Channelization for Publish-Subscribe Systems

Don Carney

September 18, 2000

1 Introduction

Publish-subscribe systems are becoming a popular way to disseminate information. They are a type of *push*-based system. Push technology comes from a very simple idea: rather than users explicitly requesting (*pulling*) the information they need, data can be sent without them specifically asking for it. In the publish-subscribe model we describe in this paper, users subscribe to updates published by a data source. The model has several important characteristics. It is *event-driven* as data items are sent as soon as they are updated. Updates are *broadcast* (*1-to-N*) and data transmission is *aperiodic* as there is no pre-defined schedule for sending or updating data. We are most concerned with publish-subscribe systems that have a large number of users who have significant overlap in their interests.

Advances in broadcast technology have enabled publish-subscribe. Broadcast technology has a long history of use. We have seen it used in radio and television for many years and more recently have benefited from its use in satellite broadcast and multicast. The major advantage of broadcast is its ability to do 1-to-N transmission where the transmission of a single item can satisfy an arbitrary number of clients. Also, its scalability permits little or no incremental cost for each additional user.

Essential to any publish-subscribe system are *profiles*. A profile describes a user's interest in the data at the data source. In our system it indicates the specific items that a client wants to receive when those items are updated. The profile is stored at the data source and can be thought of as a continually evaluated query.

Our publish-subscribe system will depend on broadcast channels to deliver data updates to users. The specific problem that we attempt to solve is 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.

2 Problem

The goal of our publish-subscribe system is to deliver data updates to a large number of users (Figure 1). However, there are two problems: 1) We don't want to send all updates to all users; and, 2) We 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.

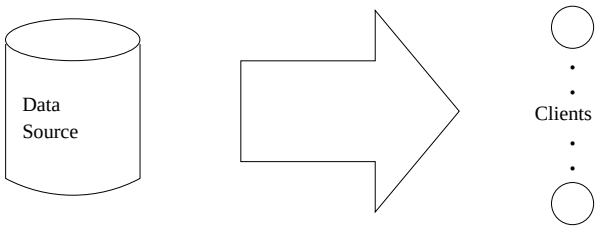


Figure 1: Push: Sending Data Updates to Users

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. A channel in our model is a transmission medium with a fixed bandwidth.

The introduction of channels brings about two issues (see Figure 2). First, clients must be assigned to broadcast channels from which they will receive data updates. Second, as data is updated, 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.

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. In the model that we discuss herein, each data item has an update frequency representing its expected bandwidth requirement and therefore its expected additional load on each channel to which it is assigned. Since multiple users are assigned to each channel, users will 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

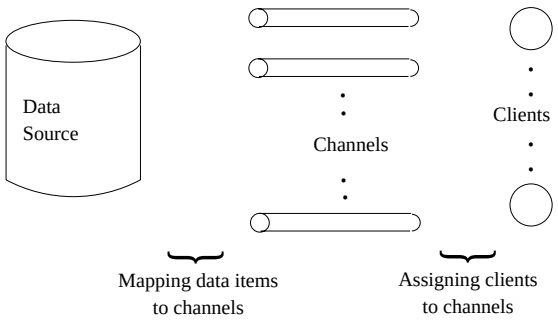


Figure 2: Channels & Issues

of what they don't want. So, it is important to group similar clients (based on their profiles) on the same channel.

We now define two metrics we will use to determine if a good solution is given for the problem. *Fluff* represents the items that a user receives that are not specified in its profile. *Maximum Channel Load* is the load on the maximum loaded channel. A publish-subscribe system is as weak as its most loaded channel. As the rate of data updates increases, the channel with the heaviest load will break first. A good solution will minimize both fluff and max channel load. (These metrics will be further defined as they relate to simulation experiments discussed in later sections).

3 Simulating the Publish-Subscribe Model

To better understand our publish-subscribe model, we constructed a simulator, based on the architecture shown in Figure 3. We used a standard simulation package, CSIM [7], as a tool to develop our simulation study. As data items are updated, they are filtered by the *Profiler* for those items that exist in profiles. If the items are in profiles, they are then mapped by the *Mapper* to channels where clients interested in the items are assigned. The simulator has several variables, or knobs, which can be set or varied during experiments. The knobs are listed below:

- *Number of Clients:* The number of users.
- *Client Interest Distribution:* This parameter represents the distribution of interest that clients have in database pages and it is manifest in the pages selected for each client's profile. A Zipf distribution is

