Calculating Large Prime Numbers using Distributed Clusters

and Related Performance Analysis

Vineet Pandit
Erin Moy
Thomas Reed

Abstract:
We argue that it is possible to construct a program which runs a prime-searching algorithm over a distributed cluster at a higher speed than a similar algorithm running on a single machine. Network overhead is a considerable problem for such an implementation, but this paper shows that an algorithm can be constructed with sufficient parallelization that the increased speed of calculation overcomes the network overhead. We show that modifying several variables has an affect on network overhead and give examples of execution time for different runs.

Acknowledgements:
We'd like to thank our professor, Dr Wang, for making this project possible. We'd also like to thank all major distributed computing networks for providing inspiration. We'd also like to thank Google, because Google Drive made collaboration possible.

Table of Contents

1 Design Documentation and Flowchart

Server Algorithm

1. Server starts up
2. Accepts arguments for
    a. Final limit
    b. Chunk Size
    c. Initial Calculation
    d. A list of all clients (no port numbers, normalizes at 10101)
3. Open socket connection with all clients and sends limit to them (clients are already waiting). Open a separate thread for each client.
4. Calculate all values up to Initial Calculation and adds it to an ArrayList primesList
5. Add primes to an ArrayList toBeBroadcasted for all clients.
6. Broadcast ToBeBroadcasted to all clients
7. For each client, in their separate thread, send a chunk assignment
    a. First, check if toBeBroadcasted contains anything. If it does, send toBeBroadcasted, wait for client to send “OK,” and then broadcast the chunk assignment
8. When response is received from client, add result (if applicable) to primesList and repeat from step 5 until limit is reached

Client Algorithm

1. Open client and wait for connection from server
2. When client receives primes:
    a. Add to Known Primes
    b. Send OK
3. When client receives an assignment
    a. Calculate all primes over that assignment
    b. If any primes in that are less than sqrt(limit), send everything back
    c. Otherwise send OK

- Why square root? What does Chunk Size do?

2 Project Assumptions

a *Structure of the EDC computers in our cluster*

Our performance analysis of the cluster program mentioned specifically that we will be running the cluster on the EDC machines. This has several advantages and disadvantages. First, the computers are homogenous: all clients will run at relatively the same speed, communication is approximately the same speed, and they're easy to set up through SSH. Second, they're all on the same network switch, which means that communication is extremely quick. The greatest disadvantage is inconsistency. It's difficult to control how much work the individual machines are doing since another user could be using or SSHing into them and running their own programs during our runtime. Be that as it may, we've controlled for that by running several trials for each data point, reducing outliers.

b *Why the clients are not sending complete data back to the server*

Our first implementation of the project had the clients send a complete list of primes found back to the server, with the server only rebroadcasting the primes up to sqrt(limit). We quickly discovered that this implementation was infeasible and scrapped it. The reason is due to the structure of the problem and how it differs from other distributed solutions: consider the program Prime95. Computers running Prime95 calculate large prime numbers, then occasionally send a single found prime back to the coordinator. Our program sends hundreds of millions of doubles during the course of its execution. One of our largest runs, with a limit of 100 billion, involves 50,847,534 primes which would correspond to at least 406 megabytes of data sent to the server by the clients. The scope of our problem is to find the speed at which the calculation would complete given the overhead from the server, not the speed at which the output would complete. We theorize that it's possible to find some massive upper limit even with that constraint but, since such a large number (probably over ten trillion) would likely crash the EDC machines the cluster runs on, we decided that's outside the scope of our project.

c *Why are our calculations going up to sqrt(limit) instead of all the way to limit?*

The Sieve of Eratosthenes, which we're using to calculate our primes, functions by setting all numbers which are multiples of a prime to composite, and what's left over must be prime. In order to be composite, a number must be able to divided by other numbers other than 1 and itself. It stands to reason that any composite number which has a number larger than the sqrt as a factor is the product of that number and a number less than the sqrt, and thus would have already been caught by the sieve.

d *Why we're using static port numbers*

Our first implementation of the coordinator and calculation nodes used dynamic port numbers. When we got to the testing phase, we realized that every one of the hundreds of tests we had to

run would take several minutes just to set up, so we scrapped the dynamic ports and made them static. Every port is 10101, which saved us enough time to run many more tests.

