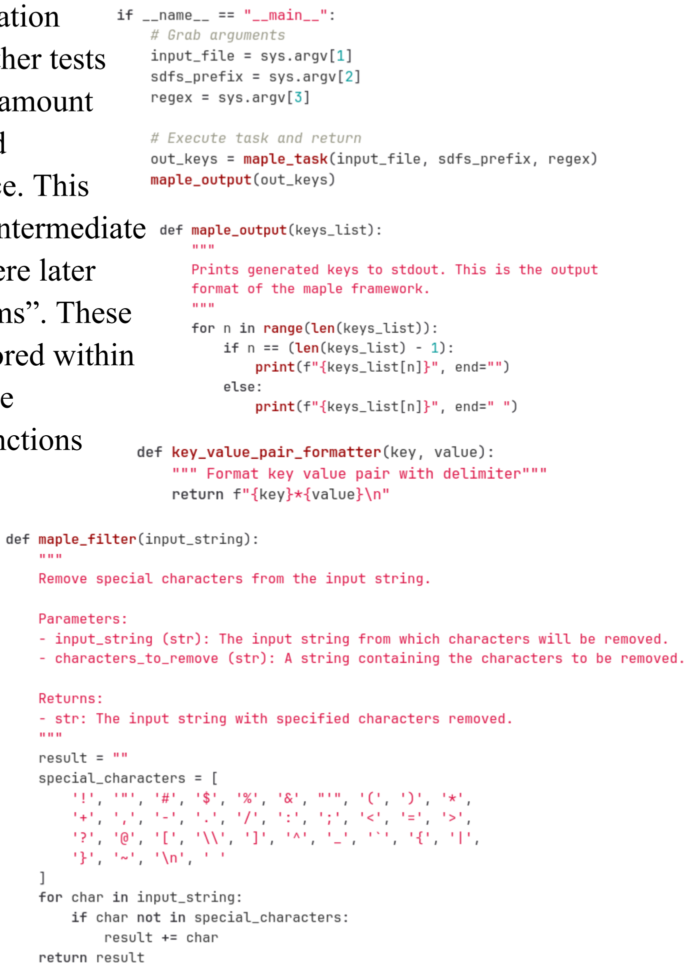
MP4: MapleJuice + SQL

Design

A MapleJuice job consists of a Maple and Juice stage. Each node in the cluster submits jobs, which are queued and handled by a centralized leader. The leader supports a ResourceManager (RM) that schedules the respective Maple and Juice stages. It partitions input data files, handles failure re-execution, and collects outputs from nodes in the cluster. A thread barrier is placed between Maple and Juice execution stages. The implementation is broken into three classes: MapleJuice, MapleEngine, and JuiceEngine. The MapleJuice class encapsulates the RM, which interfaces with MapleEngine and JuiceEngine to handle task partitioning and scheduling. It uses SDFS to store and replicate output and intermediate files from each job. This proved to be the major bottleneck in our design, as SDFS file replication constituted the majority of job execution time. Further tests found the bottleneck to be from replicating a large amount of small key files. To improve our design, we could implement a Combiner Phase similar to MapReduce. This would reduce the number of key files by merging intermediate results after the Map/Maple Phase. SQL queries were later layered on MapleJuice in the form of “user programs”. These are separate Python files inputted by clients and stored within the SDFS for execution. Maple and Juice phases are programmed separately, but have similar helper functions for correct key, value pair output generation.