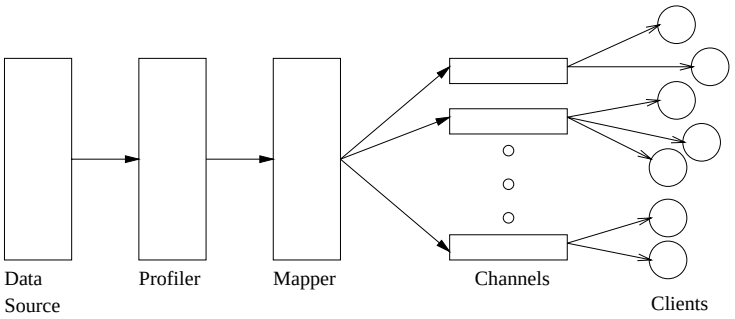


Figure 3: Simulation Model

commonly used to model non-uniform data access. As its parameter θ increases, the access pattern becomes increasingly skewed. The probability of accessing any page i is proportional to $(1/i)^{\theta}$ where $0 \leq \theta \leq 1$.

- *Profile Length:* This parameter represents the maximum number of pages that can be contained in a client's profile.

- *Offset of Client Interest:* The distributions for client interest and page update are distributions over the set of all pages in the database. The starting point for the page update distribution is page 1. The starting point for the client interest distribution can vary. This parameter represents the fraction of the database that the client interest is offset from page 1. For example, if the database size is 1000 pages and the offset is 0.25, then the client interest distribution starts at page 251. Note that an offset client interest distribution will wrap (i.e. all pages are still included in the distribution)

- *Number of Channels per Client:* This parameter describes the number of channels from which a client is allowed to receive updates. For the results presented here, this parameter is fixed at 1.

- *Number of Channels:* This parameter is obvious.

- *Per Channel Capacity:* This is the amount of bandwidth allocated to each channel.

- *Mapping Algorithms:* These are the focus of our research and are discussed in detail in the section 5.

- *Database Size:* This parameter defines the number of data items contained in the data source.

- *Page Update Distribution:* This parameter represents the distribution of update frequencies of pages in the database. Similar to the client interest distribution this parameter uses a Zipf distribution.

4 Problem Analysis

In this section we discuss how the previously described parameters affect our publish-subscribe model.

4.1 Client Impact

The number of clients affects the system in one significant and obvious way. As the number of clients increases, the maximum channel load should generally increase or at least stay the same.

As we have noted previously, and will see throughout the system analysis, page replication is one of the most important influences on system performance. As the number of clients increases, replication increases rapidly at first, then levels off. The law of large numbers comes into play when there are many clients - the probability that a channel exists that satisfies a client increases as the number of clients increases. (see Figure 4)

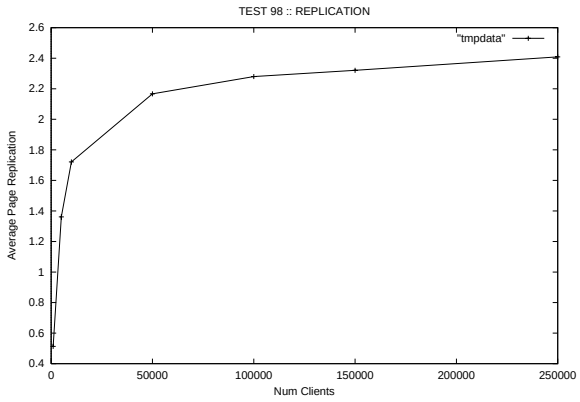


Figure 4: Page Replication

When client interests are in line with with the server distribution (i.e.

offset == 0.0), clients will be most interested in the most frequently updated pages. Most client-to-client overlap will occur in the high frequency pages.

Overlap is the main cause of page replication. If a large number of clients are interested in a page, then in the current model (1 channel per client) two things can happen:

1. that page can be assigned to one channel and all other pages that are of interest to all these clients are also assigned to that channel

2. there is not enough space on the channel to do 1., instead, that page is replicated on multiple channels, and the other pages are distributed among the channels

It is important that the pages with the highest update frequencies be replicated as little as possible.

4.2 Impact of Database Size

The size of the database has an interesting relation with the number of clients. Note that the maximum possible number of clients that could exist grows exponentially as the number of pages in the database increases. The actual maximum is equal to

$$\binom{N}{P}$$

for N pages in the database, and P pages per profile (See Figure 5).

