# Pervasive Computing: Supporting Group Activities

Dominic Jonak (dom@cmu.edu)
Advisor: Peter Steenkiste
(prs@cs.cmu.edu)

May 2, 2003

## Abstract

We have developed a group scheduler which allows users to specify personal preferences and request a suggestion for an activity and group of partners. This service confronts three major challenges; coordinating multiple users, modeling user intent, and maintaining privacy. We overcame these obstacles by using a central web server and optimizing a heuristic. We also describe potential attacks against this system and show how to defend the data by injecting random noise.

## 1 Introduction

We have developed a group scheduler which allows users to specify personal preferences and request a suggestion for an activity and partners. This system demonstrates how private data from many sources can be used to answer a query while revealing little personal information.

For example, suppose we have two activities (both of which require two people) and two users in the system. Alice is very agreeable, she likes both activities and Bob. Bob is ambivalent towards the activities and doesn't get along with Alice.

Now a third user, Chuck, enters the scene. Chuck is new to the area and doesn't know anyone, but he can specify his preferences towards both activities. When he requests a suggestion, the scheduler will most likely match him with Alice since she rated both activities favorably.

We could guarantee a very effective system by publishing all the data. Then Chuck could make what he considers the perfect choice. But Alice and Bob would not be happy; their private data has been disclosed.

On the other extreme we could guarantee privacy by never revealing any data. Now Alice and Bob are sure that no one knows their preference, but the system is useless for Chuck.

We take a middle position by using a trusted third party. This takes the form of a website where each user can enter their private data. These ratings can then be accessed only by the server to generate a suggestion.

On behalf of a user, the server will rank all potential matchings. Each suggestion is given a numerical score using a heuristic. Only the highest ranking result is returned.

To protect against malicious users, random noise is added. The noise prevents attackers from accurately reproducing the database. Furthermore this element of non-determinism turns out to improve the quality of the system.

The rest of this paper is organized as follows. In section 2, we compare the scheduler to similar modern systems. In section 3, we highlight the major challenges and briefly sketch the steps we took to confront

them. In sections 4 and 5, we discuss in detail how the system works. In section 6, we describe an attack by a malicious user. In section 7, we evaluate how effective and safe this system is. In section 8. we suggest future lines of research and development. And in section 9, we draw our final conclusions.

# 2 Related Work

Multi-user services have been online for over a decade[1] and are still evolving. Modern systems have begun to use private data and have taken some measures against malicious users. We present two common services; online dating and partner matching for games. In both arenas, the designers are using systems that similar to our own, but have fundamental differences.

## 2.1 Online Dating

There is a preponderance of online dating sites available on the web. These clearly involve some sensitive information and all rely on a matching heuristic. Some of the more successful services use a very sophisticated model[5] but they don't reveal how their algorithms work.

Our scheduler is different in a number of ways. Firstly there is a choice of activities beyond dating. Second we allow for different sized groups. Even within the same activity, groups of various sizes should be considered. Finally these systems typically have a binary privacy model where data is categorized public or private. We allow for sensitive data which is critical for making a good match but should not be revealed.

## 2.2 Online Gaming

Online gaming is becoming a large market and many new games offer matching systems. These are becoming more advanced and take into consideration a growing list of factors. Some use skill level to ensure a fair game and even allow players to set preferences for game options[4].

While they do share many characteristics with our scheduler, there are two major distinctions. Firstly they fail to address privacy concerns, which admittedly may not be an issue in a game setting. The other key difference is that these systems are tailored for each game, so once again the activity has already been chosen by the users.

# 3 Major Challenges

The scheduler gathers data from many private sources in order match up users. It must sift through a large collection of preferences to generate these matchings. Furthermore we assume that all data involved is sensitive, in other words, no user wants their preferences revealed. This presents three major challenges; coordinating the users, generating suggestions, and maintaining privacy.

## 3.1 Coordination

Each user can enter their preferences for activities and other people. They can also request a suggestion which returns an activity and group of people. This matching may include anyone in the system including people that user does not know.

We use a trusted third party that has access to all the data. This takes the form of a web server with a database backend.

Using a third party ensures that we can find matchings involving any user or activity. Since it is trusted by everyone, we also have access to everyone's data and can produce a globally optimal solution.

## 3.2 Generating Suggestions

The end-user receives the matching and is the ultimate test for what constitutes a good result. This small piece of information must be generated from all the data in the system.

