Chapter 1

Publish-Subscribe

1 Introduction

What is publish/subscribe? We all know that publish/subscribe is a mean to deliver information from the sources to the users who subscribe to them. However, in the computer world, the concept of publish/subscribe is more complicated than that.

To get information from servers who provide data sources, a user basically goes to the servers to look for them; however, most of the time it is not a practical way for the user to collect information because it might take a lot of internet resources such as the bandwidth, and the user's time. Thus, publish/subscribe models are devised to help out delivering information to users. According to publish/subscribe protocols, a user specifies a set of interests and submits it to the server (which is the fundamental concept of user profiles), then the server gathers the data which match the user's interests and transfer to the client.

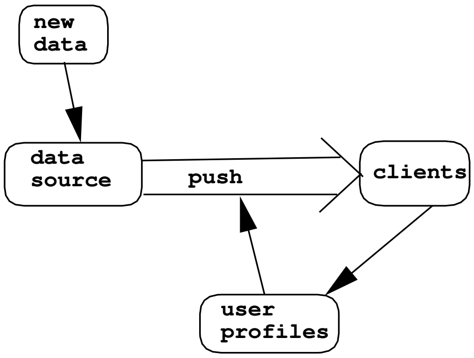

Figure 1.1: Abstract picture of publish/subscribe

Although the concept of publish/subscribe mentioned above is quite simple, the problems behind it are not. Nowadays information technologies advanced rapidly, data can be spread farther away easily; however, if millions of users want to obtain the same information from the same source at the same time, the problem of scale surfaces. The information transmit through the network might be delayed or disrupted due to bandwidth problem, most of the users might not be able to reach the source at all.

1.1 The problems

1.1.1 The scalability

Scalability means that the system must be available over a wide-area network populated by numerous users and many data sources.

1.1.2 The asymmetry

Generally, asymmetry means that the server side and the user side have different property of some specific stuff. There are many kinds of asymmetries. For example, in many networks, the bandwidth from the server to the user is much larger than the bandwidth in the opposite direction; a system may have a few servers and a much bigger number of users. And if only newly generated data or updates of existing data should be delivered to the users, it is also a kind of asymmetry.

1.1.3 Why are scalability and asymmetry a cause for publish/subscribe?

Since the extremely popularity of Internet, Intranet, distributing the data generated in real-time to the users is becoming more and more important. Such transfer of data involves the timely distribution of data to a large set of users. And it has some special characteristics: tremendous scale, high degree overlap in user data needs, and asymmetric data flow from sources to consumers. These characteristics make the traditional client-server data management methods ineffective. The request/response mode will cause catastrophic congestion in the web traffic. This has been demonstrated by frequent delays and service disruptions when accessing networked data sources.

1.2 publish/subscribe models

Publish-subscribe systems provide a convenient method to interconnect applications on a distributed network. In the publish- subscribe system, information providers publish units of information called events, and information consumers subscribe to particular categories of events.There are several different kinds of the publish/subscribe interaction model. Each one has its own advantages but also shortcomings.

1.2.1 Topic-based publish/subscribe

The classical topic-based or subject-based style gives a static classification of the message by introducing group-like notions. And it is implemented in most industrial application softwares. In this kind of publish/subscribe system, it is easy to get the scalability and good performance by using group-based multi-cast techniques, more exactly, by assigning each subject to a multi-cast group.

1.2.2 Content-based publish/subscribe

However, research efforts have been targeted more towards content-based publish/subscribe systems. This style is more flexible. It supports an event schema defining the type of information contained in each event. It removes completely the "arbitrary" division of the messages, and lets the consumers delineate their individual interests by expressing properties of messages they wish to receive.

While the topic-based systems only offer a limited selection capacity, typically based on a predefined set of "channels" . The content-based systems can offer a flexible and fine-grained selection mechanism to describe precisely those events or combinations of events in which they are interested.(i.e. to get a good filter to screen out those irrelevant information).

1.3 Context

For any data delivery system, it may have different choice among the delivery options:

1.3.1 Push or Pull

In data dissemination system, "pull-based" means that the server will search and transfer the related information to the client only after receiving the request from the client. So the clients have to pro-actively known the data. It is the client that initials the data delivery. The request-response is pull-based. While "push-based" means that the server begins the data transmission. It will send the information to a set of clients without any specific request from those clients.

1.3.2 Periodic or aperiodic

Both push and pull can be done in either periodic or aperiodic way. Periodic delivery is done according to some pro-arranged schedule, which can be fixed or flexible. In contrast, aperiodic delivery is event-driven. For pull, a data delivery is triggered by a client request. For push, a data transmission is executed only when there is data update or new data.

1.3.3 Unicast or 1-to-N

In unicast mechanism, the data items are sent to one client from one server. While 1-to-N mechanism allows the data items are sent to multiple users. Publish/Subscribe are typically performed in 1-to-N way. But up to now, in the internet cases, the most publish/subscribe systems are done by unicast because there is no support for it, we know HTTP is based on request/response ,a unicast(i.e. point to point) method of data delivery.

Publish/subscribe is push-based, The server initials the data transmission, and is aperiodic. There is no predefined schedule for delivering data.

messages) subscribes to that information, signaling that it wishes to receive all pieces of information satisfied the specified characteristics.

