Stream Processing On IoT Devices Using Calvin Framework

Student: Ameya Nayak | akn6749@rit.edu
Advisor: Dr. Peizhao Hu | ph@cs.rit.edu

Introduction and Motivation
- Since IoT devices have low memory and disk space, the data collected from these devices is pushed to the cloud for data processing and analysis.
- Some drawbacks of performing these tasks are as follows:
  - If the internet is down, data cannot be pushed to the cloud.
  - Storing the data and performing data analysis on the cloud is costly.
- Using existing big data processing frameworks like Apache Spark would be an overkill since it requires a lot of memory and disk space.
- We propose a lightweight framework using Calvin for data processing and analysis on the IoT devices which can run on such devices with low memory and disk space.
- This framework would help automate data flow and serve as a pipeline for parallel stream data processing.

Calvin - Overview
- Calvin was developed by Ericsson Research and open sourced in 2015.
- Calvin combines ideas from Actor and Flow Based Programming models to provide a framework to build distributed applications.
- Benefits of using Calvin are as follows:
  - Requires low memory and disk space.
  - Supports Python.
  - Users can leverage inbuilt methods for data migration from one device to another in case of node failures.
  - Users can specify backup nodes in case the current node fails.
  - Allows users to distribute workload among different nodes.
- Developing a Calvin Application involves 4 phases:
  - Describe: Implementing individual components of the application.
  - Connect: These components interact with each other in the form of a graph.
  - Deploy: Instantiating the application as a graph.
  - Manage: Automating allocation of resources to the components during the lifecycle of the application.

Calvin - Building Blocks
- Calvin Actors:
  - Actors are python classes where the actual code for data processing and analysis resides.
- Calvin Runtime:
  - A Calvin runtime is an execution environment API required to run calvin actors.
  - Each runtime runs on a specific port.
  - Multiple calvin actors can run in a single runtime.
- Calvin Script:
  - A Calvin script is used to deploy the application as a graph.
  - The calvin script is a .calvin file which allows the users to specify individual components of the application as nodes and also specify the relationship between the nodes as edges.

Calvin - Overview
- Calvin was developed by Ericsson Research and open sourced in 2015.
- Calvin combines ideas from Actor and Flow Based Programming models to provide a framework to build distributed applications.
- Benefits of using Calvin are as follows:
  - Requires low memory and disk space.
  - Supports Python.
  - Users can leverage inbuilt methods for data migration from one device to another in case of node failures.
  - Users can specify backup nodes in case the current node fails.
  - Allows users to distribute workload among different nodes.
- Developing a Calvin Application involves 4 phases:
  - Describe: Implementing individual components of the application.
  - Connect: These components interact with each other in the form of a graph.
  - Deploy: Instantiating the application as a graph.
  - Manage: Automating allocation of resources to the components during the lifecycle of the application.

Calvin - Building Blocks
- Calvin Actors:
  - Actors are python classes where the actual code for data processing and analysis resides.
- Calvin Runtime:
  - A Calvin runtime is an execution environment API required to run calvin actors.
  - Each runtime runs on a specific port.
  - Multiple calvin actors can run in a single runtime.
- Calvin Script:
  - A Calvin script is used to deploy the application as a graph.
  - The calvin script is a .calvin file which allows the users to specify individual components of the application as nodes and also specify the relationship between the nodes as edges.

Data Flow
- As described in the architecture we have built an application for predicting whether the centralized AC/Heater should be turned ‘ON’ or ‘OFF’ based on the room temperatures from 3 rooms.

Results
- We deploy this application on a cluster of Odroid devices upto 5 devices, to simulate the working of this application on a cluster of IoT devices.

<table>
<thead>
<tr>
<th>Number of nodes in the cluster</th>
<th>Memory utilization for Naive Bayesian implemented on a Single Actor (in MB)</th>
<th>Memory utilization for Naive Bayesian implemented using Multiple Actors (in MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Node</td>
<td>97 -105</td>
<td>95 -100</td>
</tr>
<tr>
<td>5 Nodes</td>
<td>95 - 101</td>
<td>93 - 97</td>
</tr>
</tbody>
</table>

- Running the framework (and the machine learning algorithm) using multiple nodes distributes the computation load on each node thereby reducing memory utilization per node as compared to running the framework on single node.

<table>
<thead>
<tr>
<th>Number of nodes in the cluster</th>
<th>Execution time for Naive Bayesian implemented using Single Actor (in seconds)</th>
<th>Execution time for Naive Bayesian implemented using Multiple Actors (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Node</td>
<td>Training: 65-70 Prediction: 0-2</td>
<td>Training: 70 - 80 Prediction: 0 - 3</td>
</tr>
<tr>
<td>5 Nodes</td>
<td>Training: 70-75 Prediction: 0-3</td>
<td>Training: 80-90 Prediction: 0.4</td>
</tr>
</tbody>
</table>

- Running the framework (and the machine learning algorithm) using multiple nodes as compared to running the framework on single node involves an additional message passing overhead from one actor to another thereby increasing the time required to run the application.

Drawbacks & Challenges
- Does not support programming languages other than python.
- Since all the Calvin runtimes need to be configured before deployment, achieving auto-scalability is difficult.
- Debugging the code in a Calvin Actor is difficult since the error messages are most of the time vague and not at all helpful.

Future Work
- Implement more generic actors which can be used as in-built actors for implementing various applications.
- Dividing other machine learning algorithms into subcomponents and implementing these subcomponents on individual actors, thus allowing users to implement such algorithms on single or multiple runtimes.

Figure 1. Architecture of the Data Processing Framework
- Pre-processing data and machine learning algorithm may be implemented as a single or multiple actors.
- Once the actor pushes the output data it is ready for processing the next input token even though the data has not been processed the entire framework.
- To achieve distributed functionality, the mapping of calvin actors to calvin runtimes has to be specified in a .deployjson file.
- Each runtime runs on a different device in the cluster.

Figure 2. Naive Bayes Classification Implemented in Single Runtime
- Training: 80-90
- Prediction: 0-4

Figure 3. Naive Bayes Classification Implemented in Multiple Runtimes
- Training: 80-90
- Prediction: 0-4