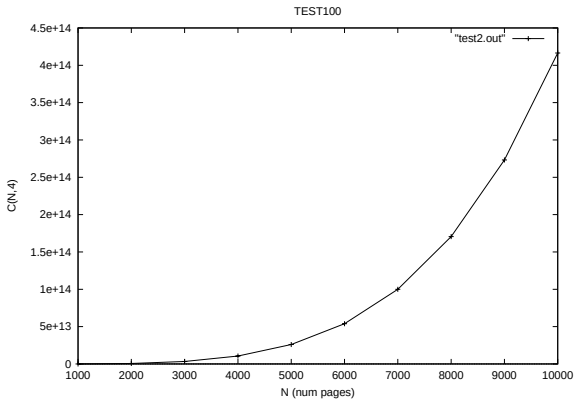


Figure 5: Number of Clients vs. Database Size

So, as the number of pages in the database increases, the overlap [1] between any two clients, given that the number of clients remains the same, decreases (See Figure 6). For the moment, disregard the different plots, and pay attention to the shape of the curve, noticing that as the number of pages increases, the client overlap decreases rapidly. (Note: this figure will be referred to during the discussion on Client Skew)

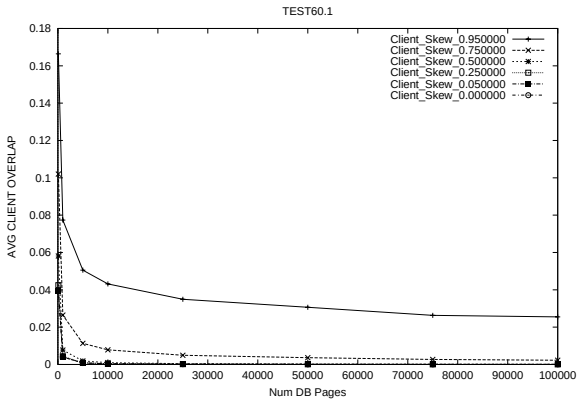


Figure 6: Client Overlap vs. Database Size

4.3 Impact of Number of Channels

Increasing the number of channels means that fewer clients need to be assigned to each channel and correspondingly, the maximum load on the channels should decrease. However, as clients are spread across a larger number of channels, replication will increase.

4.4 Impact of Server Skew

As server skew increase, we are creating a hot zone in the database. In that hot zone, pages are updated frequently. Note that the update frequency follows a Zipf distribution with the corresponding skew.

System variation when server skew changes is very dependent on what client profiles look like (client skew, offset). If clients are interested in hot

[1] Overlap is calculated as the average number of pages that overlap between any two clients. So, if client A has 3 pages in its profile, and client B has 4 pages in its profile, and they have one page in common, then client A's overlap with client B is 0.33, and client B's overlap with Client A is 0.25.

pages, then as server skew increases, channel load will increase. In general, when client skew and server skew are the same, max channel load is at its highest. This is because all clients are most interested in the frequently updated pages, and as a result, those pages require more replication.

4.5 Impact of Client Skew

As client skew is changed, the general interest of all clients changes. Increasing client skew means that clients are becoming more similar. That is, they are being clustered, and there is more overlap in their profiles (See Figure 6). Decreasing client skew means that clients become more dissimilar (less overlap).

System variation when client skew changes is highly dependent on server skew and profile offset.

4.6 Impact of Profile Offset

Profile offset is the fraction of offset in client interest from page 1, which is where the page update distribution always starts. Also, when offset is not equal to 0.0, client interest wraps at 1.0. As offset approaches 1.0, page 1 becomes more interesting. So, as profile offset moves from 0.0, client interests are clustered around pages that are not in the hot zone of the database. Whereas there will be the same overlap in client interests, the average client bandwidth requirement will decrease. Even though clients have the same size profiles, the pages that compose the profiles each have lower update frequencies and thus take up less bandwidth.

4.7 Impact of Profile Length

As profile length increases two things happen. First and most obvious, the average per client bandwidth requirement increases. Second, profile overlap increases. Figure 7 shows the relation between profile length and overlap. As profile length increases, overlap increases. We will see that it is more difficult to separate clients, and as a result more page replication will be required.

Figure 7 also depicts the significance of client skew and its impact on this system. As client skew increases, client interests become more tightly clustered, and therefore more overlap between profiles. Note also, that the number of possible clients grows quickly as the profile length increases.

