Architecture and Implementation of a Content-based Data Dissemination System

Austin Park
Brown University
austinp@cs.brown.edu

ABSTRACT

SemCast is a content-based dissemination model for large-scale data delivery applications. The model aims to segregate stream content over multiple channels, each of which acts as an independent dissemination tree. The channel contents, rates and destinations are monitored to determine if performance improvements are possible on the system. For my masters project, I implement the basic system model of SemCast.

1. Introduction

Content-based dissemination is a routing model that forwards data based on their content instead of having attached routing information. This model is inherently necessary for large-scale distributed applications, because traditional IP-based routing model would result in a hefty overhead. Routing the packets based on their content allows the information to be sent and passed down the network based on the application schema and independent of the underlying network structure. Examples of high-volume streaming data applications that would benefit from content-based dissemination are environment monitoring, networked games, and medical and military data distribution on wireless devices.

In this model, sources generate data streams with no routing or destination specified. In other words, the destination and source are independent and these streams are instead identified and routed by their profiles, which are declarative specifications provided by the destination nodes. The goal of the system is to reduce overhead and efficiently route data from the source to the relevant destination nodes, also known as subscribers.

In this paper, I provide a short overview of SemCast [1], an overlay-network system that performs distributed content-based routing of highly-distributed and high-volume data streams. I also describe my work implementing SemCast.

SemCast creates a set of dissemination channels, each being an independent tree of brokers, and each one containing different content. The content of each channel is specified by the collection of profiles of all the clients attached to the specific channel. Sources forward the incoming message to all the channels with matching content. Subscribers listen to all the channels that disseminate the content they are interested in.

SemCast requires content-based filtering only at the source and destination brokers, instead of at each level of the distribution tree. As each incoming message is mapped to specific channel(s), “routing” at each broker in the tree requires only reading the channel identifier and forwarding the message to the relevant children. This minimizes overhead by eliminating the need to do any complicated filtering at each broker, as is common in more traditional publish subscribe systems. Creating independent distribution trees also allows the model to periodically and easily reconfigure the channel layout to optimize system performance.

In this paper, I provide a background description of SemCast and the different components that make up its cost-based channelization model. This is followed by the section describing the core of my work: setting up the dissemination channels and propagating information with a specific application. I will explain the different stages of creating channels, and forwarding information to the subscribers. Lastly, I will identify future work that will be needed in order to provide a full and complete implementation of SemCast.

2. Background

In this section, I will describe the basic design and architecture of SemCast and its different components and the system data flow [1].

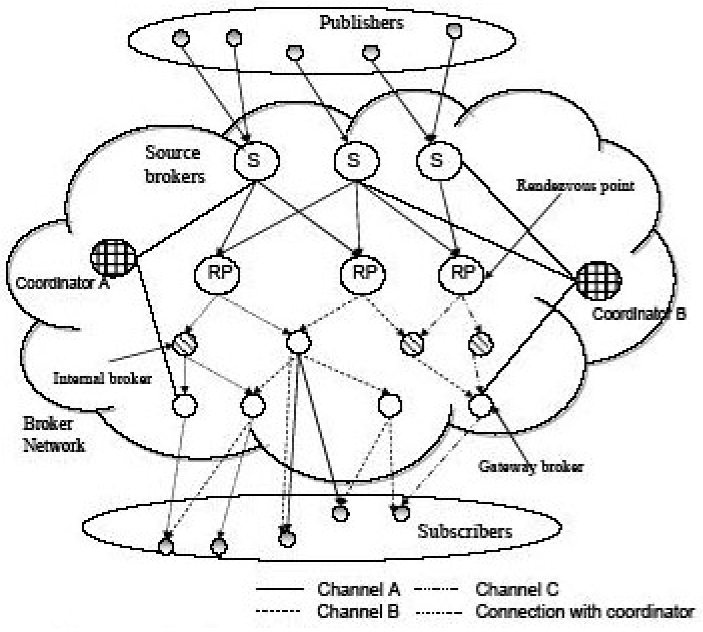


Fig. 1 Basic SemCast system model

2.1 System model

SemCast is based on a content-based publish-subscribe model where the subscribers express their subscriptions using predicate-based expressions called profiles. The system consists of a set of broker nodes (i.e. servers) organized in an application-level overlay network as in Fig. 1. Publishers produce the content and the source brokers (otherwise referred as sources) poll the publishers to introduce the content to the system. Rendezvous points are in charge of one or more channels and serve as the root node of those channels' distribution trees. Subscribers connect to the system through the gateway brokers, which are the final destinations for the data streams. Each subscriber can be connected to more than one gateway broker, if it subscribes to multiple channels. Brokers that are connected to neither a publisher nor a subscriber are called interior brokers. The system also contains a coordinator node, which performs all the channelization and manages the dissemination channels. It keeps track of the existing channels and their content expressions, and the rendezvous nodes responsible for each channel. It communicates with the sources and gateway brokers to inform them about the existing channels or the need for new ones.