3 Data Analysis and Discussion

a Goals

i. Find performance of clusters of various size as range size increases
ii. Find network overhead by comparing a cluster of size 1 to a non-cluster
iii. Find how the speed of a cluster is affected by chunk size
iv. Find data on scalability
    1. How much does adding one machine to the cluster affect speed?
    2. How much does adding one machine to the cluster affect overhead?

b Project Variables

i. Input
    1. Size of the cluster : How many computers are connected to the cluster at once. This specifically refers to the computers doing the calculation, not including the coordinator.
    2. Final Limit to search for : The programs are designed to find all prime numbers up to this limit. Limit is the upper bound of the calculation. Logarithmic scale.
    3. Size of chunks assigned to clients : Clients are assigned chunks of certain size to iterate over. Larger chunk sizes should theoretically decrease network overhead and make for faster program runtimes. Logarithmic scale.
ii. Results
    1. Execution time, the time it takes for the server to run from beginning to the final limit
iii. Artificial
    1. Speed, measured as (Limit / Execution Time) : The limit of a specific test divided by the average execution time over several trials for that specific limit. Our goals include finding how fast the programs perform calculations, for which we need to control for the length of the calculation

Controlled Speed, measured as (Speed * Number of nodes) : Controlling speed for the number of calculation nodes shows overhead as the number of nodes increases.

4 Graphs

i Speed vs Limit

1. Non-cluster single machine
2. Cluster size 1, 2, 3, 4

- This is the meat of our project, showing performance of the clusters as compared to performance of a non-cluster.
- Limit is logarithmic scale: f(x+1) = 2 * f(x)
- Expectations
  - Speed is slow at the beginning due to setup overhead, but increases as the limit gets very large
  - Speed becomes asymptotically close to (approaches) a limit for a certain N

Results Analysis

- The Performance for all clusters is not a flat number. It begins low, quickly rises, but then decreases very slowly
- The performance of any cluster with size > 1 is better than the performance of a single machine

We expected significantly larger network overhead, but the constraints of our project were constructed in such a way that those results are unsurprising. The latency of machines running in the same engineering center, on the same network switch, could be low enough to decrease network latency and, therefore, network overhead.

The most surprising results were for very large numbers. We expected the speed to get asymptotically close to an upper limit, not to decrease, however slightly, as numbers became very large. We believe this has to deal with the structure of the machines themselves. The decrease begins happening at about 10 billion, which is 10^10, and the sqrt(10^10) = 10^5, or 10K. Our primes are stored as doubles, meaning that the decrease begins around the time that the machine is storing 80KB. The Dell Precision T3500 machines in the EDC are using X58 motherboards, which means they're using 1366 sockets, which means that their L1 cache is 32kb and the system is swapping data starting at the 5 billion run -- and looking closely, we see that the 5 billion run is statistically equivalent to the 10 billion run, corresponding to the point at which network overhead is about equivalent to the slow from swapping.

ii Speed vs Limit

1. Non-cluster single machine
2. Single-machine cluster

- The intention of this graph is to show the overhead of the cluster. On both of these cases, the calculation is being performed by a single machine, but the second has network overhead communicating with the server in addition to those calculations.
- Expectations:
  - The network overhead that the single-node cluster suffers means that it is slower than the non-cluster single machine for all runs

Results Analysis:

- The single-client solution quickly drops off in speed as limit becomes large
- Network overhead is significant
- Network overhead remains roughly constant before single-client dropoff