In Publish/Subscribe communication, publishers "publish" information onto a network, labeling the information using a hierarchically structured name. Subscribers "subscribe" to this information by using the same name. In most applications, however,each party can be publisher and subscriber, which allows a very flexible interaction.

1.4 Compare and Contrast traditional query-based systems with Publish/Subscribe

In the traditional query-based systems, each user should send requests for data to the server. The large number users for a popular event can generate huge overhead in the servers, resulting in long delay and server crashes. The situation is that users must continually poll the server to obtain the most current data, resulting in multiple requests for the same data items from each user. If the interests of a large part of the population are known a priori, most of these requests are unnecessary.

In a publish/subscribe system, the user can submit a profile to the server once, and then continuously receive data that are (supposedly) relevant to him or her without need for submitting the same query over and over again. Whenever a new item is added to the database, it will be filtered by those profiles. If it matches a profile, the owner of that profile will receive the new data.

Additionally, given the fact that user interests are changing over time, the profiles must be updated accordingly to reflect the users' information needs.

1.5 History of Publish/subscribe

1.5.1 Information Bus

The information bus is the first idea to perform distributed information in large scale. It is one kind of publish/subscribe protocol. It gives several principles for the design of the scalable distributed system.

1.5.2 SIFT

The SIFT system was developed at Stanford University as a method to distribute data to a client community. SIFT is a push-based system, the document source delivers data to the users by matching the data with the profiles they provided. The SIFT provides a novel technology in indexing the client profiles.

1.5.3 Pointcast

Pointcast is a popular data dissemination service. It obtains the profiles from the users that describe their interests, and then use these profiles to filter those data before they are sent to every client.

1.6 Chapter outline

In the following, we discusse the technologies and architecture involved to build publish/subscribe systems in details, then give some examples of applications and products of publish/subscribe systems used nowadays. (still need to be modified)

2 Technology Issues

This section discusses the physical and algorithmic issues involved in Publish-Subscribe systems. As we have seen from the introduction there are many technologies that go into making a Publish-Subscribe system. These involve databases, networks and users. Publish-Subscribe systems must efficiently organize user profiles, query and handle data, move the data to the users, and make sure that the users are happy.

2.1 Broadcast, Multicast, Unicast

Here we discuss network delivery options available for Publish-Subscribe systems. As discussed in the Introduction, two types of options exist: unicast and 1-to-N; and there are three possible technologies that can be used:

- *Broadcast Broadcast* technology has a long history of use. It is a 1-to-N delivery mechanism that we have seen used in radio and television for many years. In most wired Publish-Subscribe envionments the Ethernet broadcast protocol is used. The advantage of using broadcast is the ability to deliver the same data to a large number of users without a performance penalty. A single transmission satisfies all users who are listening to the broadcast. The disadvantage to using broadcast is that the set of users who can receive the data is impossible to identify and unbounded in size. The environment in which broadcast is feasible for Publish-Subsribe is a local area network (LAN) where the user base of the Publish-Subscribe system is dense. Since some Publish-Subscribe systems can publish a generous amount of data, broadcasting to all machines on a network can generate a log of network traffic. And, if there are only a few subscribers to the system, broadcast can have an unnecessary negative impact on network performance.

- *Unicast* The alternative to broadcast that is available on most networks, is *unicast* communication. A unicast message is one that is sent from one machine to one other specific machine on a network. As we will see in the Products section of this chapter unicast delivery is used in most commercial Publish-Subscribe applications because it is the only delivery mechanism available of the three: broadcast, multicast, and unicast. The disadvantage to using unicast is that it has a significant impact on the performance of a publish-subscribe system in two ways. First, the machine that sends the data has to repeatedly send the same message impacting its CPU. Second, the same message is being sent over the same path using up bandwidth.

- *Multicast* An alternative to broadcast that provides many of the same benefits is *multicast*[?]. Multicast is a 1-to-N delivery mechanism similar to broadcast in that a single transmission of data satisfies all users. The difference from broadcast is that the routers along the path from the data source to the users maintain lists to determine where to send multicast packets. The users who receive data can be identified. Also, multicast can be made reliable [?] so that the delivery of data can be guaranteed.

As mentioned in the Intoduction, scalability is an important concern for the designers of any Publish-Subscribe system. However, scalability cannot come at the cost of overwhelming the network or unwanting users with unnecessary data transmission. For this reason, multicast is the solution that most reasonably solves both cost and scalability concerns.

The type of network for the Publish-Subscribe system is designed also impacts the decision for network data delivery. At the moment, most Wide Area Networks do not provide support for multicast. A significant cost comes from the memory needed in the routers to store state, and higher processing power is required for the processing of control messages. [2] [3]

2.2 Internet vs. Intranets

This section is yet to be written

- Give definition of Internet and Intranets. How is Publish/Subscribe applied to Internet vs. Intranets?
- Give examples.
- Problems and solutions

2.3 Channelization (matchmaking data with clients)

Channels have been a popular mechanism for delivering information to large groups of individuals. Examples of technology that use channels include satellite broadcast, multicast, and, of course, TV and radio have long sent information to individuals via channels. Some publish-subscribe systems depend on broadcast channels to deliver data to users. A problem that is common to many of them is solving the assignment of users and data items to these channels so as to make efficient use of the network resources and minimize delivery of superfluous data to users.

