Structuring a Contact Sharing Database - Mac Finnie

In the fall and into Winter Study I was able to work on developing a system for using a smartwatch to share contacts. My project proposal is to continue this work. Firstly, I need to assess the state of the project - remember where we left off. Then, I see the next real systems problem to be one of scalability. The backend server currently consists of a PHP script receiving data from phones, storing that data in a SQL database and then running a matching algorithm on the data. As the server receives more requests, the number of accesses to the database will also rise, leading to problems in blocking and synchronization. I want to ensure that the matching algorithm always has access to the relevant data when pairing for a user, and I believe that the key to this is the structure of the SQL database and the PHP scripts accessing the database. Finally, I intend to fully organize the progress I have made so far, so that work can more easily continue in the future.

Contact sharing happens by pairing two users based on the data received from their pebble smartwatch. At this point, that data is the time at which the user pressed a button on the watch. The button press is intended to represent the recognition of a contact-sharing gesture, and could thus be replaced with a detection of a handshake, fistbump, etc. The pairing occurs by passing on the data to the Android Phone which the watch is connected to via bluetooth. Then, the phone makes an HTTP GET request to a PHP script on the back-end server, Sysnet0, providing the time of the button-press and any other relevant details as parameters. The PHP script enters the data into a SQL table, and then runs a matching algorithm on the data in the table, seeking to pair the user and button-press with another user and button-press.

The system as of today is composed of three main machines,

1. Back-end server (sysnet0)
2. Android Smartphone
3. Pebble Smartwatch

And some other pieces,

1. Database (SQL)
2. Server-Phone Web Interface (Apache & PHP)
3. Android App (Java)
4. Smartwatch App (C)

There is a problem with scaling this system. As more users come online and begin to send data and request pairings from the server, there will be a downtrend in performance *and* reliability. The number of accesses to the database table is equal to the number of users entering data at that time. I am not sure if this is true, but in the worst case, write-access to a SQL table is blocking, and thus creates a backup. This is analogous to a decline in both harvest *and* yield; complete data is not being used in the matching algorithm *and thus* the number of queries successfully completed is declining.

Even if write-access is synchronized, we have another problem - what data is the matching algorithm supposed to run on? In our case we have a single table upon which every matching algorithm is run.

But, this is inefficient - we have data to indicate what entries are relevant to a given user's request for a pairing; namely, GPS. A contact sharing event occurs in a given *locale*. I believe structuring the data in the SQL database to match this is important. Then, our algorithm can run on smaller datasets and thus be faster.

Ideally, the structuring of the database would be automatic - coded into the database or some external program controlling the organization of data. But, at this point I think the project would consist of creating a uniform representation of a pairing requests data in the form of a JSON object or some other key value mapping. Then, work would need to be done in PHP scripts to route a given data entry and request to its corresponding table in the SQL Database. A lot of this reminds me of redirection, content distribution networks, and how connecting to a site like CNN means connecting to the nearest hosting server.