ASPC Root Project

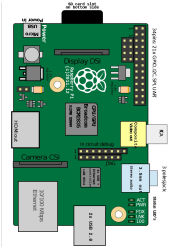
David Cowles, Logan Crowe, Ayden Martin, Trevor Hamilton

ASPC Overview

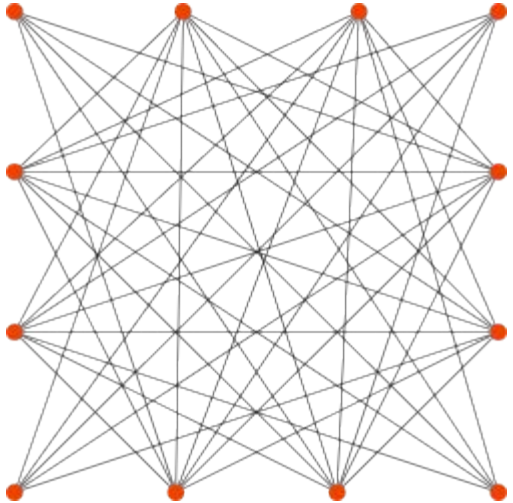
Automatic Spring Products Company (ASPC) has multiple locations with large floors full of spring production workcenters. Each workcenter is manually assigned appropriate jobs by floor workers, who are responsible for setting up the machines and interfacing with Plex. Plex is a cloud based enterprise resource planning software package used by aspc to keep track of transactions, manufacturing and supply chain details, employees, etc.. Manufacturing jobs are distributed to workstations by higher-ups in the company, however the floor workers have autonomy in deciding which jobs are in progress in a given day. Additionally, they are responsible for recording the state of the workstations throughout the production process using a kiosk to interface with Plex.

There are several problems with this setup. Firstly, employees are constantly moving between the workstations and the Plex kiosk to ensure that the cloud is updated to reflect the status of the workstation. Secondly, some material is used during machine setup that is not currently being recorded and sent to the cloud. This results in inaccurate records of available raw materials, causing unexpected shortages and production delays.

Project Overview



In order to solve the problems outlined earlier the client requested that two GVSU teams collaborate to setup a network of sensors in order to automate the integration between the workstations and the cloud. This achieves duel goals of increasing worker productivity and ensuring correct raw materials records.



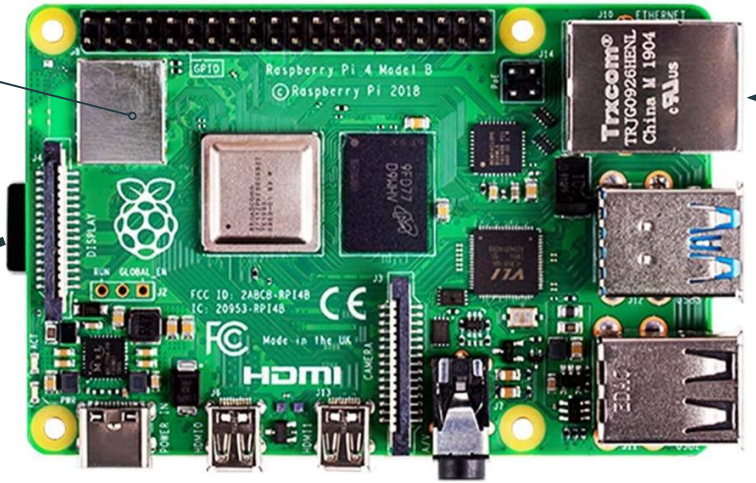
Our group was assigned to develop a root server for the purpose of coordinating and communicating with the sensors as well as sending and receiving data from the cloud. We act as a hub for the sensors and an intermediary between the workcenters and cloud.

This portion of the project was largely exploratory due to the clients lack of information about the resources we would be working with and uncertainty about the project goals. In addition to code, we were asked to deliver a detailed report documenting everything we learned throughout the project for future use by the company.