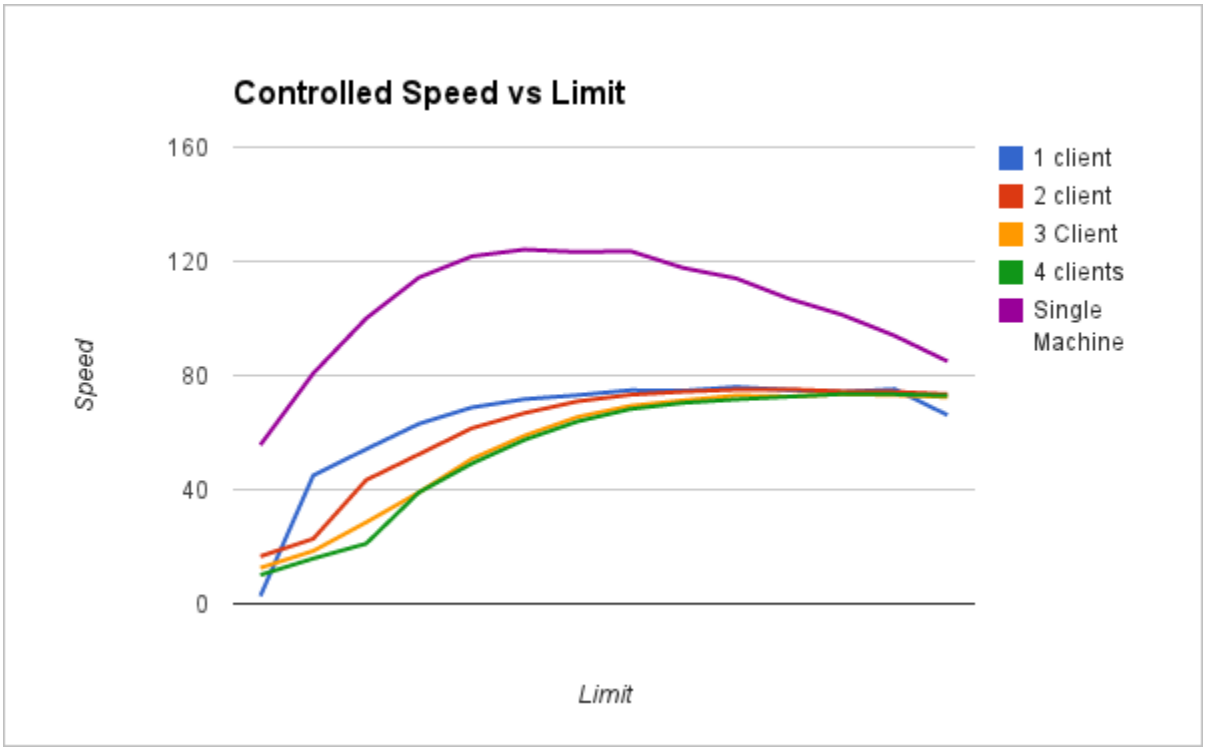
The speed dropoff of the non-distributed program is likely due to structural differences in the program. The single computer solution is a repurposed implementation of the cluster programs, but the cluster programs were rebuilt to be more efficient after the two directions were split. The decrease in performance, though on a different scale, has exactly the same root cause on both programs: size of the cache. The single-computer solution appears to be storing more doubles, and the greater dropoff in performance is probably because it's surpassing L2 cache instead of just L1.

Regardless, the intention of the graph is to find the overhead between the two programs, and we can clearly see that the difference is significant. More importantly, network overhead remains roughly constant until the performance dropoff of the single-machine program.

iii Controlled Speed vs Limit

1 1-4 nodes

- The intention of this graph is to show how network overhead increases as the number of clients increases.
- Expectations :
  - As the number of clients increases, speed/client (Controlled Speed) decreases
  - This difference remains as a percentage of total speed over a range of limits



Results Analysis :

- For small limits, controlled speed is smaller for larger numbers of clients
- For very large limits, controlled speed is approximately equal across all cluster sizes

