the subscriptions within the range of the cluster c_i.

The Multicast Group Identification problem is solved by evaluating the intersections among the hyper-rectangle b_{ij}, expressing the preference j of subscriber v_i, and the clusters in the events space. The intercepted clusters determine the multicast groups to which the subscriber v_i will be associated. Every time an event ω is published, the Multicast Group Matching process looks for the cluster containing the event ω and delivers the event ω through the associated multicast tree. The clusters are organized into a R-Tree data structure.

The Multicast Group Creation process is responsible to evaluate the inefficiency due to the actual configuration of the multicast groups and the dynamics of the subscribers and publishers. If the efficiency decreases the multicast group will be reorganized.

4.2.1 Multicast Group Identification process

When a new node joins the overlay network, it receives from the bootstrap node the following information:

- the access to the network, by the *nodeId* and the routing tables;
- the application schema, that contains the list of all the attributes and their domains and permits the node to express its own preferences and to publish the events;
- the R-Tree data structure, representing the set of all clusters.

When a node v_i subscribes for a preference b_{ij}, the *MGI* process evaluates the interception among the subscription b_{ij} and the clusters. Then it assigns the node to one or more multicast groups. More formally, the process can be defined by the following steps:

- Determine the set $MG(b_{ij})$ of the clusters intercepting the subscription b_{ij}.
- Determine the set of centroids $cen(b_{ij})$ of the clusters in $MG(b_{ij})$.
- Determine the set of the corresponding multicast tree MT(b_{ij}).
- For each multicast tree *MTree(k)* contained in $MT(b_{ij})$, subscribe v_i to the multicast tree *MTree(k)* at the DHT-multicast layer.

If a new subscription does not match any active multicast tree (meaning that the multicast tree does not exist), a new multicast tree is then created and the R-tree is modified by a leaf node insertion.

4.2.2 Multicast Group Matching process

Let us suppose that the publisher node has already joined the network. The node has already received from the bootstrap node the *application schema* and the *R-Tree data structure* containing the set of actual clusters. The node can resolve locally the MGM process, having all the information to compute the interception between the generated event ω and the clusters c_k. More formally, we can formulate the *MGM* process by the following steps:

- Determine the cluster c_k containing the event ω.
- Determine the centroid *cen(k)* of the cluster c_k.
- If the multicast tree *MTree(k)* exists, deliver ω *MTree(k)*, else drop the event ω.

As previously mentioned, during the step 1 the process determines the cluster c_k searching the *R-Tree structure* organizing the actual clusters. The *MGM* process need as input only local information: the event ω and the set of clusters. Thus it can be locally processed by each node in the overlay network.

4.2.3 Multicast Group Creation process

This process is responsible to evaluate the inefficiency of the actual configuration with respect to the pattern of subscriptions and the number of multicast groups. On the basis of the evaluated inefficiency, the process re-organizes the multicast groups and the associated multicast trees. As final result of the re-organization, the subscribers will be associated to more efficient multicast groups with respect to the existing groups.

Periodically the MGC process executes three macro-steps:

1) evaluate the inefficiency of the actual clusters configuration.
2) Estimate the impact, on the efficiency and on the cost constraints, of a new clusters configuration.
3) Re-organize (if needed) the multicast trees, on the basis of the new multicast groups. This means to map all existing subscriptions to the new multicast groups.

The definition of system inefficiency is reported in Section 4.2.4. The pseudo-code for the Multicast Group Matching process taking in account the system inefficiency as described in Section 4.2.4, is here reported in the following.

(1) Choose K
(2) Choose the cost constraint c;
(3) Compute the new set of clusters $\{c_1, c_2, ..., c_k\}$
(4) If $\gamma_I \Delta FN(I) + K \gamma_R < c$
Then
(5)Distribute the new set of clusters; to all the publishers and subscribers;
(6)Re-organize the multicast groups (and multicast trees);
Else
(7) set $K=K'$
(8) go to step (3)

4.2.4 System inefficiency

The system inefficiency clustering is caused by subscriptions not correctly assigned to the existing clusters. A subscription not assigned to the right cluster is a source of false negatives, while a subscription that intercepts more than one cluster would be a source of duplicated messages (false positives). We use the number of false negatives as performance metric considering not acceptable that some

event could be lost and that some subscribers will not receive events they are looking for.

The false negatives number for the subscription b_{ij} with respect to the current set of clusters $\{c_1, c_2,..., c_k,\}$ is defined as $FN(b_{ij})$, while the overall system inefficiency is $FN(I)$.

After varying the set of clusters $\{c_1, c_2,..., c_k,\}$, the variation of the inefficiency can be measured as the difference among the new system inefficiency and the old one. We call the variation of inefficiency $\Delta FN(I)$.

If $\Delta FN(I) < 0$ then the new set of clusters decreases the number of false negatives. Otherwise, if $\Delta FN(I) > 0$ the new set of clusters increases the inefficiency and if $\Delta FN(I) = 0$, the effects are neutral for the inefficiency.