The goal of a publish-subscribe system is to deliver data to large numbers of users. However, there are two problems: 1) You don't want to send all updates to all users; and, 2) You can't unicast to individual clients. So, we are left with only one alternative, we have to group users and broadcast updates to only those groups with interested users. An important restriction is that each user must receive all updates to items specified in its respective profile.

The introduction of channels brings about two issues First, clients must be assigned to broadcast channels from which they will receive data updates. Second, as data is sent, it must be mapped to the broadcast channels. As harmless as these two issues sound, they create an enormous search space. There are n^c possible ways to assign c clients to n channels and there are 2^{np} possible ways to map p pages to n channels.

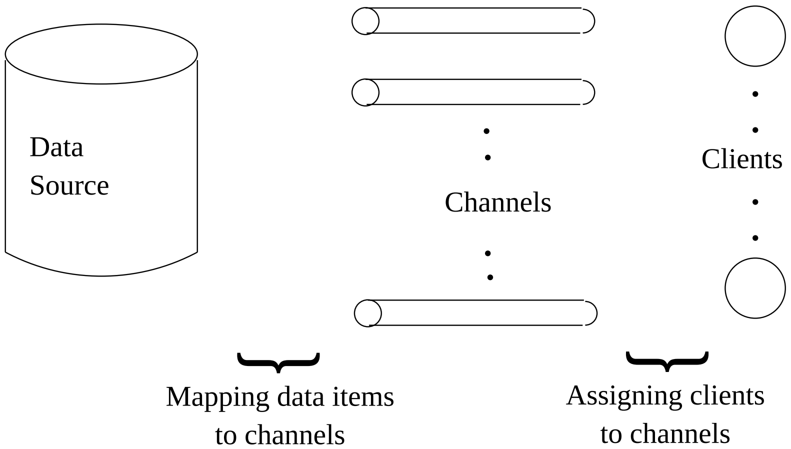

Figure 1.2: Channels & Issues

A good solution to the problem attempts to make an effective use of the limited available bandwidth and also tries to keep the clients happy while being scalable in the number of users. Since multiple users are assigned to each channel, users receive not only those updates represented in their respective profiles, but also updates intended for other clients assigned to the same channel. Clients are happiest when they are getting more of what they want and less of what they don't want. So, it is important to group similar clients (based on their profiles) on the same channel.

In [4] a K-Means clustering algorithm is is presented as a method to solve the channelization problem. An approach is presented that finds a locally optimal solution to the channelization problem. Each channel is considered a cluster of client profiles and the *k-means* method is used to partition the clusters. It works as follows. For each profile P_n in some cluster G_i, the algorithm switches P_n to another cluster G_j if it is more "similar" to the set of data items belonging to G_j. The algorithm stops when no profile can be moved from its current cluster to another.

A profile is similar to a cluster if it has a large overlap in the data items. The distance function described measures similarity as the percentage of data all clients assigned to the channel desire from the cluster G_k. The distance between a profile P_n and cluster G_k is the average decrease in this percentage if P_n is added to G_k. Thus, the smaller the distance, the more similar P_n is to the other profiles in G_k. Unfortunately, the k-means method has an unbounded running time.

2.4 Profiles & Continuous Queries

As discussed in the Introduction, in Publish-Subscribe systems users describe their interest in data in the form of a *profile*. Users submit profiles to the system and thereafter continuously receive data that is considered relevant. Profiles are usually stored at the data source. In this way, profiles are a type of continually evaluated query [13] [15] [14].

2.4.1 Information filtering

The model for Publish-Subscribe systems is based on continuously collecting new data items from data sources, filtering them against specific user interests (profiles), and delivering relevant data to users. In the internet environment where huge volumes of data are are being updated all the time and large numbers of users are interested in accessing the data, efficiency and scalability are important concerns. As previously described, Publish-Subscribe systems are event driven systems where data is sent to interested users as it is updated. One challenge to sending the data is figuring out which updates are interesting to users and to which users individual updates should sent. The first concept (figuring out which updates are interesting) is called information filtering.

Most push-based systems use simple keyword matching Information Retrieval (IR) techniques to match user profiles with new data items. This technique limits the user's ability to express interest and therefore increases the likelihood that users will receive more irrelevant data and less relevant data. These IR techniques have focused on the effectiveness of profiles rather than the efficiency of filtering. In other words, the IR community's solution does not scale.

2.4.2 Ways to Express Profiles

More recently, profiles are seen as queries whose form can range from a simple set of keywords to queries expressed in a high-level query language like XMLQL[NiagaraCQ]. It is the management of profiles that is interesting in the context of publish-subscribe. Profile modeling and matching has been studied in both the the Data Dissemination and Information Retrieval (IR) communities. (For a survey of Data Dissemination see [6] also see [][][] (Stan's papers on Dissemination)) Profiles in the Information Retrieval community have been used under the context of Information Filtering [16]. Typically sets of keywords are used to represent profiles and are intended for text-based systems where the data is unstructured. These models are can be classified as either Boolean or Similarity-based. In the boolean model an exact match is required for keywords attached by boolean operators. In the similarity-based model, matches are given a "similarity" value - documents are similar to a profile if the value is greater than a threshold value. Several similarity based approaches have been presented in the literature [][][][]. SIFT (The Stanford Information Filtering Tool) is a Publish-Subscribe system which originally used boolean profiles and was later changed to used a similarity based approach.