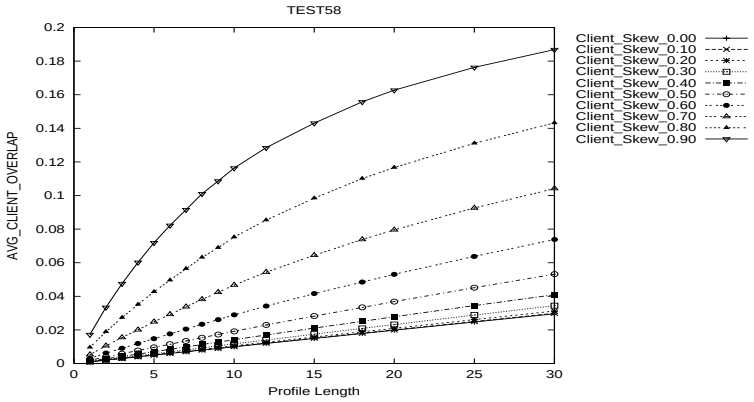


Figure 7: Client Overlap vs. Profile Length

5 Approaches (Mapping Algorithms)

This section describes the different algorithms developed to solve the channelization problem.

5.1 Exhaustive Search

The exhaustive search looks through all possible assignments of clients to channels. Remember that there are n^c possible possible assignments (c clients and n channels). For example, a system with 50 clients and 4 channels has 1267650600228229401496703205376 possible assignments of clients to channels (that's more that one nonillion!). This algorithm is, of course, impractical for large systems with thousands of clients and possibly hundreds of channels.

5.2 Random

A random algorithm was developed to demonstrate the faults of a completely brain-dead mapping algorithm. However, we will see later that other smarter, more time-consuming algorithms do not perform much better than this one. The random algorithm is simple: each client is assigned to a random channel, and its respective profile items are mapped accordingly.

5.3 Best-Fit

The best-fit algorithm was developed as a smarter alternative to the random algorithm. First, each channel is seeded with a random client. Then, until all clients are assigned the algorithm greedily assigns the "best-fit" client to the least loaded channel. The best-fit client is the client who, if assigned to the channel, would contribute the least load to the channel.

5.4 K-Means Clustering

The K-Means clustering algorithm is based on the work presented in Wong, Katz, and McCanne [1]. They present an approach 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. Unfortunately, the k-means method has an unbounded running time.

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.

5.5 Association

The development and implementation of the algorithms described in sections 5.3, 5.2, and 5.1 taught a very important lesson about this problem: replicating the most frequently updated pages is has a negative impact on fluff and max channel load because they require higher bandwidth. This lesson provided the intuition for the *association* algorithm which is to greedily assign items to as few channels as possible based on unassigned profiles and previously assigned items. The heuristic is to assign the most frequently updated page first. The algorithm works as follows. Assign the most frequently updated page to as few channels as possible. Next, assign the next most frequently updated page given that all higher frequency pages are already assigned. And, so on. The algorithm is wrapped with a binary search to minimize max channel load (iteratively apply algorithm and in each iteration we adjust the channel capacity). Pseudocode for the Association

Algorithm is shown in Figure 8.

```
exclusion_set = NULL;

main
{
    solveR ( set of all profiles )
}

solveR ( profile_set )
{
        while ( profile_set is not empty )
        {
                page = page in profile_set with highest update frequency that is not
                        in exclusion_set
                assoc_profile_set = profiles in profile_set that are associated by page
                chn = findBestChannel( assoc_profile_set )
                if ( chn > 0 )
                        putOnChannel( chn, assoc_profile_set )
                else
                {
                        exclusion_set.insert(page)
                        solveR ( assoc_profile_set )
                        exclusion_set.remove(page)
                }
        }
}

findBestChannel( profile_set )
{
        find channel with most (weighted) overlap with pages in profile_set
        if ( profile_set does not fit on any channel )
                return -1
        else
                return channel
}
```

Figure 8: Association Algorithm

6 Results

We conducted several simulation experiments to compare the performance of the different algorithms. The parameters used in the simulations are listed in Table 1. The exhaustive search algorithm is not represented in the results that follow because it is implausible to find the optimal solution even for the smallest experiment (1000 clients). Note that we are concerned with the relative performance of the different algorithms with regard to the *fluff* and *max channel load* metrics. Also, we are concerned with the scalability of the algorithms in the number of users.