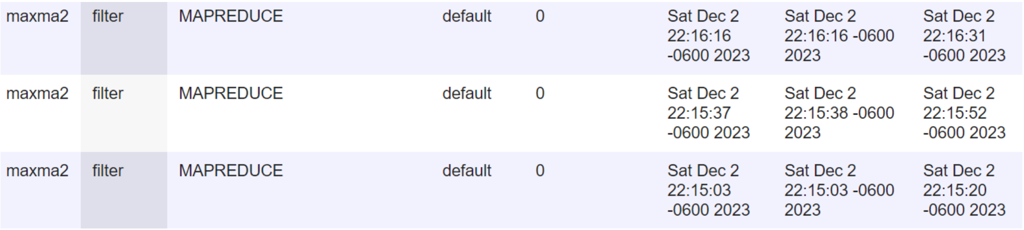
MapleJuice vs. Hadoop/MapReduce

Due to the SDFS bottleneck, Hadoop was more performant on the majority of tests. MapleJuice measurements were taken from output logs with execution time for each job. Hadoop’s came from the application user interface shown below. Datasets were taken from the Champaign GIS Database and are indicated by the source number for each test they were used in.



```
if __name__ == "__main__":
    # Grab arguments
    input_file = sys.argv[1]
    sdfs_prefix = sys.argv[2]
    regex = sys.argv[3]

    # Execute task and return
    out_keys = maple_task(input_file, sdfs_prefix, regex)
    maple_output(out_keys)

def maple_output(keys_list):
    """
    Prints generated keys to stdout. This is the output
    format of the maple framework.
    """
    for n in range(len(keys_list)):
        if n == (len(keys_list) - 1):
            print(f"{keys_list[n]}", end="")
        else:
            print(f"{keys_list[n]}", end=" ")

def key_value_pair_formatter(key, value):
    """ Format key value pair with delimiter"""
    return f"{key}*{value}\n"

def maple_filter(input_string):
    """
    Remove special characters from the input string.

    Parameters:
    - input_string (str): The input string from which characters will be removed.
    - characters_to_remove (str): A string containing the characters to be removed.

    Returns:
    - str: The input string with specified characters removed.
    """
    result = ""
    special_characters = [
        '!', '"', '#', '$', '%', '&', "'", '(', ')', '*',
        '+', ',', '-', '.', '/', ':', ';', '<', '=', '>',
        '?', '@', '[', '\\', ']', '^', '_', '`', '{', '|',
        '}', '~', '\n', ' '
    ]
    for char in input_string:
        if char not in special_characters:
            result += char
    return result
```

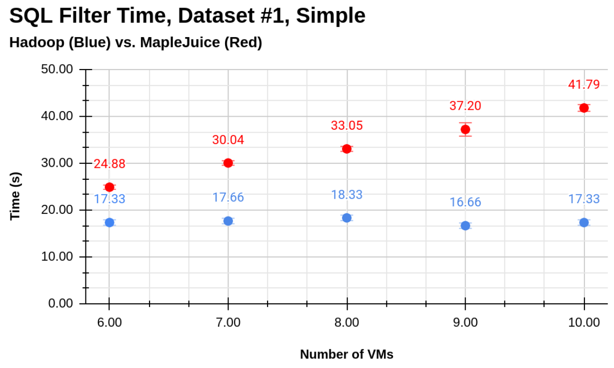
Figure 1: Maple Framework (Above)



maxma2	filter	MAPREDUCE	default	0	Sat Dec 2 22:16:16 -0600 2023	Sat Dec 2 22:16:16 -0600 2023	Sat Dec 2 22:16:31 -0600 2023
maxma2	filter	MAPREDUCE	default	0	Sat Dec 2 22:15:37 -0600 2023	Sat Dec 2 22:15:38 -0600 2023	Sat Dec 2 22:15:52 -0600 2023
maxma2	filter	MAPREDUCE	default	0	Sat Dec 2 22:15:03 -0600 2023	Sat Dec 2 22:15:03 -0600 2023	Sat Dec 2 22:15:20 -0600 2023

Figure 2: Hadoop User Interface Time Measurements

Direct comparisons were made between the performance of MapleJuice and Hadoop in eight different scenarios. The first four sets of measurements below are from FILTER queries on varying complexities of regex expressions. The cluster sizes go from six to ten VMs.