The SIFT system evolved to use the Vector Space Model. In the Vector Space Model [??] a document is represented as a vector of terms (a very common model for text representation). The vector, which may be weighted, represents the document's content. Likewise, a user's profile can be represented by a vector of keywords [][]. IR techniques try to correlate the vector that represents the document with the vector that represents the profile, and provide the user with the best (most relevant) matches. The IR community focuses their work on methods for determining the weights to give to terms in the document or the profile.

2.4.3 Continuous Queries

The database community has studied query-based profile models in the context of Continuous Queries (CQ). These are standing queries that allow users to get data when an update occurs in a database. Continuous Queries were studied specifically for relational databases in Terry et al for the Tapestry system[13]. The recent importance of dissemination technology has brought about interest in CQ for internet-scale information delivery.

Continuous queries can be categorized based on the the event that triggers their execution [NiagaraCQ]:

- *Change-based* continuous queries are triggered as soon as new data becomes available. As soon as data is changed the query is executed. An example of this type of continuous query is one that requests changes in the value of a stock price. Someone who actively trades stocks is very interested in changes in the price.

- *Timer-based* continuous queries are executed based on time intervals specified by the user or user profile. An example of this type of query is weather forecast information in Providence. Whereas the weather is always changing ever so slightly, a normal person does not need to receive weather updates any more frequently than once an hour.

The type of continuous query that a Publish-Subscribe system can handle has a significant impact on scalability. Change-based queries provide better response time and thus less-stale data, however, system resources are wasted when instantaneous results are not really required. Timer-based continuous queries can be supported more efficiently. An interesting insight noted in [NiagaraCQ] is that change-based continuous queries can be grouped more easily than timer-based queries, since users can specify various overlapping time intervals.

2.4.4 Case Studies

Two recent systems provide architectures and approaches for continuous queries in an internet-scale environment:

- The OpenCQ [15][14] work presents an architecture for an Continual Query system. The OpenCQ information delivery system has a three tier architecture consisting of a client, server, and wrapper. The OpenCQ server has three components: a trigger condition evaluator, several event detectors, and a CQ manager. The trigger condition evaluator and event detectors monitor data updates based on user specified interest thresholds and time constraints. The CQ manager coordinates tasks and communication between the client/server tiers and the server/wrapper tier. The client tier provides the user interface to the clients, management and coordination of requests, and also limited preprocessing of the query. OpenCQ talks with data sources via CQ wrappers. A wrapper is required for each data source because different data sources have different access methods and different data formats for representing query results. The wrapper is effectively a translator, converting queries into a format understood by the remote data source, and converting responses into a format understood by OpenCQ.

- The NiagaraCQ [17] system provides similar functionality to that of the OpenCQ system. NiagaraCQ was developed to query distributed XML data sets using a query language like XML-QL [18]. The NiagaraCQ system attempts to increase scalability for large numbers of users to submit continuous queries in a high-level query language. The assumption of the system is that many of the queries will be similar and therefore can be grouped together. The expected result of grouping queries is that computation can be shared, memory can be used more efficiently (common execution plans can reside in memory), and I/O costs can be saved (satisfying multiple queries at once). A new group optimization technique is presented which allows for new queries to be added to existing queries without regrouping of existing queries while requiring minimal changes to a general purpose query engine.

The above systems suffer from scalability problems. They are limited because as data changes, all profiles are matched against the change to the database, or the full new database, to determine if any new data is of interest to users. For large scale systems (internet-scale) with large numbers of users, and therefore large numbers of profiles, this approach is very limiting. NiagaraCQ approaches the problem by trying to eliminate similar queries, but that does little to impact the more general scalability problem (as the number of users increases, and the number and variety of queries that have to be satisfied increases, the traditional method of running all queries against the data set does not scale).

The next section addresses this particular scalability issue.

2.4.5 Indexing Profiles (Reversing the Role of Queries and Data)

This section is yet to be written

- Reversing the Role of queries and data (Indexing Profiles)
  - SIFT was the first to reverse these roles
  - In traditional DB systems the data items are indexed and stored and queries are applied individually.
  - In dissemination systems, large numbers of queries are stored and indexed and the data is individually applied. The data becomes the query.
  - Indexing Profiles
    * Sift
    * XFilter

2.5 Mobile Computing

The proliferation of mobile computing is a result of the availability of low-cost powerful portable devices and the development of high-speed wireless networking. The mobile environment brings about many interesting issues for network data management (footnote: for an excellent discussion of mobile computing and databases see [19]). The specific issues that impact all network data management systems that utilize mobile computing, including Publish Subscribe systems, are:

- *Network asymmetry* Most wireless networks provide much greater downstream bandwidth (from server to client) than upstream bandwidth.

- *Battery life/savings* Mobile devices like PDAs, cell phones, tablet computers, and laptop computers have batteries with very limited life. To give users the perception that the battery lasts longer, these devices usually go into a power-save mode, where they effectively shutdown.

When such a device is in this mode, it becomes disconnected from wireless networks and therefore cannot receive data being sent from a Publish-Subscribe system.