Figure 9 shows the relative performance of the algorithms with regard to fluff. The plot shows some interesting and surprising results. First, note that both the simple *random* and significantly more complex *best-fit* algorithms provide solutions that are comparable in fluff. They each perform

Table 1: Table of Constants

Number of Clients	1000 - 50000
Client Interest Distribution (Zipf θ)	0.95
Profile Length	4
Profile Offset	0.0
Number of Channels	4
Number of DB Pages	1000
Server Update Distribution (Zipf θ)	0.95

poorly because neither algorithm tries to group profiles or minimize replication of frequently updated pages. Next, we note that the *k-means* and the *association* algorithms perform similarly. It is very interesting that the association algorithm provides a solution that is so close to a locally optimal solution. The reason is that grouping clients based on frequently updated pages is a very good heuristic.

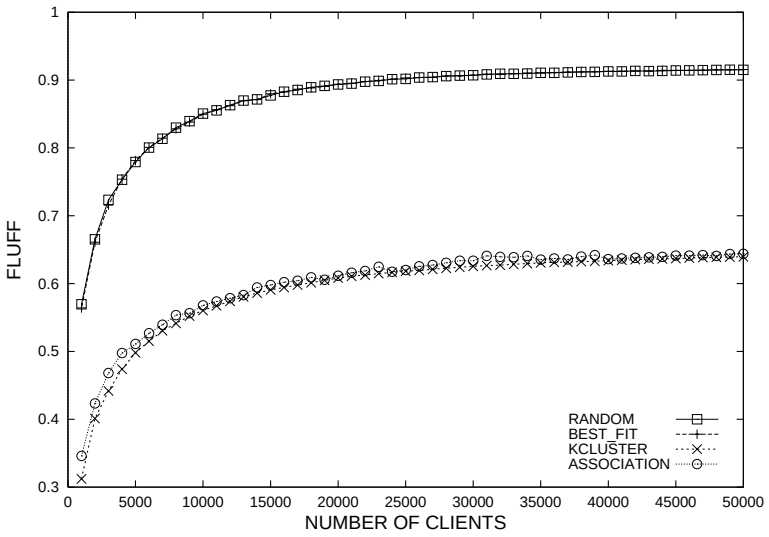


Figure 9: Fluff

Figure 10 shows the relative performance of the algorithms with regard to maximum channel load. Here we see that only the *association* algorithm

provides a good solution. Even the *k-means* is performing as poorly as the *random* algorithm. The reason is simple. Only the association algorithm is concerned with minimizing the maximum channel load. Recall that it is wrapped with a binary search for the best max channel load. The next result shows that it would be impractical for the other algorithms to also optimize for max channel load in a similar manner.

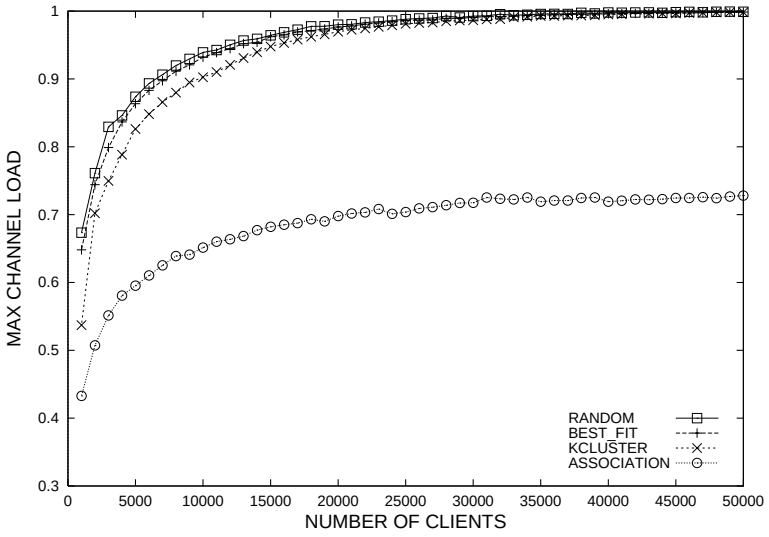


Figure 10: Maximum Channel Load

Figure 11 plots the running times for this series of simulations. Since *k-means* is a local search and has an unbounded running time, we set the number of iterations of the search to 20, though we found that in most cases, the search converges in fewer than 5 iterations. We see in this figure that as the number of clients increases into the tens of thousands, other solutions become impractical. For 50000 clients, the k-means algorithm requires more than 2 hours to complete an assignment whereas the association algorithm provides an assignment that is comparable in terms of fluff with lower maximum channel load in 105 seconds. This makes the association algorithm a viable solution for publish-subscribe systems that require on-line mapping of clients and pages to channels.