The content of each channel is defined by a channel expression, which is a union of the client profiles served by that channel. A profile is assigned to a channel if the channel expression overlaps with the profile. For a given channel, the corresponding dissemination tree includes all gateway brokers whose subscribers' profile is included in that channel expression.

Basic data-flow:

In SemCast, the coordinator makes decisions on the content expression and existence of channels and forwards this information to the bootstrap. The bootstrap is a designated node that polls the source servers for the data that will be distributed through the channels. The rendezvous points connect to the bootstrap, and when a message enters the system, the bootstrap will match the relevant content expression to the assigned rendezvous points and forward that data to them. The rendezvous points then forward the contents to the relevant gateway brokers through the corresponding channel tree. Brokers must update and reference a lookup table mapping the channel identifiers to the children brokers because there may be multiple channels going through that broker. Upon receiving a message, the broker must look up the channel identifier and forward it to the returned list of children. Each gateway broker maintains a local filtering table mapping its subscribers' profiles to their IP addresses and also a mapping of channels to its subscribers. Received messages from a channel are matched against the profiles of the subscribers and forwarded to the relevant ones. This insures that the subscribers will not receive irrelevant data.

Channelization algorithm:

The network topology in many applications will change over time in regards to network and data conditions. The channels created initially may no longer be the most efficient in terms of bandwidth consumption, resource usage, and load. Therefore, SemCast periodically reevaluates its channelization decisions and may reorganize the channels. This reorganization, also known as channelization, addresses the following issues: how many channels to create, the contents of each channel, and which channels each broker must listen to. SemCast uses a cost-model based on profile semantics and stream statistics, and network characteristics to make its decision.

SemCast uses the knowledge of "overlap" among profiles to determine channel content. Two profiles overlap if the intersection of the set of messages from both profiles is non empty. Assigning overlapping profiles to the same channel allows matching messages to be forwarded only once through the same dissemination tree, eliminating redundant transmissions. SemCast discovers containment hierarchies among profiles, where each profile has as its ancestor a profile that covers it, and as its descendant profiles that are covered by it.

SemCast performs channelization in two phases. First, SemCast constructs the profile containment hierarchy using syntactical similarities among profiles and stream rate statistics and profile overlap. Each level of the hierarchy is a good candidate for serving as the dissemination channel because of the overlap of constituent profiles. However, creating channels solely based on this containment information may miss optimization opportunities with partially overlapping profiles.

The second phase identifies partially overlapping profiles and determines if placing them in the same channel would further reduce the run-time costs. Merging partially overlapping hierarchies does not guarantee a reduction in redundant transmissions. This is because while common messages to both profiles will be transmitted only once, uncommon messages will also be forwarded to the destinations not interested in them, increasing unnecessary bandwidth consumption.

2.2 Previous approaches

There have been various approaches to implementing content-based dissemination models (e.g., [2, 3, 4, 5]), most of them involving unicasting data to each subscriber, or to do content filtering at each level of a single tree. Both methods are inefficient since the former sends redundant data through the network and consumes more bandwidth, and the latter consumes valuable server resources. SemCast is an improvement over these models because its subscribers will join and listen on the relevant channels, so it is sufficient to send the messages once through the network. Also, content filtering occurs only at two points in the SemCast model, the bootstrap and gateway brokers, as described in section 3 below.

3. Implementation

For my SemCast implementation, I used XPORT, a high-volume rss-server feed propagation application. In XPORT, broker nodes form a single data distribution tree. Subscribers connect to the brokers to access the rss feed content. The bootstrap node polls the different rss-feeds and propagates the content its children. Nodes connect to the tree through the physically closest neighbor node. Fig. 2 shows the basic XPORT system model. The section inside the yellow cloud is where data routing occurs, and where SemCast is implemented. Instead of a single distribution tree, multiple channels are constructed.