- *Network unreliabilty (connection and bandwidth)* Wireless networks are very unreliable. The amount of bandwidth between the device and a wireless access point at any given time can vary wildly due to any number of factors including interference by building structures and other devices, the number of other users currently connected to the same wireless network, and roaming transfers to network *cells* (a cell could be a cell in a cellular communication network or a wireless local area network access point).

- *Device memory limitations* Most mobile devices have serious memory limitations. In a publish-subscribe environment, where data is continuously being sent to interested clients, devices must be concerned with what data is most important to the user.

- *Screen size* Some mobile devices, such as PDAs and cell phones, have very small screens.

Each of these issues has an impact on the way data must be managed in a mobile environment. The network asymmetry, network unreliability, and power management issues make broadcast an attractive delivery mechanism but also creates a complication for the implementation of data management. This environment exacerbates the general concerns of Publish-Subscribe systems. Prioritizing data becomes very important: given that some clients might have limited memory and also might not always be connected to the network, the system should ensure that clients get important data with a higher probability.

Sending data periodically or aperiodically is another concern for publish-subscribe systems in a mobile environment. Periodically broadcasting data to clients allows for clients to be disconnected from the network for periods of time and to connect when data is scheduled to be transmitted. Aperiodic push makes more efficient use of bandwidth but at the cost of clients not receiving data.

The *Broadcast Disks* project proposed an architecture for periodic push in a mobile environment. The architecture attempts to exploit communication asymmetry by repeatedly and cyclicly broadcasting a stream of data as a storage device. There are two novel aspects to the broadcast disks work:

- Multiple broadcast streams (”disks”) with different latencies are superimposed on a single broadcast channel. The goal of this is to improve performance of non-uniform data access patterns and increase the availability of more important data by allocating more bandwidth to it.

- Mechanisms are presented for managing client storage resources to tailor caching and prefetching to perform efficiently with multiple disk broadcasts.

More Broadcast Disk discussion here (1 more paragraph)

3 TMP-Products

4 Software & System Architecture for Publish/Subscribe

The architecture of the event service describes the software components that realize the event service together with their connections. The description should emphasize the physical locations of components, the topology of their connections, and how those connections are controlled. Following we will mainly talk about the server topology.

4.1 Centralized server topology

A centralized topology means there is only a single event server in the system, And every client should register in this server. Clearly, it is the bottleneck of the scalability. When designing an architecture with multiple servers, we must also specify:

1. the connections among some servers, i.e. which pairs of servers can communicate directly, and

2. the protocol used in the communications between those servers.

These two elements decide a topology of the servers.

We also should note that the connections among servers are not necessarily the physical channels, but rather logical connections. In general, this is also the case with clients. When we say that a server X is connected to another server Y , we mean that X has Y's address and can send messages directly to Y. In the "physical" implementation, there may no a materialized connection(channel) between them.

4.2 hierarchical topology

The second way of connecting event servers is to use hierarchical topology. It is an extension of centralized architecture.

As shown in Figure 1.3 , each server has a number of clients that can be either publishers or subscribers, or other event servers. Besides these connections, every server(except the root) also has a special connection to its parent server. And for every server, the connection to the parent server is the only outgoing arrow.

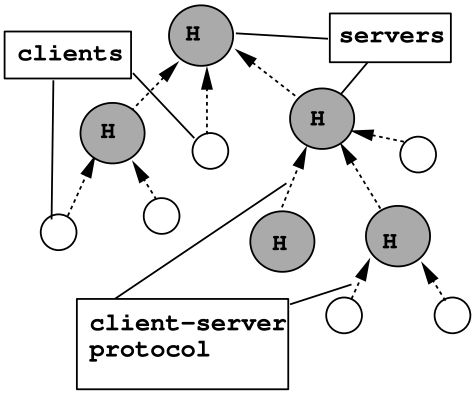

Figure 1.3: Hierarchical server topology

In this topology, the server-server and client-server communications share the same protocol. Thus, a server does not distinguish other servers from the publishers or subscribers among its clients. In the application, this means that a parent server will be able to receive notifications(published information) and subscriptions from all its clients, but it will send only notifications back to them.

The hierarchical topology is the natural extension of the centralized topology. It only requires an extension to the algorithm of the servers that can push information through the parent server. As for the configuration of servers, this topology allows entire subnets to join a community of servers by simply connecting their root servers to any server in the community. This topology is currently used in the USENET News network, in JEDI, in Keryx, and in TIB/Rendezvous.

However, the hierarchical topology has some main problems such as : the overloading of higher-level servers, and that every server is critical to the normal working of the whole system. In fact, a failure in one server will disconnect all the subnets reachable from its parent server and all its clients subnets from each other.

4.3 Acyclic peer-to-peer topology

In the acyclic peer-to-peer topology, all servers are the same. They communicate with each other as peers, thus using a communication protocol that allows a bi-directional flow of subscriptions and notifications. By “commu-

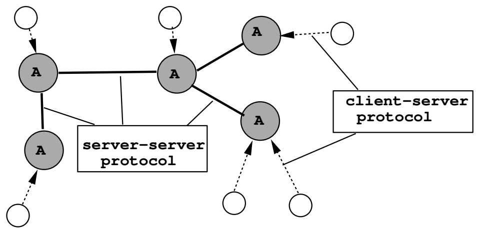

Figure 1.4: Acyclic peer-to-peer topology

nication protocol", we means the type and amount of the information that event servers exchange. This protocol is obviously implemented on the top of other communication mechanism such as, shared memory, SMTP or HTTP protocols. Above Figure 1.4 is an example of an acyclic peer-to-peer topology of servers. The different line styles between clients and servers and among servers means that the different kinds of communication being used.