We model the subjective value of a matching with a deterministic heuristic. In this way every possible matching can be assigned a score. The scheduler optimizes this heuristic over all matchings and only the highest ranking one will be given to the user.

Clearly a simple heuristic can not accurately model the user's intent. No heuristic can generate optimal results because every user's notion of the "best" activity will be different. Furthermore since the user can not access the entire database, they are unable to determine what the best match for them is. Consequently it is crucial that the heuristic comes close and provides reasonable values.

### 3.3 Privacy

Multiple users have private data that should not be revealed. In this domain, some of the private data will be exposed off-line when people actually get together. However even then only a limited number of preferences will be revealed with low accuracy.

All the pieces of data stored in the system are highly interdependent. Any change by any user could potentially impact the suggestion given to others. Also note that each suggestion reveals some information about the data used to generate it.

For these reasons, it is critical that the system be protected against malicious users. It is not practical to completely block any malicious usage, instead we aim to limit the amount of information they can extract

To address this challenge we inject random noise into the data used by the heuristic. The amount is based on the volume of queries which is one indicator of malicious usage. This change makes the full algorithm non-deterministic and actually improves the quality of results. Furthermore it makes the malicious user less certain of the results they are getting.

Balancing privacy against system effectiveness is a challenge presented to all collaborative services. As these systems become more common they will involve more people and use more private data. Thus an ever-growing class of applications faces these challenges.

## 4 System Design

The scheduler relies on a trusted third party with full access to all the data and a model that captures the user's intent. To this end, we have designed a web-based service coupled with a database.

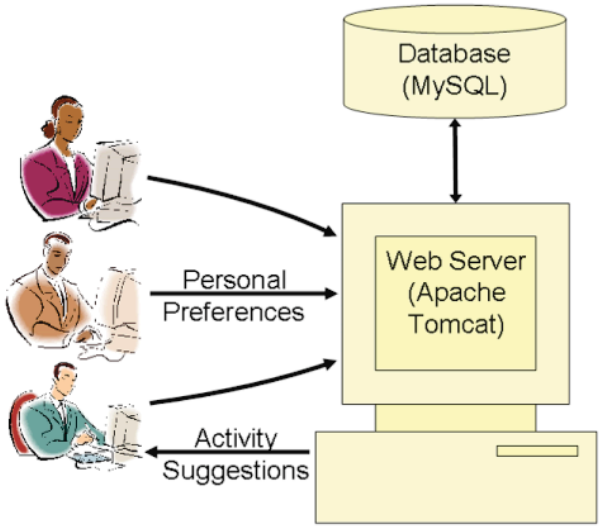


Figure 1: System Overview

### 4.1 Trusted web server

This system has a web interface, which was developed using the Apache Tomcat web server[3]. Tomcat supports Java servlets and SSL connections. This allows dynamic content to be sent securely to the user.

Users can connect to the server through Secure Hypertext Transport Protocol (https), which provides an encrypted communications channel. When a user connects to the web server, they must authenticate themselves with a password which is transmitted over this secure connection.

### 4.2 Servlets

The servlets running on the server contain all the system logic. This includes the entire matching algorithm and user interaction.

When a user clicks on any link, they trigger an event which causes Tomcat to run a servlet. Each servlet first checks the user's credentials. This is done by calling an authenticator which searches for a cookie. If this cookie is not found or does not match the record in the database then the user is shown a login prompt. Otherwise the servlet is allowed to continue.

Once the user has been authenticated, the servlet is allowed to continue its function. Generally this will involve making a database query to fetch some data and generate a web page. The matching servlet is the only one that significantly differs. It has the added role of randomizing the data before running the optimization algorithm.

### 4.3 Central database

A MySQL[2] database runs on the same host as the web server. This database provides the backend for the website and is not accessible remotely.

This database stores the information for each user and activity. For each user we must store their login information as well their preference for potentially every user and activity. We must also store some information for each activity, name the number of participants. For instance an activity like reading a book involves just one person, whereas playing volleyball can involve anywhere between 4 and 12 people.

Between the core logic and the database lies a set of wrappers. This abstraction is a piece of Java code that communicates with the database using standard UNIX interprocess communication.

This important software layer also provides security protections. It helps the matching servlet apply timestamps and randomization to perturb the inputs to the matching heuristic.

### 4.4 Email notification

After a user has received a suggestion, she can choose to accept or reject it. When a match is accepted, the scheduler will notify all the group members of the proposed activity. Currently the system is prepared to send an email to each person in the group. Since the system is still being tested, this feature is disabled.

