CSC 462/562

Evaluation of Distributed System (Bitcoin Miner)

Intro and Motivation:

We wanted to evalulate the efficiency of our distributed system, and wanted to see how the performance (total run-time) changes with respect to different variables. The way we saw it, there were three variables: the number of nodes (as miners), max nonce (how much load do the miners handle), the run-time. In our implementation, the number of nonce each node handles was directly related to max nonce, thus we excluded it from our argument.

Then, there were two cases that were most interesting. Case 1 was when we fixed the number of max nonces, and increased the number of nodes, to observe how performance changed. Case 2 was when we fixed the number of nodes, and increased the number of max nonces, to observe how performanced changed.

Our initial hypothesis was that the as the number of nodes increase, the performance will also increase, because having more nodes correlates to more computing power. We also hypothesized that increasing the number of nonces would decrease performance, because increasing load on our distributed system would make our performance suffer.

Source code

In [38]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import csv
from scipy.interpolate import BSpline
```

Question: (Case 1) How does increasing the number of nodes affect performance?

We hypothesized that increasing the number of nodes would relate to increased performance (= less total running time), and CPU power.

We fixed the number of nonces in each of the below cases, and at the same time increased the number of nodes. The results are as follows.

In [33]:
```
#max nonce fixed @ 99999
dataOne = pd.read_csv('/Users/joshua/desktop/analysis1.csv')
```

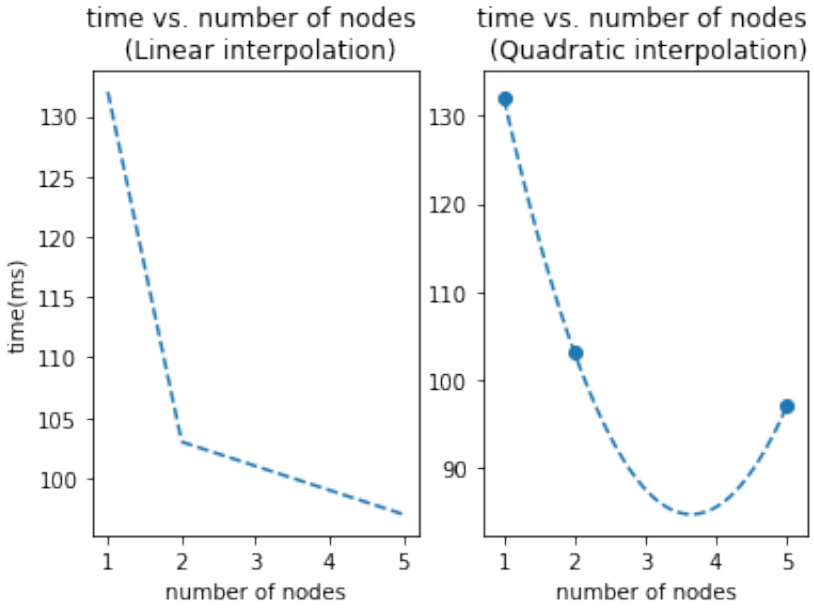
In [34]:
```
#max nonce fixed @ 999999
dataTwo = pd.read_csv('/Users/joshua/desktop/analysis2.csv')
```

In [35]:
```
#max nonce fixed @ 9999999
dataThree = pd.read_csv('/Users/joshua/desktop/analysis3.csv')
```

In [36]:
```
#evalute case where max. nonce fixed @ 99999 and number of nodes increas
numNodes = dataOne["number_of_nodes"]
runTime = dataOne["total_running_time"]
```

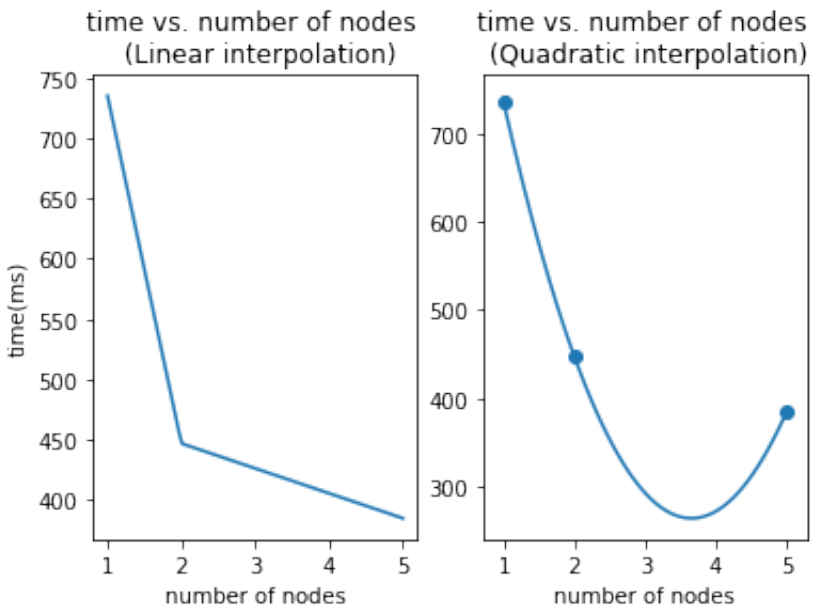
In [39]:
```
#plot linear interpolation & quadratic interpolation of the points
def plot_graph(numNodes,runTime, ls):
    xnew= np.linspace(numNodes.min(), numNodes.max(), 100)
    spline = make_interp_spline(numNodes,runTime,k=2)
    smooth = spline(xnew)
    plt.scatter(numNodes, runTime)
    plt.subplot(1,2,2)
    plt.plot(xnew, smooth, linestyle = ls)