Considering server-to-server links as non-directed edge, the tree of this topology has no cycle in it. It is very important to use a procedure to make sure this property is maintained when adding new ones to this topology since specific algorithms might rely on it. And that the incremental procedure suggested for the hierarchical topology can be used in this case as well. For example,we initialize every server X with at most one peer server to which X connects and add Y to X's direct peers. At the same time, send a request from X to Y. On receiving the request, add X to Y's list of peers. Every server will send the request once and be initialized once. And when it is initialized, make sure no redundant path is produced(no cycle is produced).

In a network-wide service, if each server is administered locally, this procedure might be not good. Other control protocols must be used to validate the configuration of servers. Also,just like the hierarchical topology, this topology suffers from the lack of redundancy in the connection graph. Because the connection graph is a tree, a failure in one server X isolates all the groups of subtrees which are directly linked to X.

4.4 Generic peer-to-peer topology

Without the constraint : no cycle is allowed from the acyclic peer-to-peer topology, we get the generic peer-to-peer topology. Similar as the acyclic peer-to-peer, this topology allows bi-directional communications between two servers. But for generic peer-to-peer topology, the network of connections

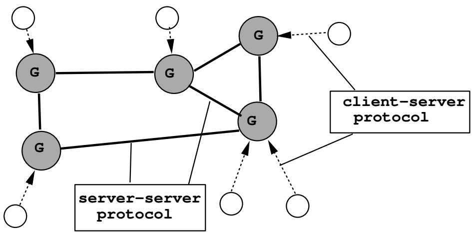

Figure 1.5: Generic peer-to-peer topology

among servers is a generic graph. There may be multiple paths between servers, See Figure 1.5 for an example.

The goodness of this topology is that it requires less coordination and have more flexibility in the configuration of connections among servers. Moreover, the redundancy in the connection between different servers in the network, makes it more robust with respect to the single server failures. The short-coming is that some special algorithms should be used to select the best paths without cycles.

4.5 Hybrid topologies

A network-wide service may have different requirements at different levels of network,such as the local-area network and the wide-area network. The hybrid topologies can satisfy their specific requirements. A hybrid network can has different topologies at different levels.

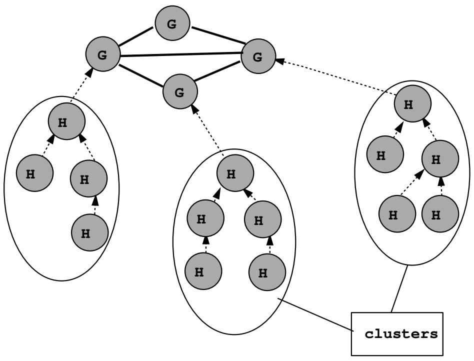

Figure 1.6: Hybrid topology: hierarchical/generic

For example, in a big company, system administrators may be able to manage the whole network of servers which includes their subnets, because they require a high degree of control and coordination in the administration of a set of subnets. So it will be better to use a hierarchical topology inside the cluster although the global network is generic topology. See Figure 1.6 for an example.

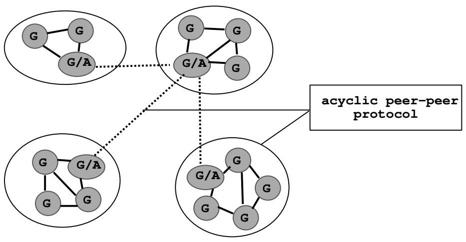

Figure 1.7: Hybrid topology: acyclic/generic

In other cases, Figure 1.7, the situation might be very different. We may need to design the whole system the opposite way. For example, assume that some clusters of subnets have a very intense traffic of local events, and we only want a small part of the events to be reachable from outside of the cluster for some special reason (e.g. the security). Then we can use a generic topology inside the cluster while design the high-level topology as acyclic graph.

For every cluster, there will be a "gateway" server which can filter the messages used for the protocol inside the cluster or change them to the protocol used among clusters.

Obviously, the architecture has a significant impact on functionality and scalability of a system. A centralized event service is relatively easy to implement complex filtering of notifications, but it is difficult to be scaled to a wide-area networks. The system with hierarchical structure is simple and effective in many cases. But it still has some fundamental shortcomings when implemented in the wide-area networks. It introduces unnecessary message traffic in its higher nodes in the hierarchy, and it has a single point failure in every node. On the other hand, a distributed topology seems to have more scalability, but at the same time, it is much harder to do the complex filtering. So we should choose the suitable topology according to the specific requirement of the networks.