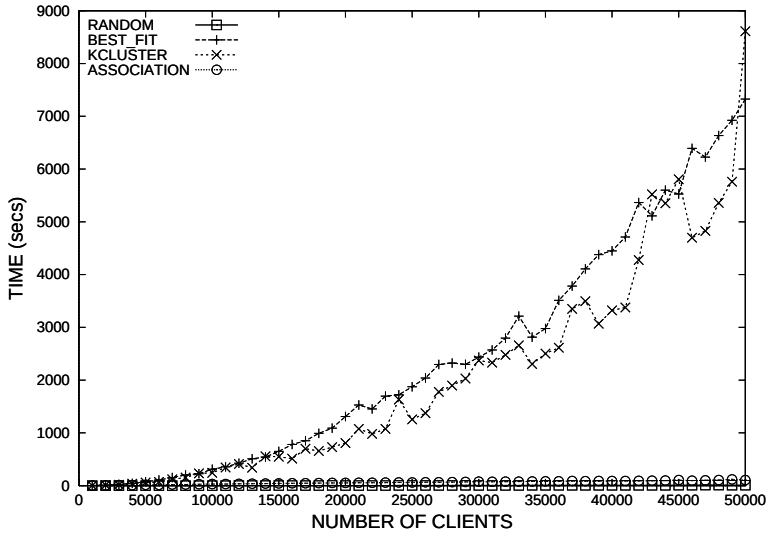


Figure 11: Algorithm Running Time (seconds)

7 Sensitivity Analysis

In this section we show the Association Algorithm's sensitivity to changes in the various parameters. As in the experiments shown earlier the sensitivity tests are performed as runs of the simulator. Each point on a plot is a separate run of the simulator. Each test presents the variation of two parameters while all other parameters remain constant. During any simulation those parameters that remain constant are set as specified in Table 2.

Table 2: Table of Constants

Number of DB Pages	1000
Server Skew (Zipf)	0.95
Client Skew (Zipf)	0.95
Number of Clients	10000
Profile Length	4
Profile Offset	0.0
Number of Channels	4

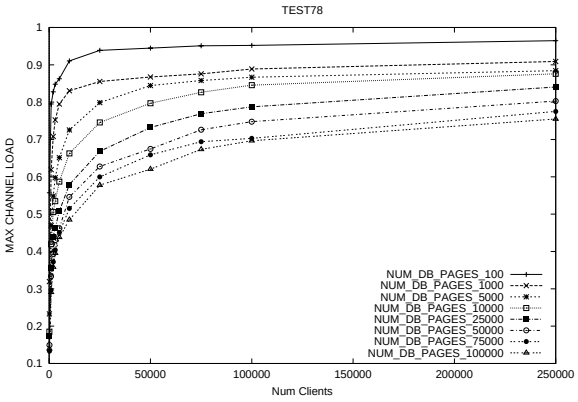


Figure 12: Number of Clients and Size of Database

We start by looking at how changes in the client characteristics are impacted by changes in other parameters. What is very interesting in all of these graphs is that as the number of clients increase beyond a certain point, with all other parameters constant, the maximum channel load is not significantly impacted. That is, after a certain point for any given simulation, adding more clients, has minimal impact. This is very interesting because it shows that solutions are not necessarily dependent on the scale of the client population. Instead, we see that some of the other parameters have more of an impact.

Figure 12 shows what happens as the number of pages in the database changes. The number of clients vs. the total possible number of clients gets smaller as the number of pages increases (see Figure 5). Combinatorics at work here: profile length 4 : C(1000 pages, 4) $\ll$ C(10000 pages, 4). Recall that client overlap decreases as the size of the database increases. (see Figure 6). This is because there are more pages and the number of clients remains the same. Something worth noting for these simulations, is that pages have frequencies of update, and for a database, the frequencies must add up to 1.0. So, as the number of pages increases, the average size (frequency) of a page decreases.

Figures 13 and 14 show what happens as the skews of the server and client distributions change with changes in the number of clients, respectively. The plots are relatively simple. For server skew, the weights of the most interesting pages are increasing as the server skew increases. And with client skew, client interest moves towards higher frequency pages as client

skew increases.