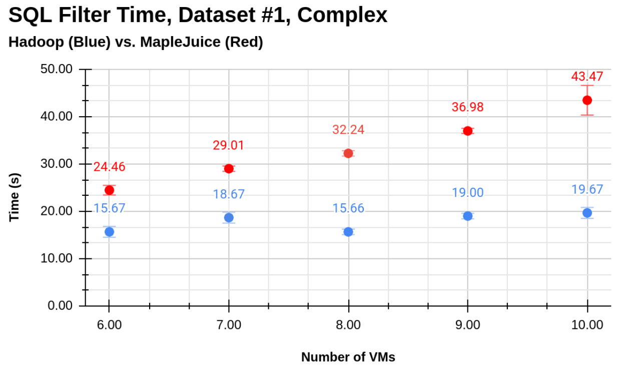
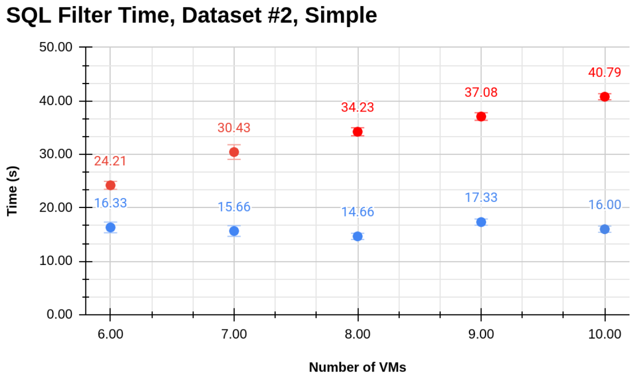


Figure3: “Yellow” on Source 1

Figure 4: “Center|Chestnut” on Source 1

Figure 3 shows the results from both frameworks for a filter query on a simple regex, “Yellow”, and Figure 4 for a complex regex, “Center|Chestnut”.



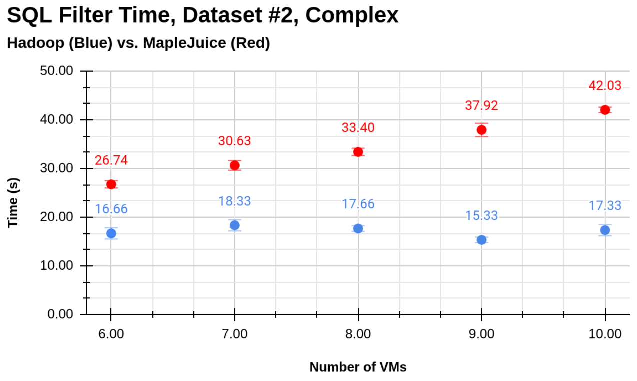


Figure 5: “Concrete” on Source 2

*Figure 6: “Arm.*150” on Source 2*

Figure 5 shows the results from both frameworks for a filter query on a simple regex, “Concrete”, and Figure 6 for a complex regex, "Arm.*150".

As shown above, MapleJuice takes longer to execute the filter queries compared to Hadoop, whose execution time was relatively constant, as the cluster never scheduled more than 2-3 tasks per job in any scenario. Surprisingly, MapleJuice performed worse with more VMs, likely due to large communication overhead of adding more nodes. Contributing factors include the gossip-style failure detection and partitioning of the input dataset among nodes in the cluster.

The next four sets of measurements compare the frameworks on JOIN queries. Again, the regex expressions are of varying complexity, but instead of clusters, the data sizes were changed.