5 Applications

People have been applying the technologies and architecture developed to publish/subscribe topics; the followings are some applications of the efforts:

5.1 Newsgroups/Mailing Lists

Newsgroup/mailing list is a simple kind of information dissemination service. At the newsgroup severs, there are many news that is constantly updated and divided into different groups, and the clients whoever wish to read the news submit profiles to the servers of their email addresses and lists of the newsgroups they wish to subscribe to. The users could also post messages to the newsgroup to inform other users of news. The severs keep the profiles as mailing lists, and according to the matched profiles the servers deliver the news either periodically or aperiodically to the clients through internet. Some severs might apply different technologies such as push to deliver the news automatically or let the clients pull the news to them. The system is a continuous operation, working 24 hours a day and 7 days a week ("24 by 7") as the news are constantly updated and delivered to clients. This way the news was published to the clients who subscribe to the newsgroup they are interested. There are some examples of the application such as USENET News (www.usenet.com), USATODAY E-mail Newsletters (www.usatoday.com), and the Brown university news (news.brown.edu). A problem with newsgroup/mailing lists application is that it does not provide detailed enough interest matching. A user whose information that does not exactly match certain lists will either receive too many irrelevant or too few relevant messages.

5.2 Stock and News

Stock and news application is similar to newsgroup/mailing lists application except that it has to remain real-time operation; thus, it requires the servers to build dynamic information update system within. Here the clients submit their profiles again to the servers, and the servers deliver the information clients required through internet, intranets, 1-to-N (broadcast, mulitcast) or channels, either by push or pull technology. The operational has to remain "24 by 7" also [ref]. Examples of the application are election results, Pointcast, TIBCO, etc.

5.3 Traffic Information Systems

Same as the stock and news application, the information has to be real-time most of the time. The information are distributed through 1-to-N (broadcast, multicast), internet, or channels, either by push or pull technology. The information is also distributed through mobile computing. (have yet to find examples)

5.4 Software Distribution

Many systems require to have their software updated frequently. For example, the software system at an investment bank functions for the security trading needs to be updated frequently and extensively, since securities trading is a hightly competitive business, and a one-minute delay could mean millions of dollars in lost profits; thus, it is advantageous to use the latest software. The servers could distribute the newest software updates to their clients through internet, intranets, multicast or channels, either by push or pull technologies. However, since some software contains the important issuse of security, some clients might need to submit their password before they are allowed to download the software. Also, the software distribution has to be continuous operation since many system need to remain operational "24 by 7" in order to accommodate the real world operation. Examples are Vitria, TIBCO, etc. [ref]

5.5 Battlefield Data Dissemination

In the battlefield, it is vital that the fighters are to receive the most updated and correct information from their superior. The information is distributed either through multicast or unicast, and the operation has to remain "24 by 7". [ref]

6 Products

Up to date, there are several systems based on publish/subscribe paradigm developed to disseminate information and data, they are all in one way or another similar to each other, and each system thrives to do better than the others. The following are some examples:

6.1 Pointcast

Just like the applications mentioned in the previous section (such as stock and news), there are many news and information the users desire to know at the Pointcast server. The clients first send their profiles to the server then gather the information they are interested.

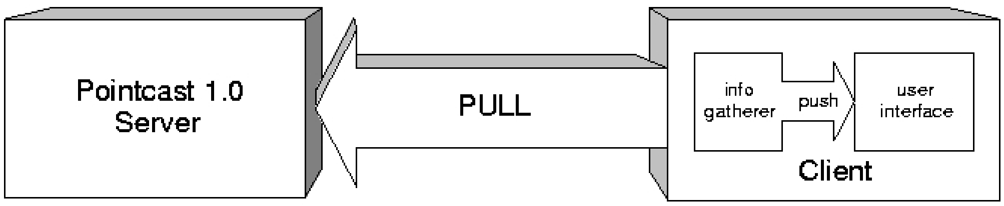

Figure 1.8: Pointcast 1.0

The above figure [ref] shows mechanisms of how the data was delivered to the users in Pointcast 1.0. In order to deliver the data to users, the Pointcast system first obtains profiles from clients which contain the subscribers' interests. Then the clients ask for the news that match their profiles and pull/download the customized infomation to their local computer system. The arrived data then is pushed to their computer screen [ref].

The Pointcast system was thought as one of the first push-based systems; however, as we can see from the above, the machanism that delivers data is actaully pull-based. This system is quite different from SIFT, as we shall see in the following that SIFT is push-based.

6.2 SIFT (Stanford Information Filtering Tool)

The SIFT system combines data management ideas from information retrieval with a publish/subscribe model to disseminate data. Here all kinds of data from a document source through a one-to-many broadcast medium was pushed to the system server, then with indexed client profiles, the server matches users' profiles with the arrived document, and push/distributes the information to users. Also, with the technology of indexing client profiles, which will match clients' profiles against newly arriving data, the clients get exactly what they are interested[ref]. The following figure [ref] shows a high-level SIFT architecture, which is cited from "A Framework for Scalable Dissemination-Based Systems" [ref]:

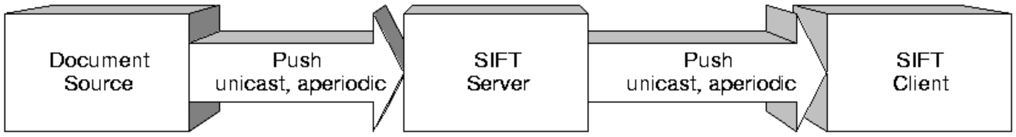

Figure 1.9: Shift Architecture

6.3 TIBCO (http://www.tibco.com)

TIBCO provides real-time e-business infrastructure software, which enable businesses to integrate enterprise applications, and efficinetly deliver personalized information through enterprise portals. The software is based on an architecture -The Information Bus- developed by Teknek Software Systems, Inc., the technology employs the mulit-cast protocal, with each broadcast message would contain a "Subject" identifier. This subject could indicate the nature of the message as well as the originator. At the client side, functionally would filter out messages based on the subject. For corporate sites messages could be filtered out before the firewall, at the client desktop and in between. This presumably offers advantages over plain multicast because it can be made reliable, thus guaranteeing delivery to users.