## 5 Algorithm Design

The trusted third party attempts to model the user's intent. We have developed a heuristic which captures several key features of the subjective "goodness" of a proposed matching. Since we have access to the entire database of preferences, we can use this to evaluate every potential matching.

In this way, every suggestion can be given a numerical value. The matching algorithm simply considers all possible matchings and finds the global optimum. The user is then given only the highest ranking result.

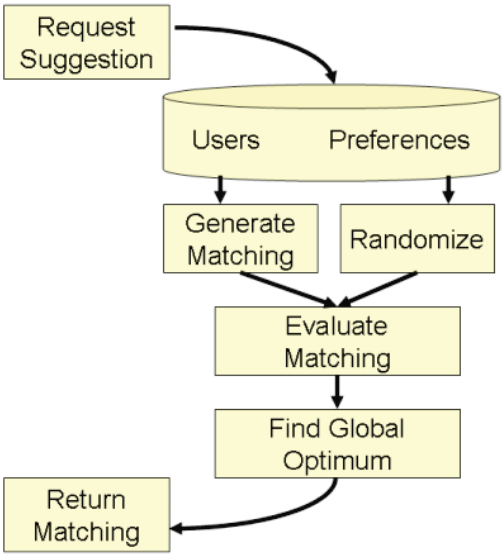


Figure 2: Algorithm Overview

### 5.1 Editing Ratings

Each rating for a person or activity is stored as a value between 0 and 1, unknown values are treated as 0.5. When a user updates their preference, this value is immediately stored in the database. It should be noted that the user sees a scale of 1-10, which is

translated into an internal value. Since the final value of a matching is a weighted average of these ratings, it too is assigned a value between 0 and 1.

## 5.2 Matching heuristic

The following factors affect how "good" a suggestion is:

- User's rating for the activity
- User's rating for the partner(s)
- Other partner(s) rating(s) for the activity
- Other partner(s) rating for each person in the group

These factors are not of equal importance, in fact different types of people value each in different ways. The user issuing the query is very concerned with the first two, however the last two are more important for a globally optimal solution.

The final heuristic is the weighted average of these factors. Each of the four factors is assigned a weight based on its importance. Lacking data to support any particular weight distribution, we simply gave each factor an equal weight.

The last two factors may actually be averages themselves. If a matching contains more than one other person we treat each of them equally. Thus the second two factors are the average rating for the activity or partners.

Giving every other person an equal weight is not the only option. Other potential schemes include weighting their preferences based on the user's rating for them. This makes the algorithm more complex and uses certain ratings multiple times. For these reasons we did not pursue such a weighting method.

This heuristic has two potential problems. Firstly the user has direct control over half the inputs to the function. Secondly it is entirely deterministic. Both these factors could be exploited by a malicious user because they give her precise control of part of heuristic. This could lead to serious problems because the heuristic is the only thing that the matching algorithm considers.

## 5.3 Perturbing results

Every input is blended with a random value. The amount of noise is called the user's penalty score. It is based on the volume and frequency of activity. We use the blending function:

$$v\prime = v(1-p) + rp$$

where $v$ is the true value, $v\prime$ is the observed value, $p$ is the penalty score, and $r$ is a random number. This function is applied to every rating considered in an evaluation. Notice that in the case $p = 1$ the observed value is completely random, so the heuristic value of a matching would also be completely random.

Every time a query is made, the user's penalty is updated. It is first discounted with an exponential decay. This is based on the amount of time since the last query was issued. Then the penalty is increased by a small additive constant. We use the penalty update equation:

$$p\prime \leftarrow c + p2^{-t}$$

where $p$ is the original penalty, $p\prime$ is the updated penalty, $c$ is the additive constant, and $t$ is the time since the last query.

Typical values for $p$ and $c$ are 0.15 and 0.05 respectively. With a penalty score of 0.15, a preference of 1 could actually be observed as small as 0.85. Similarly a rating of 0.5 would be transformed into the range $0.425 - 0.575$.

The goal is to ensure that a malicious user would merely see the results of optimizing almost completely random noise. Note that a flood of queries will raise this penalty quickly. The small increments will add up and the frequency of queries prevents substantial decreases.

As the penalty approaches 1, the matching algorithm will use values that are increasingly random. The optimization algorithm is still deterministic, but it's inputs become less reliable. Consequently a malicious user will have little confidence in the responses.

