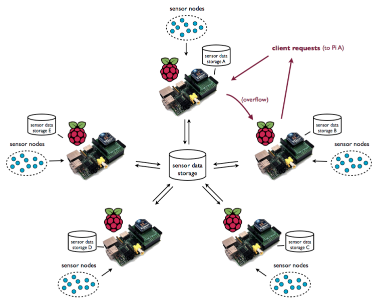
FRAICHE

Fast Response and Intelligently Controlled Harvest Environment



The system architecture: each Pi is located in a garden where sensors communicate soil readings to it. Each Pi keeps contact with a central data repository, which it uses to fill overflow requests from clients.

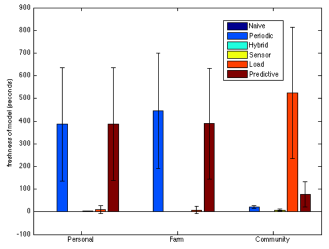
Gardeners have problems, too

Managing a community garden or small farm is currently a process that requires community members or the farmer to directly monitor their land in an ad-hoc manner. Additionally, beginner gardeners who lack a large base of knowledge sometimes find it difficult to react effectively to the signals they see coming from their plants, watering them to the point of drowning or failing to notice when they are parched. Computing has been leveraged to improve the ease of interaction with and analysis of many other physical processes, and here it can play an effective role in monitoring and predicting water needs in the challenging gardening environment.

FRAICHE is a complete small-scale system which includes an in-garden device that measures moisture and a Raspberry Pi elsewhere that acts as a webserver and a machine learning platform to predict moisture changes.

Designing for the Raspberry Pi

The Raspberry Pi, the widely-hailed cheap and easy to use computer-on-a-chip, has several important limitations, the most important of which is CPU. We considered two approaches to this issue: intelligent scheduling, and parallelisation. Both are worth considering as the first allows us to support the minimal-setup, single device, case, while the second shows the potential for scaling as needs grow.



The freshness of predictions served by the scheduling algorithms.

Six Scheduling Algorithms

Naive method - sensor readings are cached until the server is queried by a gardener. Then the prediction model is updated, ensuring maximum freshness.

Periodic offline computation - model updates occur according to a schedule, reducing the delay when a query comes in, at the cost of freshness.

Sensor-triggered updates - when a fixed number of sensor updates have arrived the system updates the model, giving a freshness loss bounded by collected information rather than a fixed time.

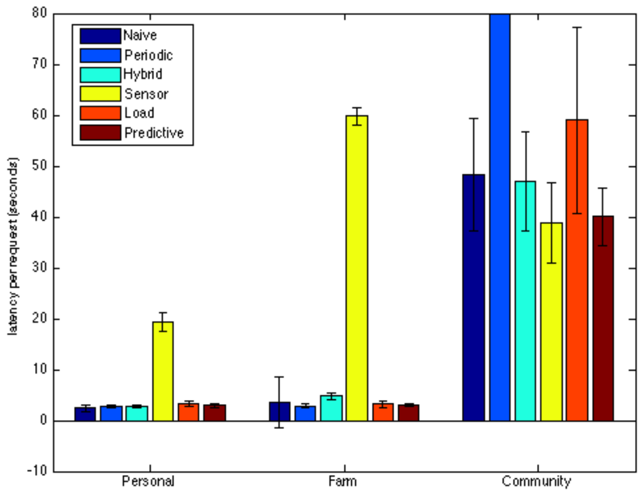
Hybrid naive/periodic timing - the model is updated periodically and naively when a query comes in, ensuring freshness at a lower cost than the naive approach.

Updates when load is low - updates occur when few clients are connected to and querying sensor readings.

Updates prior to predicted requests - model query traffic and update the model shortly before the predicted query time.

Multiple Pis

We added an additional Raspberry Pi and facilities for the two to communicate and load balance using Amazon Cloud Services as an intermediary data storage location. This system is scalable and self-balancing.



The latency for clients served by each scheduling algorithm in each scenario.

Evaluation

We implemented all system components, including remote sensors, the server on the raspberry pi, and the Amazon based multiple pi setup. To enable repeatable experiments we created mock sensors based on a simple world simulation, and mock clients that ran firefox instances and periodically made queries.

We used this setup to test three scenarios: community (100 plants, 32 clients), farm (100 plants, 1 client), and personal (5 plants, 1 client). We ran semi-patterned traffic for each of our six scheduling algorithms.

Peggy Chi
Jonathan Kummerfeld
Valkyrie Savage

CS262A Final Project