Root hardware

wlan0 - Wifi AP

Static IP
10.20.1.1



eth0 -
LAN/Internet

Dynamic IP

- Operating System
- Server
- Database
- Router

Wifi Mesh Network Access Point

Requirements

Setup and maintain

- DHCP service
- DNS service
- Routing
- IP masquerading

Must do all of the above, as well as convert a wifi receiver into a wireless access point and create it in such a way that little to no human interaction is required after creation.

Progress & Solutions

All main goals have been met with the exception of full automation. Currently, everything runs and maintains itself after initial setup but setup process has not been automated for replication or restoration

1. DHCP and DNS services
    a. Defined the wireless interface configuration in /etc/dhcpcd.conf
    b. Used dnsmasq to configure DHCP to deliver IPs based on specifications we provide
2. Routing and IP masquerading
    a. Configured the file /etc/sysctl/routed-ap.conf
    b. Configured firewall to use IP masquerading
    c. Utilized netfilter-persistent and its plugin iptables-persistent for maintenance of network on reboot
3. Wireless Access Point
    a. Created hostapd configuration file with wireless network specifications

Branch & Leaf Interaction Requirements

- A server that will always be listening for connections
  - The server will need to be capable of having multiple connections going on at a single time so server can communicate with leafs, there is a possibility that more than a hundred connections may be needed at once.
- Able to add new Leafs to a database and correlate them with their proper workstation
  - When a leaf is initially added to a new workstation root will have to be able to add new leaf into the database so that leaf and ip address can be referenced later on. Required due to leafs needing to store little to no static data on them of workstations.
- Retrieve job information from PLEX and database, send to Leaf when requested
  - When a workstation operator begins a new job Branch & Leaf will need to retrieve the job information from root
- Detect if a Leaf in the Database is still online
  - Pings all leaves on a set time interval to detect if the Leaf is still online and functional. Useful for finding and replacing bad machines.
- Reboot server if crashes
  - Incase the Root Server crashes, need a method to easily and quickly restart it so the B&L team can always expect to get a connection with Root, and Root is always able to answer data requests from a Leaf.

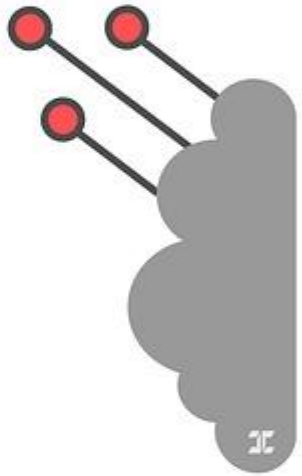
Problems & Solutions

A lot of problems that cropped up came with more understanding of the systems and with objectives being replaced.

- Leafs won’t remember what workstation they are currently working with.
  - The B&L team was told they shouldn’t store workstation id, so that means we would have to find some way to correlate the Leaf job request with workstation to give a proper response. Solved this by creating a DHCP server to assign static ip addresses and then using IP addresses as a key to get workstation information.
  - Had to create a setup method for when a new Leaf is assigned to a workstation
- Data needing to be sent and received changing.
  - With problems coming in from other spots such as PLEX, the data we needed to send and receive to Leaf had to constantly undergo change. This involved us updating our json format quite frequently to include information such as wheel diameter.

PLEX Interaction Requirements

- Retrieve job data to be sent to sensors
  - Job Id - Id assigned by plex as a key for job data
  - Part Id - Id assigned by plex to differentiate between different part types.
  - Part_Type - Explicit description of product to be produced at a workstation as
  - Part_Capacity - Number of parts to be produced in order to complete a specified joB
- Send production information to cloud to automate updates
  - Scrap Count - The length of material consumed in workcenter setup
  - Workcenter Status - [idle, setup, production], description of what the workstation is currently doing