These equations can easily be tuned to be more aggressive towards potential attackers. By raising the

constant, the system will be quicker to add noise. Similarly one could scale the time units so that the penalty takes longer to dissipate. This would prolong an attack because because the attacker would need to wait longer between queries, allowing more time for the attack to be detected or peoples preferences to change.

# 6 Attacks

By using a central server, this system is susceptible to traditional information warfare attacks such as denial of service attacks and software vulnerability exploits. We do not address these concerns because they have been studied in detail and exploit weaknesses beyond our control. Instead we focus on malicious users trying to deduce personal preferences based on the suggestions from many queries.

## 6.1 Types of Attacks

Anyone can simply listen to the reported suggestions and gain information. However in this sort of passive attack, the users and activities involved can not be controlled. If a malicious user were able to defeat the SSL encryption, they could eavesdrop on many users' communications and quickly gain a lot of data. The strength of this form attack is that it can be done covertly.

In contrast, a targeted attack focuses on a particular preference that a certain user has set. We present one such attack later in this section which involves making many requests with carefully crafted preferences. This type of attack is the most dangerous because it allows an attacker to get exactly the information they want.

An attack may only be partially targeted if the attacker does not (or is not able to) completely constrain the portion of the database used. For example a malicious user could focus on a particular user while not targeting any particular preference they have set.

## 6.2 Similarities Among Attacks

All attacks rely on the results from many queries, which are usually generated by the attacker. To determine an accurate value it is helpful to compare with a known value, such as the heuristic's evaluation of a one-person activity. Finally it is helpful to fix as many variables in the heuristic as possible. This can be accomplished by setting preferences to the maximum and minimum ratings

## 6.3 Two-person activity

This attack allows a malicious user to determine any user's preference for an activity involving two people. Although limited to certain activities, this algorithm demonstrates that a malicious can accurately reproduce at least a portion of the database.

The attack consists of these five steps:

1. Create a new account (so the target's rating for the malicious user will be 0.5)
2. Set all ratings to 0
3. Set rating for target user and activity to 1
4. Raise rating for reading a book and make a query
5. Repeat the previous step until the one-person activity is suggested

Let $X$ be the rating being targeted and $Y$ the malicious user's rating for reading when the suggestion switches. Of the four factors that influence the evaluation function for the target activity, only $X$ is unknown.

| | |
|---|---|
| User's rating for the activity | $= 1$ |
| User's rating for the partner | $= 1$ |
| Partner's rating for the activity | $= X$ |
| Partner's rating for the user | $= 0.5$ |

At this point we can evaluate the heuristic to $(2.5 + X)/4$ for the target activity and $Y$ for the one-person activity. Steps 4 and 5 ensure that these two are nearly equal so $(2.5 + X)/4 = Y$ or

$$X = 4Y - 2.5$$

With more queries we could determine a more accu-

rate value for $Y$. In fact it is possible to do a binary search, so every query doubles the precision.

This method could fail if $X = 0$, however the attack will still succeed. In this case the heuristic evaluates to 0.625. This is the maximum that can be attained for another user or activity, so a third activity or user might be suggested instead of the target activity or the one person activity. However if this occurs we know immediately know that $X$ must be 0, so we have still determined the user's preference for the target activity.

We note that there is nothing special about the book reading activity. The malicious user simply needs a known reference point to compare the unknown with. Any 1-person activity would work. Similarly if the malicious user had two accounts she could accurately determine the value for a 2-person activity.

Using the full algorithm and adding noise limits this attack. In particular the malicious user can no longer determine exactly where the cross-over occurs. Thus they can not set the two quantities equal and must introduce an uncertainty. Now the attack produces:

$$X = 4Y - 2.5 \pm e$$

where $e$ is the maximum error attained. This confidence interval is based on the parameters of the penalty update function as well as the pattern of requests used.

# 7 Evaluation

This scheduler was tested in small controlled settings. While operating the system we considered these three subjective questions:

## 7.1 How good is the heuristic?

In controlled tests the heuristic produces an activity and grouping which reflects the entered preferences. Upon initial inspection it seems biased towards smaller groups and favors the user that made the request. This suggests that the four parts of the heuristic should not be given equal weight. However the end-user is the ultimate judge of what is reasonable, so a larger study with human subjects is required to gauge this accurately.