    spline2 = make_interp_spline(numNodes, runTime, k=1)
    smooth = spline2(xnew)
    plt.scatter(numNodes, runTime)
    plt.subplot(1,2,1)
    plt.plot(xnew, smooth, linestyle = ls)

    plt.subplot(1,2,1).title.set_text("time vs. number of nodes \n (Line
    plt.subplot(1,2,2).title.set_text("time vs. number of nodes \n (Quad

    plt.subplot(1,2,1).set_xlabel("number of nodes")
    plt.subplot(1,2,2).set_xlabel("number of nodes")
    plt.subplot(1,2,1).set_ylabel("time(ms)")
```

In [40]:
```
plot_graph(numNodes, runTime, "--") # plot of graphs where nonce fixed @
```



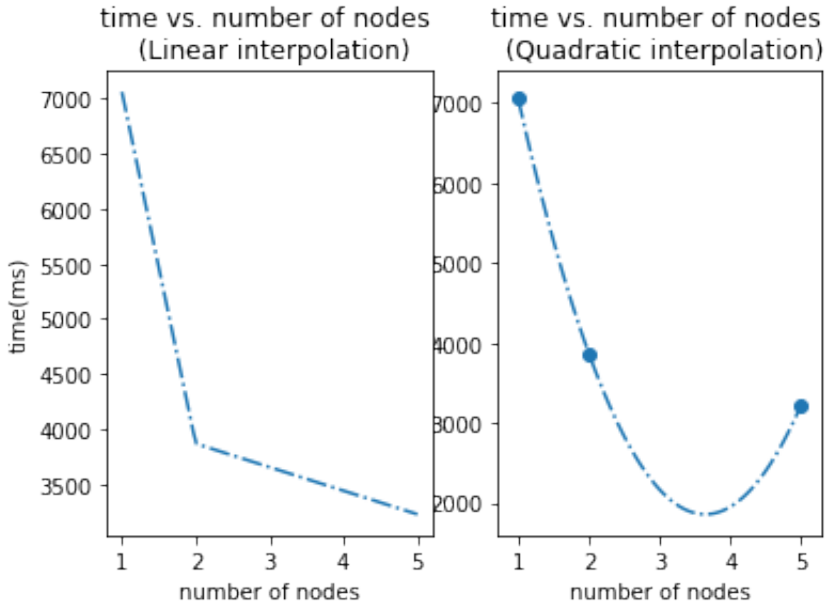
In [41]:
```
#evalute case where max. nonce fixed @ 999999, and number of nodes incre
numNodes2 = dataTwo["number_of_nodes"]
runTime2 = dataTwo["total_running_time"]
```

In [42]:
```
plot_graph(numNodes2, runTime2, '-') # nonce fixed @ 999999
```



In [43]:
```
#evalute case where max. nonce fixed @ 9999999
numNodes3 = dataThree["number_of_nodes"]
runTime3 = dataThree["total_running_time"]
```

In [44]:
```
plot_graph(numNodes3, runTime3, '-.') # nonce fixed @ 9999999
```



Explanation for Case 1

As the number of nodes increased, the performance improved (required less time to complete operation). But there was a caveat. When we plotted the quadratic interpolation of the points for a best-fitting line, and it indicated an upward trend past a certain number of nodes. Unlike our hypothesis that more equals better, this got us thinking, and we began to tinker with the idea that there could be an upper threshold to the number of nodes after which performance DECREASES.

Question: (Case 2) What happens when we fix number of nodes and change max_nonce?

In [45]:
```