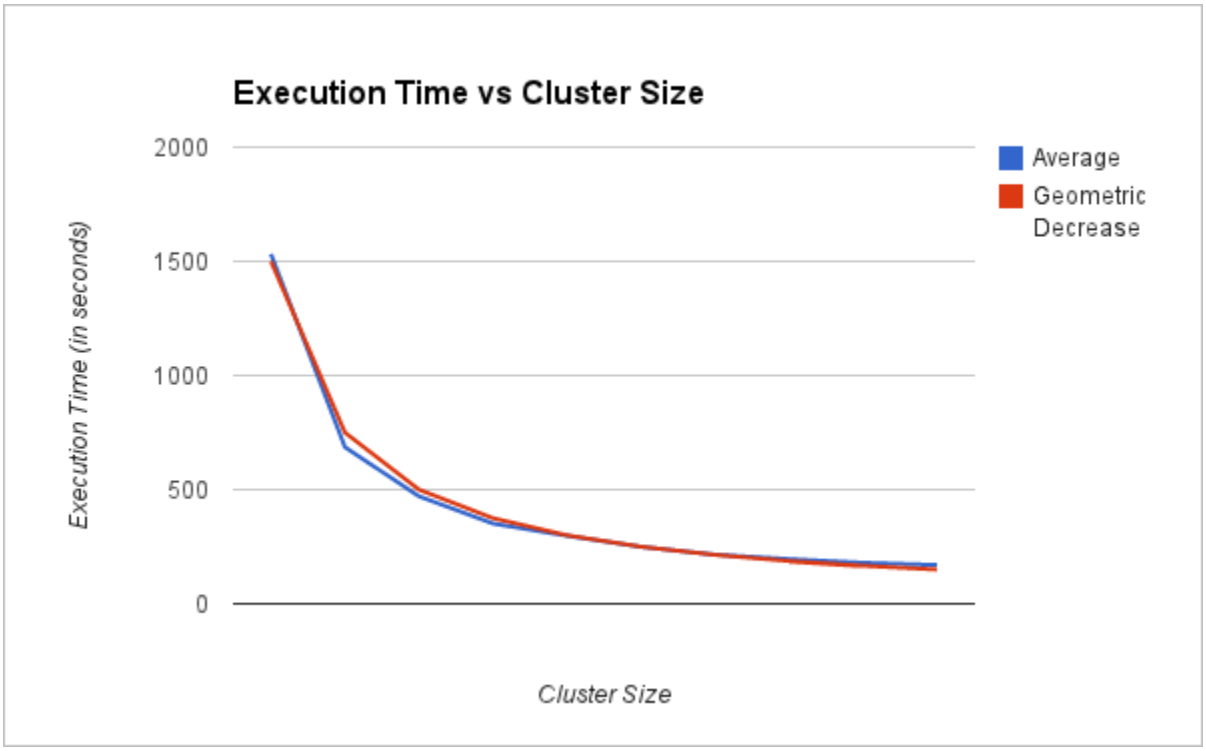
We didn't expect these results at all. First, controlled speed approaches a constant across all clients, meaning that increased overhead for adding one more computer to the cluster is approximately zero. We expected there to be at least some overhead, but all numbers look statistically equivalent.

Also interestingly, controlled speed is smaller at the lower limits than at the larger limits. Remember that controlled speed is ((Execution Time / Limit) / Cluster Size). Lower limits, at which less time is used for the major calculations and more is used for setup time, have larger numbers for larger numbers of clients. This appears to contrast with what we expect, which is a larger setup time for larger numbers of clients. Effectively, setup time is much smaller than we anticipated. This more than anything argues to the validity of our hypothesis: overhead for a distributed solution is relatively tiny.

iv Execution time vs Cluster Size for large numbers

1. Controlling for Limit = 100 billion, how is execution time affected as the number of clients approaches a large number?
2. Contrast this with a geometric decrease you would expect from splitting the calculations up perfectly (starting at same first value)
3. Measure overhead as number of clients increases

- How Scalable is our implementation? How much overhead for large numbers of clients?
- Expectations :
  - Larger cluster size decreases execution time geometrically
  - As the number of clients increases, overhead for larger cluster sizes causes the cluster to remain above the geometric graph



Results Analysis :