There are a few fundamental differences between XPORT and SemCast. XPORT does not support the notion of a coordinator, rendezvous point, source and gateway brokers. Instead of multiple channels acting as smaller dissemination trees as in SemCast, XPORT uses one large single dissemination channel. Additionally, XPORT performs in-network filtering as opposed to SemCast which does filtering only at the source and gateway brokers.

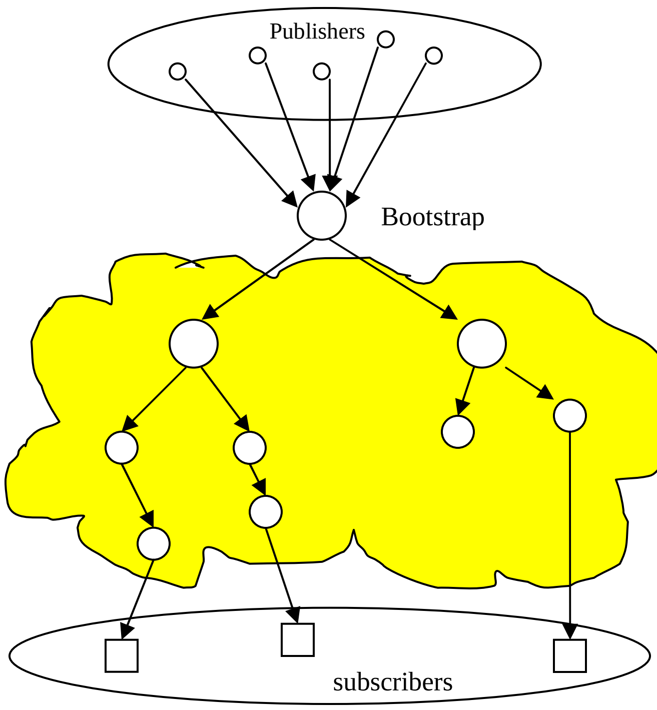
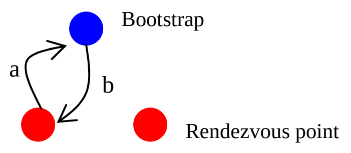


Fig. 2 Basic XPORT system model

My goal is to use SemCast as the data-distribution model to allow the application to be more efficient in bandwidth and resource usage. For the rest of this section I will describe how SemCast is constructed and its data flow. The system setup occurs in two main steps: the rendezvous points establish a persistent connection with the bootstrap. Then the gateway brokers join the routing tree by establishing a connection to the channels. These steps are described below.

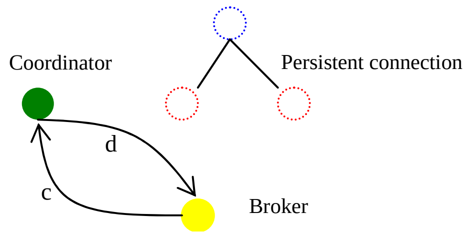
Bootstrap ←→ Rendezvous point

When the system first starts up, each rendezvous point will attempt to establish a persistent TCP connection with the bootstrap node (Fig. 3a). Each rendezvous point initiates a connection request to the bootstrap with its identification key which includes its IP address and port numbers. The bootstrap (in current implementation, I assume the bootstrap is also the coordinator) stores this key and sends back a confirmation of the request. The bootstrap stores this rendezvous point's key so that in the future, it knows which rendezvous points are in existence. The rendezvous points assume from this confirmation message that the bootstrap is ready to accept them and establishes the persistent TCP connection.



a) Rendezvous point (RP) sends bootstrap its key.
b) RP sends confirmation message and persistent TCP connection is formed.

Fig. 3a



c) Broker requests a list of RPs by sending its profile
d) Coordinator sends back a table of channel → responsible RP's key (IP address, port).

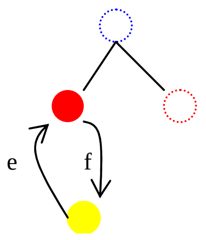
Fig. 3b

Coordinator ←→ Broker

Next, the brokers must connect to routing tree through the relevant channel(s). This occurs in two steps: the broker will contact the coordinator with its profile with a temporary TCP connection (Fig. 3b). The coordinator matches the profile to all the existing channel contents. It sends back to the broker the channels it should connect to and the rendezvous point(s) in charge of that channel(s).