#evalute case where number of nodes fixed @ 3;
dataThreeNode = pd.read_csv('/Users/joshua/desktop/Re__single_vs._distri
dataThreeNode # data for fixed node = 3, and different # of nonce each n
```

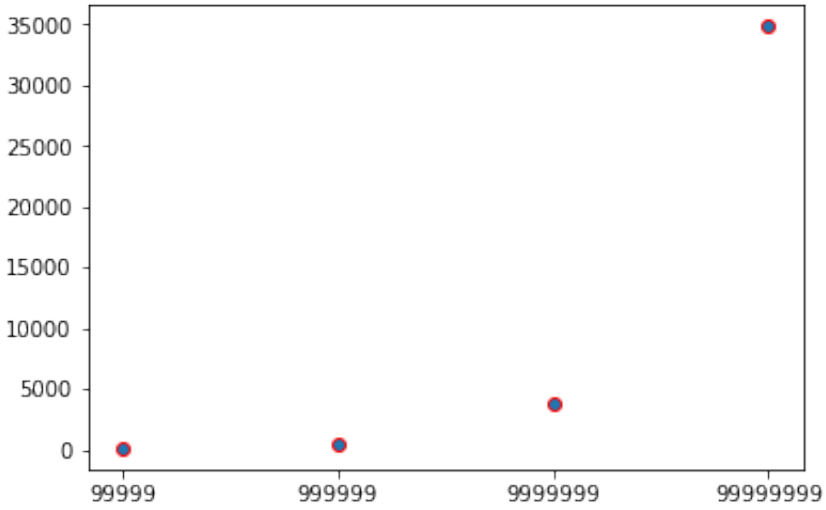
Out[45]:

	number_of_nodes	max_nonce	number_of_nonce_each_node_handle	total_running_time
0	3	99999	30000	110
1	3	999999	300000	424
2	3	9999999	3000000	3857
3	3	99999999	30000000	34820

In [46]:
```
maxNonceChange = dataThreeNode['max_nonce']
timeRun = dataThreeNode['total_running_time']
```

In [47]:
```
#plot points at different max_nonce values
def fixedNoncePlot(maxNonceChange, timeRun):
    x = [point for point in range(len(maxNonceChange))]
    plt.scatter(x, timeRun, edgeColor = 'r')
    plt.xticks(x, maxNonceChange)
    plt.show()
```

In [48]:
```
#plot of points where node = 3 and maxnonce changes
fixedNoncePlot(maxNonceChange, timeRun)
```



In [49]:
```
#evalute case where number of nodes fixed @ 3;
dataFiveNode = pd.read_csv('/Users/joshua/desktop/Re__single_vs._distrib
```

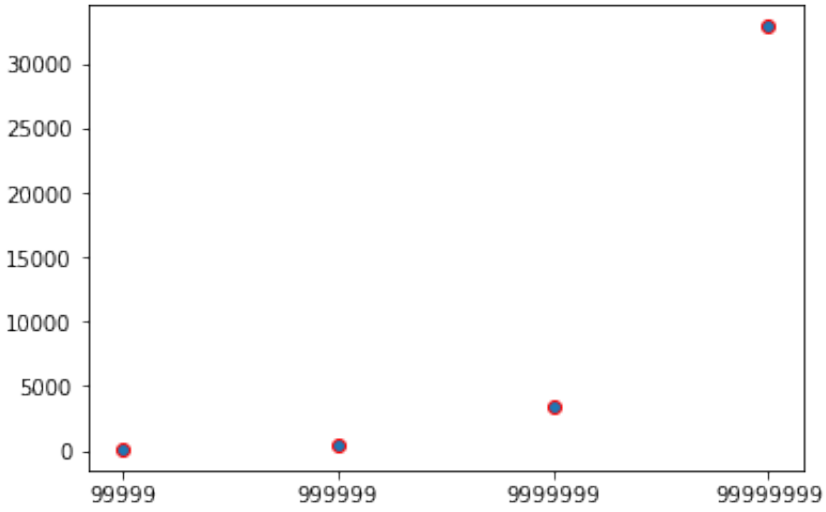
In [50]:
```
dataFiveNode.head() # data for fixed node = 5, and different number of n
```

Out[50]:

	number_of_nodes	max_nonce	number_of_nonce_each_node_handle	total_running_time
0	5	99999	30000	98
1	5	999999	300000	406
2	5	9999999	3000000	3407
3	5	99999999	30000000	32959

In [51]:
```
maxedNonceChange = dataFiveNode['max_nonce']
timedRun = dataFiveNode['total_running_time']
```

In [52]:
```
#plot of points where node = 5 and maxnonce changes
fixedNoncePlot(maxedNonceChange, timedRun)
```



Explanation for Case 2