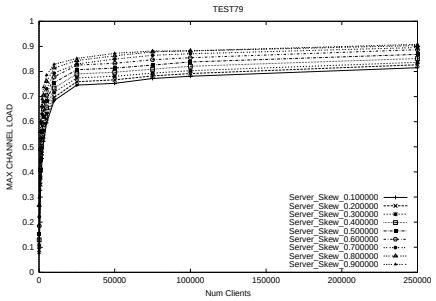


Figure 13: Number of Clients and Server Skew

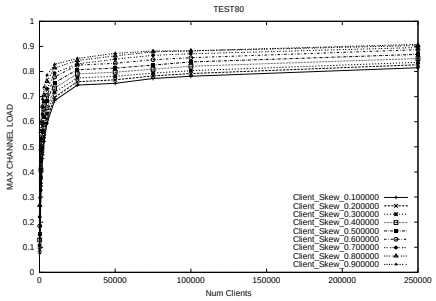


Figure 14: Number of Clients and Client Skew

Figure 15 is an interesting plot in that it shows that the length of profiles has a significant impact on the maximum channel load. As we saw earlier (Figure 7) there is a good reason for this effect. Whereas, the number of possible clients grows quickly as the profile length increases, overlap between profiles also increases.

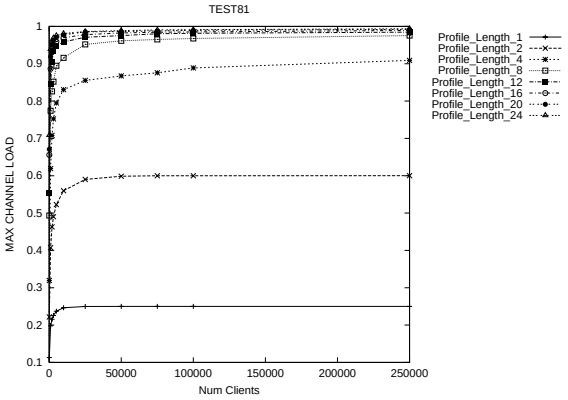
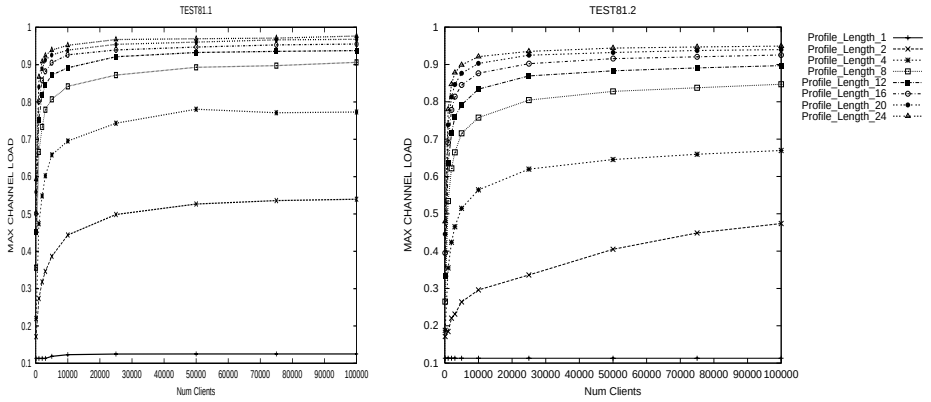


Figure 15: Number of Clients and Profile Length

As can be seen in Figures 16(a) and 16(b) the relative positions of the plots are very dependent on the number of channels available. Increasing the number of channels means that fewer clients need to be assigned to each channel. As the number of channels increases, maximum channel load decreases, however, replication increases.



(a) Number of Channels = 8 (b) Number of Channels = 16

Figure 16: Number of Clients and Profile Length

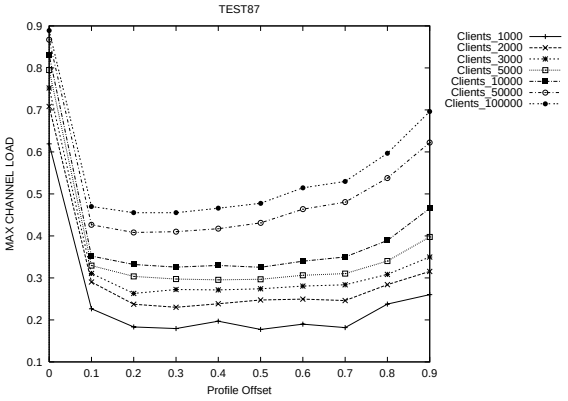


Figure 17: Number of Clients and Profile Offset

Figure 17 shows how the number of clients and the profile offset impact each other. Profile offset starts at 0.0 and client interests are in highest frequency pages. The frequencies of the most interesting pages drop drastically as soon as offset increases and the high frequency pages are in few profiles. As the offset increases (moves towards 1.0) the number of profiles which include the high frequency pages increases. Note that the distribution

of interest wraps at 1.0.