Rendezvous point ←→ Broker

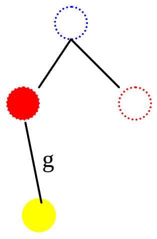
In the next step, the broker will attempt to establish a persistent connection to each of the channels (Fig. 3c). The connection will be persistent because the broker wants to receive the data flowing through that channel. The broker makes a temporary connection to the rendezvous points for each channel on the table and sends a message to each one of them asking for a list of potential parents. The rendezvous point will do a lookup for the channel to get a list of the brokers on that channel as potential parents. This list is essentially all the brokers in that channel, since any of them can be potential parents. If a channel exists, then there must be an existing tree, so the rendezvous point will send back to the requester a list of the nodes of that channel. Otherwise, the coordinator must have decided this rendezvous point will be in charge of a new channel. So it sends its own key, as the RPs is the only broker currently in this channel. The rendezvous point updates its data structures to keep track of the channels it is responsible for.



e) Broker sends request for list of nodes in the channel to each RP of each channel from step d).

f) If channel id is new (does not exist in RP's lookup table), RP sends back its own key. Otherwise, send back a list of potential parents for the channel.

Fig. 3c



g) Broker establishes a persistent TCP connection to the physically closest node from the list that accepts its request.

Fig. 3d

Channel ←→ Broker

The broker attempts to connect to the closest node from the returned list, by pinging each of them (Fig. 3d). It is possible for a parent to reject a request, so if a parent is successfully selected (potential parent responds positively), the broker node checks its local data structures to see if an existing TCP connection to this parent already exists (that is, a previous connection was already established). Otherwise, a new persistent TCP connection is made.

Message Name	**Sender**	**Receiver**	**Explanation**
COMMAND_ASSERT_RP	Rendezvous points (RP)	Bootstrap	Fig. 3a: a)
COMMAND_RESPONSE_ASSERT_RP	Bootstrap	RP	Fig. 3a: b)
COMMAND_REQUEST_BS_RP	Broker	Coordinator	Fig. 3b: c)
COMMAND_RESPONSE_BS_RP	Coordinator	Broker	Fig. 3b: d)
COMMAND_REQUEST_PARENTS	Broker	RP	Fig 3c: e)
COMMAND_RESPONSE_PARENT_LIST	RP	Broker	Fig 3c: f)
COMMAND_PARENT_SELECTED	Broker	Selected parent node (another broker or RP)	Fig 3d: g)

Fig. 4 System messages used to create channels

When the application is initiated, the bootstrap node polls the relevant rss-servers based on the current system's profiles . As the content stream is received by the bootstrap, it checks that the content matches with the profiles in the tree below. If there is an established channel, the message is sent to the rendezvous point that is managing that channel. The destination node will receive the message from the parent, process it, and the message is added to the queue to be sent out on the channel to next respective child.

4. Future Work

While the current system allows rss-feed data to be distributed over the channels, to fully implement SemCast, we must implement channelization [1]. Channelization is performed by the coordinator and is determined using a cost-model based on the profile sematics, stream statistics, and network characteristics gathered from the current system setup. Also, when the channelization request is made by the coordinator, the coordinator must allow for seamless data delivery even when brand new channels and network topology are constructed. The system layout must be saved by the coordinator prior to channelization, and once each broker is connected to the re-configured network, the old TCP connections must be cut to reflect the new connections.

In the current implementation, I assumed the coordinator to act also as the bootstrap for simplicity. In future applications, the two roles may exist on separate physical nodes. I also constructed one coordinator for the entire system, but multiple coordinators may exist if the system becomes complex. Multiple coordinators should appear as one to the rest of the system, so the coordinators will have to update each other as system updates are received and sent.

5. Conclusion

I implemented a novel semantic multicast model which allows for efficient data-stream dissemination. The main idea is to segregate the data-streams into channels, determined by the content, rates, and destinations, which are regarded as independent dissemination trees. This model has a number of advantages over other models because it does not require filtering at the interior brokers of the tree, and allows for fine-tuned control of the channels. My goal was to implement this with a practical application, allowing for further testing and data aggregation to supplement the current results of SemCast.

6. References

[1] O. Papaemmanouil, U. Cetintemel. SemCast: Semantic Multicast for Content-based Data Dissemination. In ICDE, 2005.
[2] G. Banavar, et al. An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems. In ICDCS, 1999.
[3] A. Carzaniga, et al. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3): 332-383, 2001.
[4] C.-Y. Chan and P. Feber. A Scalable Protocol for Content-Based Routing in Overlay Networks. In NCA, 2003.
[5] Y. Diao, et al. Towards an Internet-Scale XML Dissemination Service. In *VLDB*, 2004.