## 7.2 What is the effect of added noise?

In controlled tests, a small amount of noise seems to improve the results. With an entirely deterministic algorithm, the user must edit their preferences before they can get a different suggestion. However by making several requests the user will accumulate a small penalty.

This will introduce a little randomness which has the to potential to raise or lower the value of any matching. So other highly ranked suggestions could be boosted above the "optimal" one allowing the user to see these alternative choices. Since the heuristic can only guess at what the user wanted, revealing more options makes the system appear more intelligent.

## 7.3 Does entropy stop intruders?

Adding noise based on the volume of queries appears very successful in preventing a single malicious user from reconstructing a significant portion of the database.

There are two major limitations to this approach. Firstly it does not stop a group of malicious users. The system can not detect collusion among several users. Furthermore each of these users would need to issue far fewer queries so it is not sufficient to merely examine the usage patterns of a single user.

The other drawback is that it has limited success defending against a focused attack. An attack like the one described above needs relatively few queries. This defense is reactive in the sense that it becomes more and more effective as the abuse becomes more pronounced. If the attacker is only interested in a certain key piece of information, it may be determined with high accuracy before the penalties begin to set in.

Nonetheless, perturbing input data seems to provide a simple and effective deterrent. In order to produce a better estimate of a particular piece of data, an

attacker must issue more queries. This in turn raises the penalty score which introduces more randomness. As a result, the attacker has less confidence in the results.

# 8 Future Work

This system only scratches the surface of what scheduling services can do. We have sketched many interesting ways to improve this service below. Many of these ideas apply to collaborative systems in general.

Our search technique guarantees optimality of the heuristic by performing an exhaustive search. This is computationally intensive and does not necessarily reflect the quality from the users perspective. The running time is bounded by $O(AU^p)$, where $A$ is the number of activities, $U$ the number of users, and $p$ the maximum number of participants in an activity. We expect the number of activities to be relatively small, but $O(U^p)$ is still too great to scale to a large user base.

A better way to do this is with satisficing behavior, in other words to generate a good matching which is close to optimal. This technique could be a lot faster and may be indistinguishable for the end user.

While adding random noise to the data thwarts a lone attacker, there are other defenses that should be considered. For instance we may seek to identify collusion among users and penalize both. Alternatively it might be possible to get better results by injecting non-uniform noise. Finally a hybrid defense involving an intrusion detection system scanning preference settings may be able to detect attacks with greater confidence.

There is an overwhelming number of options, the system could be trained to learn preferences by examining which queries are accepted. Completely automating this process is made difficult by the relatively small number of queries issued.

The scheduling system could be expanded in several ways. One useful extension would be to provide calendar functionality allowing the system to schedule times and reserve spaces. Adding a notion of skill levels or otherwise associating users with activities would generalize the system further and make it more applicable.

One of the biggest assumptions we made was the existence of a trusted third party. However the users may not trust anyone, or are partially disconnected. One radical departure would be to design a similar system that did not rely on a centralized server. While a user could guarantee privacy by limiting what data is sent to peers, we would expect the quality of results to suffer.

# 9 Conclusions

Relying on a trusted third party allows the system to access all the data. While this makes the service susceptible to traditional information warfare techniques (e.g. cracking the host, masquerading, etc), the potential benefits outweigh these risks. Perhaps the greatest benefit is that one can guarantee a globally optimal solution.

Attacking these systems to access the private data is non-trivial but it can be accomplished. Each operation reveals some small amount of information. By carefully examining these traces, it is possible to reconstruct a portion of the database. This reproduction is imperfect and subject to accuracy limitations.

Privacy and system effectiveness are two conflicting goals. We could improve privacy by being more aggressive about adding noise. While attackers would have less confidence in the values they obtain, normal users would be adversely affected. Balancing these two depends on the sensitivity of the data and it is design choice to be made in numerous scenarios.

## 10 Acknowledgments

I want to thank my advisor, Peter Steenkiste, for his support and encouragement. I also want to thank Urs Hengartner for logistical support in getting started. Finally I want to thank Latanya Sweeney for giving me a forum to discuss my privacy concerns.

## References

[1] NannyMUD, a social game, went online in 1990
www.lysator.liu.se/nanny/

[2] MySQL is an open source database
www.mysql.com

[3] Apache Tomcat is an open source web server
jakarta.apache.org/tomcat/index.html

[4] Blizzard offers a player matching system in their Warcraft 3 game.
www.blizzard.com/war3

[5] eHarmony offers uses a matching heuristic in their dating service.
www.eharmony.com