As the number of database pages increases, the maximum channel load decreases rapidly at first, then decreases slowly. Recall from Section 4.2 that client overlap also decreases rapidly at first then slowly as the database size increases. And, as client overlap decreases, the need for replication decreases, thus channel load decreases. It is important to note that as the size of the database increases, the average page update frequency decreases. Figure 18 provides another view of what we saw earlier (Figure 12).

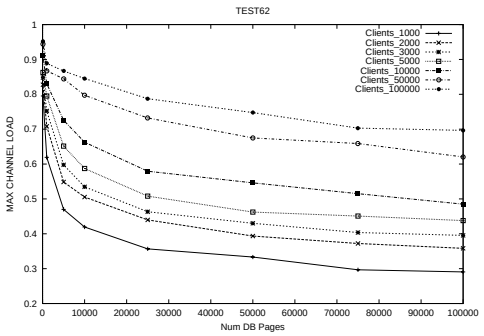


Figure 18: Database Size and Number of Clients

Recall from earlier discussions that increasing the number of channels allows clients to be spread out at the expense of replication. Of course, the benefit is that max load decreases and as we see, the general trend as the number of channels increases is that channel load decreases. Figure 19 shows the impact of increasing the number of channels on page replication.

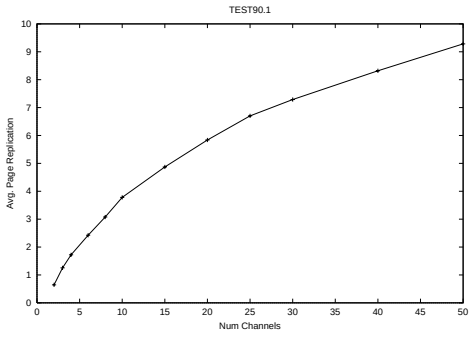


Figure 19: # Channels vs. Page Replication

Figure 20 shows an interesting graph that demonstrates how the system reacts to changes in the number of channels. Note the significant impact that profile length has on the system as the number of channels increases.

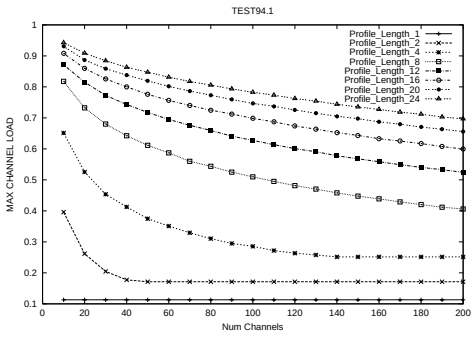


Figure 20: # Channels & Profile Length

8 Other Work

Some preliminary work has been performed on the study of allowing clients to receive updates through multiple channels. We have seen experimentally that there is a tradeoff between client happiness (fluff) and network resource usage. In particular, as clients are permitted to receive data on more channels, max channel load decreases but fluff increases.

9 Related Work

In this paper we compare several algorithms for the publish-subscribe channelization problem. The most directly related previous work on channelization algorithms is presented in [1]. The problem is presented as a clustering problem and the *k-means* local search method is used to find a solution.

The publish-subscribe dissemination model has been presented in the SIFT system [4] and the Information Bus [5]. A taxonomy of data delivery options is presented in [2] and early work on broadcast delivery is presented in [6]. The merging of query subscriptions where multicast channels are used to deliver data is examined in [3].

Publish-subscribe systems are not only studied by researchers, they have also provided several businesses with their main product. Two such companies are TIBCO [8] and Pointcast.

10 Future Work

Several of the broadcast technologies that could be used to deliver data for a publish-subscribe system allow users to receive data through multiple channels simultaneously. This makes the problem even more intractable and leaves several open questions. How many channels should a client be able to listen to? Should the number of channels that clients listen to be variable, allowing some clients to receive on more channels than others?

There is also the question of dynamic clients. It is conceivable that the number of clients could change while the system is running, rendering the initial assignment inefficient. New clients could be added, or clients could decide to turn off the channels. It may also be the case that clients may want to dynamically change their profiles.

11 Conclusion

In this paper we described the channelization problem for publish-subscribe systems. An architecture was presented from which a simulation model has been built. The simulation model provided a test-bed upon which to develop and test various mapping algorithms. We compared the algorithms using several metrics and found that our association algorithm provides an effective and efficient solution to the problem.

References

[1] 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.

[2] 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.

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

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

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

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

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

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