(figure)

6.4 Vitria (http://www.vitria.com)

Vitria is a commercial system that provides enterpise-capable publish/subscribe middleware products, which automate cross-enterprise business process and enable companies to conduct business electronically across corporate networks and over the internet. It provides the following benefits: the clients can increase the efficiency of their operations and lower operating costs by automating and analyzing their business processes in real-time, and quickly model and automate the business processes they need to bring new products and services to market.

6.5 Channels

Channels contains frequently updated information, users subscribe to channels by submitting their profiles. The information is automatically delivered through internet, intranets or multicast. Some of the examples are Marimba's Castanet, Netscape's netcaster, Microsoft's CDF.

6.5.1 Marimba's Castanet

Marimba's Castanet sends "Channels" from sever to clients, clients subscribe to the channels then periodic downloads the updates.

6.5.2 Netscape's netcaster

Netscape's netcaster integrate push technology, web channels, and castanet channels mechanisms to deliver information.

6.5.3 Microsoft's CDF (Channel Definition Format)

Microsoft's CDF allows publishers frequently update information to channels, CDF files on server specify clients request, then clients retrieve files periodically.

7 Related Work

This section is yet to be written

- Triggers & Active Databases
  - more general mechanism which can involve predicates over many data items and can initiate updates to other data items.
  - not optimized for fast matching of individual items to a large number of simple queries
  - see recent work: "Scalable Trigger Processing"
- RETE networks

Bibliography

[1] M. Franklin, and S. Zdonik, *A Framework for Scalable Dissemination-Based Systems* OOPSLA 1997: 94-105

[2] T. Ballardie, P. Francis, and J. Crowcroft, *Core Based Trees (CBT): An Architecture for Scalable Inter-Domain Multicast Routing*, in Proceedings of SIGCOMM '93 (San Francisco, CA, Sept. 1993), ACM, pp. 85-95.

[3] S. Deering, D. Estrin, D. Farinacci, and V. Jacabson, *An Architecture for Wide-Area Multicast Routing*, in Proceedings of SIGCOMM '94 (University College London, London, U.K.,Sept. 1994), ACM.

[4] T. Wong, R. Katz, and S. McCanne, *An Evaluation of Preference Clustering in Large-scale Multicast Applications*, in Proceedings of IEEE INFOCOM 2000, Tel-Aviv, Israel. March 2000.

[5] M. Franklin, S. Zdonik, *"Data in Your Face": Push Technology in Perspective* (Invited Paper) ACM SIGMOD Intl. Conference on Management of Data, Seattle, WA, June, 1998.

[6] M. Franklin, *Special Issue on Data Dissmenation*, IEEE Technical Committee on Data Engineering, Sept. 1996.

[7] A. Crespo, O. Buyukkokten, and H. Garcia-Molina, *Efficient Query Subscription Processing in a Multicast Environment* Stanford Department of Computer Science, Technical Report, 1998.

[8] T. Yan, H. Garcia-Molina, *SIFT - a Tool for Wide-Area Information Dissemination*. Proc. 1995 USENIX Technical Conference, 1995

[9] B. Oki, M. Pfluegl, A. Siegel, D. Skeen, *The Information Bus - An Architecture for Extensible Distributed Systems*, Proc. 14th SOSP, Ashville, NC, December 1993

[10] J. Wong, *Broadcast Delivery*, Proceedings of the IEEE, 76(12), December, 1988.

[11] H.D. Schewtman, *CSIM: A C-based Process Oriented Simulation Language*, Proc. of the Winter Simulation Conf., 1986.

[12] A. Chan, *Transactional publish/subscribe: the proactive multicast of database changes* Proceedings of ACM SIGMOD international conference on Management of data, 1998

[13] D. Terry, D. Goldberg, D. Nichols, and B. Oki *Continuous Queries over Append-Only Databases* SIGMOD 1992; 321-330.

[14] L. Liu, C. Pu, R. Barga, T. Zhou. *Differential Evaluation of Contiunual Queries*, ICDCS 1996: 458-465.

[15] L. Liu, C. Pu, W. Tang. *Continual Queries for Internet Scale Event-Driven Information Delivery*, TKDE 11(4): 610-628 (1999).

[16] P.W. Foltz, S.T. Dumais, *Personalized information delivery: an analysis of information filtering methods*, CACM,35(12):51-60,December 1992.

[17] J. Chen, D. DeWitt, F. Tian, Y. Wang, *NiagaraCQ: A Scalable Continuous Query System for Internet Databases*, Proc. ACM SIGMOD Conf., Dallas, TX, May, 2000.

[18] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu. *XML-QL: A Query Language for XML.* http://www.w3.org/TR/NOTE-xml-ql.

[19] D. Barbara textslMobile Computing and Databases - A Survey, IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 1, 1999