From the two graphs above, we learned that there is a definitive upward trend, as the number of nonces increase. This was expected in our hypthesis. It's only logical that if we put more load on our nodes that our performance would suffer. It would be in our best interest to find a optimal nonce value so that our performance is maximized.

Question: (Case 1 Continued) How does increasing the number of nodes affect performance?

In our discussion of Case 1, we stated that there seems to be an upper threshold, but we were inconclusive because there were few data points to consider definitively. In this part, we aim to dig deeper into this issues, and try to come up with an explanation.

In [53]:
```
#load csv with fixed max_nonce, and more nodes & data points
dataDifNodes = pd.read_csv('/Users/joshua/desktop/Re__single_vs._distrib
```

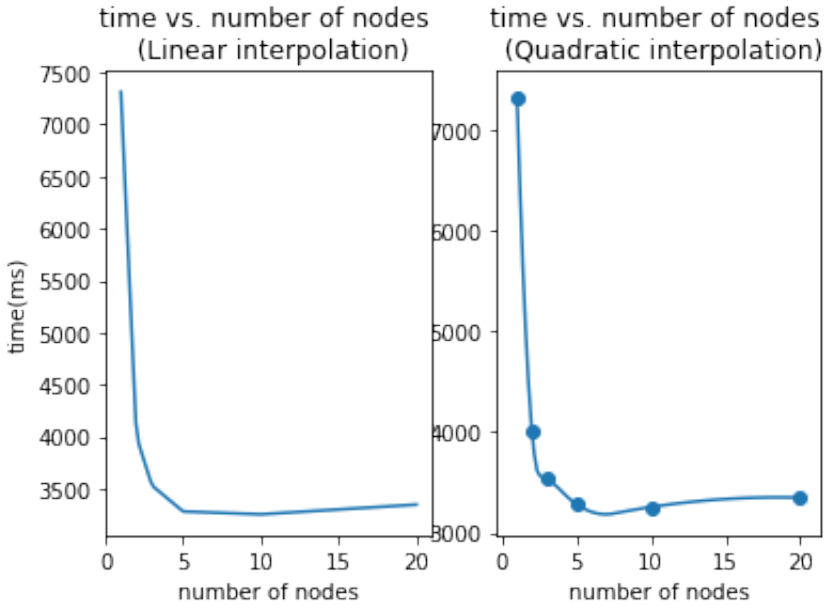
In [54]:
```
dataDifNodes # data for fixed max nonce 9999999, and different # of node
```

Out[54]:

	number_of_nodes	max_nonce	number_of_nonce_each_node_handle	total_running_time
0	1	9999999	9999999	7312
1	2	9999999	5000000	4001
2	3	9999999	3000000	3531
3	5	9999999	2000000	3282
4	10	9999999	1000000	3255
5	20	9999999	500000	3348

In [55]:
```
#evalute case where max_nonce fixed @ 9999999, and number of nodes go fr
difNumNode=dataDifNodes['number_of_nodes']
difRunTime=dataDifNodes['total_running_time']
```

In [56]:
```
#plot the graph
plot_graph(difNumNode, difRunTime, "-")
```



In [57]:
```
#evaluate case where max_nonce is fixed @ 999999, and number of nodes go
dataLessDifNodes = pd.read_csv('/Users/joshua/desktop/Re__single_vs._dis
```

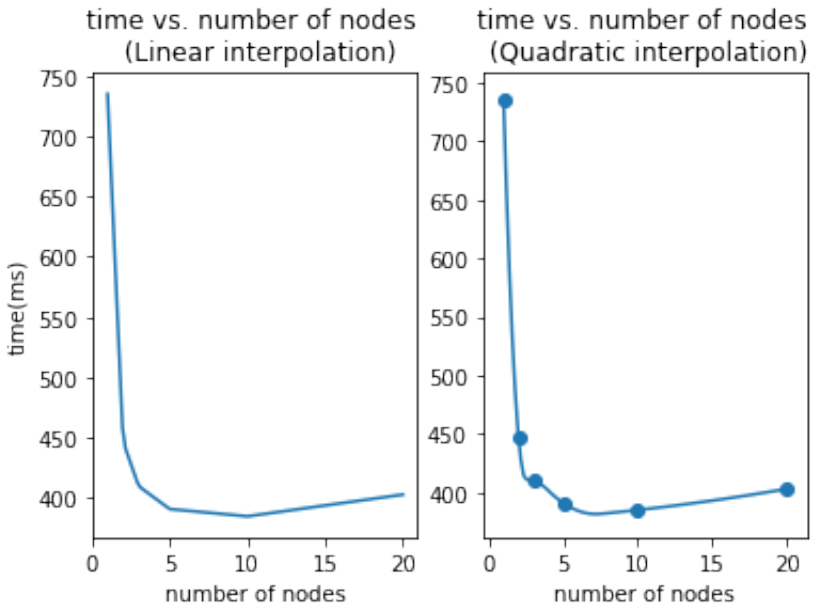
In [58]:
```
dataLessDifNodes # data for fixed max nonce = 999999, and different # of
```

Out[58]:

	number_of_nodes	max_nonce	number_of_nonce_each_node_handle	total_running_time