If we define γ_I the cost associated to a unit of inefficiency, then the cost of the overall inefficiency is $\gamma_I FN(I)$ and the gain (or lost) associated to the new set of clusters is $\gamma_I \Delta FN(I)$. The multicast group reorganization cost can be assumed proportional to the number n of multicast groups involved in the reconstruction phase. The number of clusters is fixed (K), thus the cost of re-configuration of the multicast group is constant (except if we decide explicitly to increase or decrease K). If we define γ_R the cost to reorganize a multicast group, the overall cost to reorganize K groups is $K\ \gamma_R$. The set of clusters is distributed over all the nodes only if the cost to reorganize the clusters satisfies the following cost constraint, that is $\gamma_I \Delta FN(I) + K\ \gamma_R < c$. where the constant c can be appropriately selected.

If $c<0$ means that we accept the reconfiguration (Step 3) only if gain in efficiency is grater or equal to the cost of the reconfiguration. If $c > 0$ means that we accept to reconfigure the clusters, and then the multicast group, even if we don't have a direct revenue.

5 Content-based pub/sub primitives

In this section we give an high level description of the primitives that should support the proposed content based publish/subscribe system.

These primitives allow to join a network (*cb_join()*), to subscribe for a set of events (*cb_subscribe()*), to publish events (*cb_publish()*), to register a subscriber to a multicast group *(cb_register())*, to cancel a subscriber from a multicast group *(cb_cancel())* and to notify system state management actions such as MTree reorganization (*cb_notify()*).

They invoke the DHT multicast layer primitives: *join()*, *lookup()*, *route()*, *send()*. We briefly describe the Content-based pub/sub primitives.

cb_join() allows a new publisher or subscriber to join to the content based network. When the *cb_join()* primitive is invoked, the bootstrap node is contacted and the subscriber node is registered to the overlay network. The bootstrap node delivers the application schema and the system state dependent information (for example, the set of overall subscription I or the set of overall multicast group M). At the overlay network layer, the *cb_join()* is mapped into an overlay *join()*.

cb_subscribe() allows the nodes to subscribe for a preference b_{ij}. This primitive sends the preference b_{ij} to the system and it activates the *MGI* process to register the subscription to the appropriate multicast group(s). The *cb_subscribe()* invokes one or more subscribe(s) at the overlay network layer.

cb_publish() allows a node to publish an event. The *cb_publish()* primitive activates the MGM process, looking for the cell or the cluster that contains the event. After that the multicast group is determined, the primitive delivers the event to the associated multicast tree using the overlay network layer primitives *lookup()*, *route()* and *send()* (depending on the scenario).

cb_register() registers a subscriber to a multicast group. The primitive is invoked after a *cb_subscribe()* and in the multicast group reorganization phase.

cb_cancel() delete a subscriber from the multicast group. The primitive is implicitly invoked during the clusters reorganization phase or explicitly invoked when a subscriber fails or leaves the network.

cb_notify() notifies system state management actions or events.

6 Conclusion

The proposed approach to adaptive content-based subscription management is actually a work in progress.

We propose the architecture of the system and we design a solution to the main emerging problems.

The novelty of our approach, respect the existing solutions, is in the adaptability of the systems to the load, the capability to specify a desired level of efficiency and cost constraints and the scalability, obtained using application level multicast.

As mentioned before, the load balancing among the peers and the realization of a fully distributed solution that does not need specialized servers, still remain open problems. Indeed ongoing works are aiming to realize a fully distributed solution for the computation of the system inefficiency and for the inherent decision making process in the Multicast Group Creation process.

References:

[1] P.Th. Eugster, P.A. Felber, R. Guerraoui, A.-M. Kermarrec, The Many Faces of Publish/Subscribe.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communication (JSAC*), 20(8), 2002.

[3] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. of the 18th IFIP/ACM*

International Conference on Distributed Systems Platforms (Middleware 2001), 2001.

[4] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst*., 19(3):332{ 383, 2001.

[5] R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward, Gryphon: An Information Flow Based Approach to Message Brokering.

[6] Riabov, A. Z. Liu, Wolf, J.L. and Yu, P.S., Li Zhang, New algorithms for content-based publication-subscription systems, *Proceedings. 23rd International Conference on Distributed Computing Systems*, 19-22 May, 2003

[7] R. Jain, *The Art of Computer System Performance: Analysis, Techniques for Experimental Design, Measurement, Simulation and Modeling*, John Wiley and Sons, New York, 1991.

[8] J. Kubiatowicz et alii, OceanStore: an architecture for global-scale persistent storage*, Proceedings of the ninth international conference on Architectural support for programming languages and operating systems, 2000.*

[9] M. Castro, P. Druschel, Anne-Marie Kermarrec, A.Nandi, A. Rowstron, A. Singh, SplitStream: high-bandwidth multicast in cooperative environments, *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003