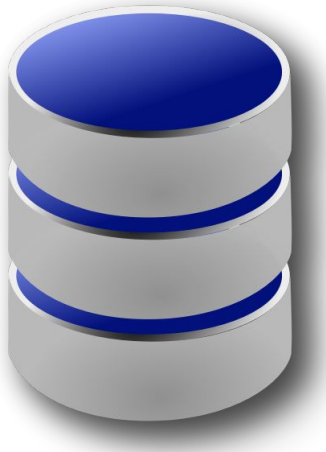


Progress & Solutions

We ran into several significant issues in attempting to integrate our system with the API

- We learned that Plex can be very slow in responding to requests for API credentials, only receiving access towards the end of the semester
    - Once receiving access, we were able to use several API calls in order to get the data necessary to send to leaf nodes, however we also learned that scrap and status updates were not possible through that API
- We then learned about a second API that might allow update calls, for which we had to gain new credentials
    - This API was entirely undocumented, and we had to get whitelisted for the specific resources we thought we might need. However without documentation it was difficult to figure out what the different data sources actually did
- At the end of the day we decided to integrate our system with both API's, however due to slowness in gaining access to the new resources it will be the last piece of our project.
- Our work learning about the API's will be useful for future projects at ASPC, which our client will be in charge of.

Database Requirements



- Database should provide a local backup for important data to be sent to the cloud server and the leaf nodes
  - Jobs - id, part id, amount completed, total amount needed, scrap parts
  - Workcenters - id, current job id, feed wheel diameter, sensor data, current state
  - Parts - id, length, number, type
- Database should be flexible and scalable
  - Table structure can be changed with minimal effort
  - New tables can be added with minimal effort
- Database should provide an easy to understand interface for other scripts to perform operations on the database
- Database should protect against errors and malicious injection attacks

Progress & Solutions

- Python's SQLite module was chosen for the means of creating a database on a file
  - A DataBase class was also created to hold the file path and initialization status
  - A one-time method is called after creating a new DataBase instance in order to create all tables
  - The constructor will recognize if an already initialized database file is passed in
- A combination of keyword parameters and formatting are used to dynamically construct commands
  - Since the user specifies the table name and column names, this allows most methods to be reused for any table
  - Users can exclude columns that are not relevant to the operation
- Dictionaries containing all column names and table names are stored as class variables
  - These are used to verify user-submitted column names and table names before any statements are executed
  - The table dictionary provides a shortcut for users by converting an index number into the corresponding name
- If an error occurs, any changes made to the database are rolled back and the error is returned
- DB-API's parameter substitution is used to prevent SQL injection
- One problem that occurred was SQLite's lack of a native boolean type
  - Our solution was to use an integer type and constrain values to 0 or 1
  - Code was added to translate these values to True or False values

Future Work

Our group has completed most of the cursory research and testing for the technologies our client wants to implement, and we have the basic functionality established. However the client has a larger vision for the project and now has the necessary information to proceed with the development of more ambitious features.

- Develop method for determining when workcenter blades are dull
- Allow leaves to be configured remotely instead of via the touchscreen.
- Potentially fleshing and implementing out our automated report system
- Error checking and handling to alert operators to workstation malfunctions
- Properly handle cloud interactions in the case of wifi outages

Future Solutions

Future work initiated by the client will be built on top of our existing project, which includes some functionality that was partially implemented while we waited for API access during the first part of our project. Our final feasibility report and project documentation should help the client avoid major pitfalls during this extension.

- The database can be extended to aggregate sonic data collected from the workcenters, an ML algorithm can be developed to “listen” to the machines and alert workers when a blade needs to be replaced
- The server can securely serve a simple web application on it’s own private network which would allow remote sensor configuration
- The existing automated report system is set up to automatically send email updates to a list of users. If the client wishes to implement this functionality an html template with the desired graphs can be created and added to the project.
- Once the client has access to the new API research will have to be done to understand the potential error scenarios, experience with the workcenters and their errors will also be necessary.
- Given information about the average duration of wifi outages, data will need to be temporarily collected until the API can be accessed again.

Concluding Statements...

Thanks!