0	1	999999	999999	735
1	2	999999	500000	447
2	3	999999	300000	410
3	5	999999	200000	391
4	10	999999	100000	385
5	20	999999	50000	403

In [59]:
```
difNumNode=dataLessDifNodes['number_of_nodes']
difRunTime=dataLessDifNodes['total_running_time']
```

In [60]:
```
#plot
plot_graph(difNumNode, difRunTime, "-")
```



(Conclusive) Explanation for Case 1:

When max-nonce was fixed, increasing the number of nodes past node 10 didn't increase performance. In fact, it decreased performance, as seen on the above graph. We initially thought that having more nodes would always be better, because it would be better to have nodes idle than to be in need. However, we learned that past a certain threshold, having more nodes isn't ideal for performance. This is because since the server is concurrent, it has to maintain and keep track of miners, as well as clients. After farming out tasks to appropriate miners, maintaining idle miners becomes extra cost that the server has to deal with (listen for incoming request, keep its minerID, etc). Due to this, it is an important to note that there is trade off and more miners != improved performance.

Conclusion

From single-node (n=1) application to distributed (n>1) application, our project showed tremendous improvement in performance, as seen in this evaluation. At the same, we learned that distributed systems, too, isn't without its trade offs. Having many miners for a big task is good and improves performance, but having many miners for a small task quickly becomes burdensome and less efficient. As in building any architecture, measuring the trade offs and tackling the problem with the right tools in hand would be ideal.