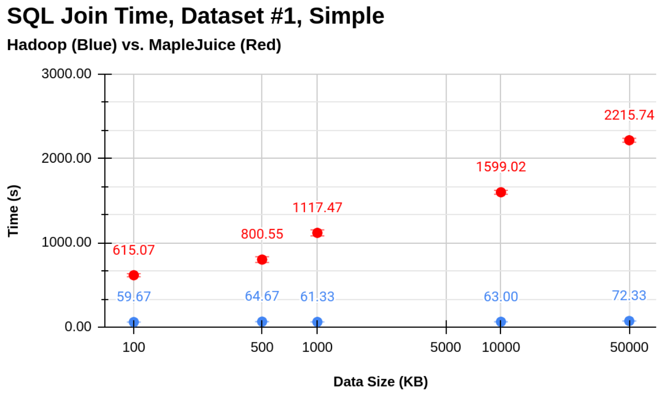


Figure 7: "D1.X = D2.X" on Sources 1, 2

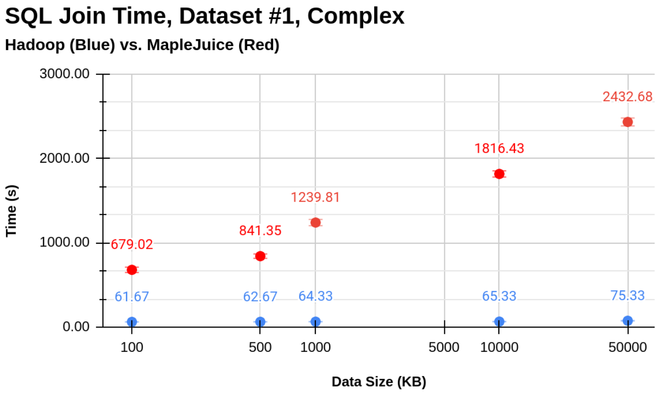


Figure 8:"D1.Pole_Material = D2.Color" on Sources 2, 3

Figure 7 shows the results of a join query on a simple conditional expression, "D1.X = D2.X", and Figure 8 for a complex conditional expression, "D1.Pole_Material = D2.Color".

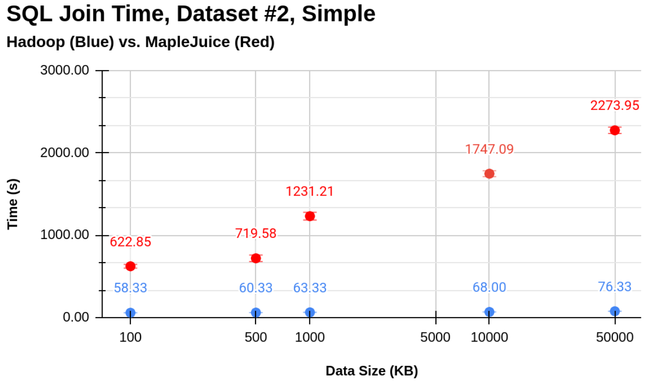


Figure 9: "D1.X = D2.X" on Sources 3, 4

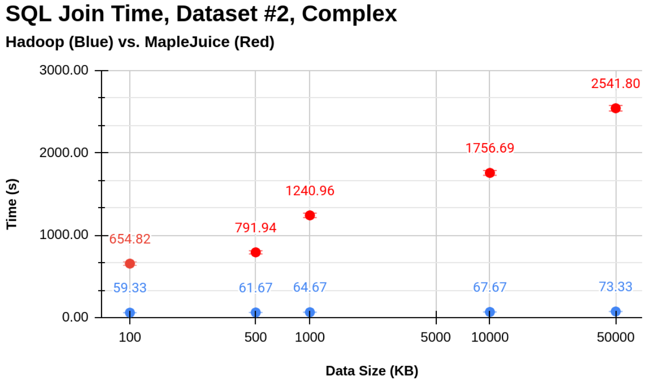


Figure 10:"D1.Rate = D2.Price" on Sources 2, 3

Figure 9 shows the results of a join query on a simple conditional expression, "D1.X = D2.X", and Figure 10 for a complex conditional expression, "D1.Rate = D2.Price".

Here, the difference between the execution times is even more pronounced. This may be caused by MapleJuice's replication of all intermediate key files in SDFS, growing quickly with the size of the input data. As mentioned before, a potential optimization would be to merge the results of the Maple phase to reduce the number of key files to be replicated later.

Sources:

1. https://gis-cityofchampaign.opendata.arcgis.com/datasets/cityofchampaign::traffic-signal-intersections/explore
2. https://gis-cityofchampaign.opendata.arcgis.com/datasets/cityofchampaign::streetlights/explore
3. https://gis-cityofchampaign.opendata.arcgis.com/datasets/cityofchampaign::parking-permit-spaces/explore
4. https://gis-cityofchampaign.opendata.arcgis.com/datasets/cityofchampaign::parking-meter-spaces/explore