- The two lines are functionally identical

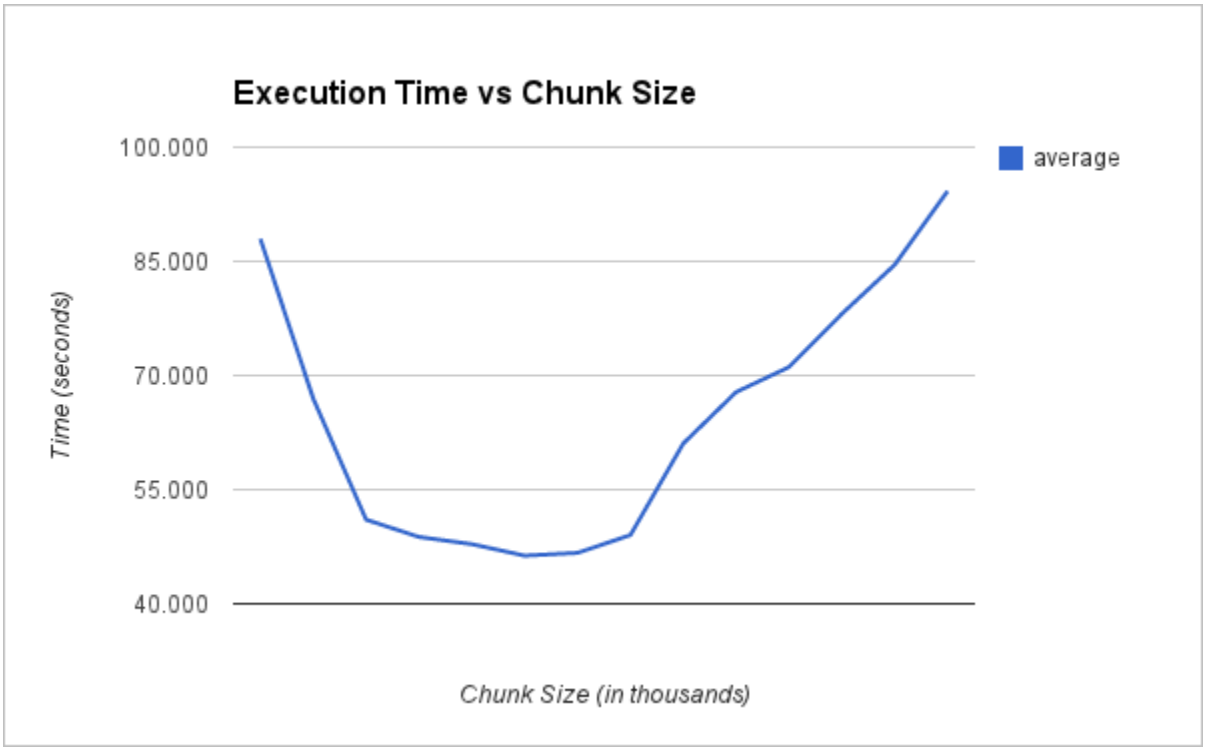
We can see that the results of this graph are consistent with the results of Controlled Speed vs Limit : the overhead for adding another client to the cluster is negligible. This is imperfect since the first value of the geometric series is estimated at 1500 and thus includes overhead at that level, but this graph shows that overhead doesn't increase disproportionately with the number of clients.

v **Speed vs Chunk Size**

1. Controlling for limit = 10 billion
2. Controlling for initial calculation = 1 million
3. Lines
    a. Number of clients: 3

- Expectations :
    - We expect execution time to decrease as chunk size approaches a certain fraction of the limit due to lowered network overhead
    - We expect execution time to increase after chunk size becomes larger than that fraction, due to being unable to fully utilize all computers in the cluster

The graph that uses chunk size also functions on a logarithmic scale: We started with chunk size of 100,000 and doubled every time to get numbers. Every other graph assumes chunk size is fixed at 10 million. See the attached spreadsheet with actual numbers.



Results Analysis :

- Execution time bottoms out between chunk sizes of 800K and 1.28 million, taking into account only numbers we've measured.
- This range may be specific for the constraints of this trial, being a limit of 10 billion or 3 clients. Further research is suggested to find a mathematically optimal value.

5 Conclusions

Our hypothesis implied that we expected network overhead to be rather significant and that we'd have to do major work or run multiple clients in order to reach equilibrium. In actuality, performance about evens out at two clients and is drastically better at three or four. This isn't perfect because of the structure of our problem: since we're measuring speed of calculation instead of speed of output, network overhead is minimized. However, network overhead is overcome quickly at even moderate limits, disproving the negative hypothesis.

The negative hypothesis is proved conditionally false in the specific case of our program, the case that no output is sent to the server -- such as when either the results of the equation are stored locally or transmitted at a later time.

6 Suggestions for further research

- Extremely large prime numbers
  - Our tests run to 80 billion. Tests at extraordinarily large numbers, such as exceeding 1 trillion, could show different data as the stored ArrayList size exceeds L2 or L3 cache. Alternately, rewrite the program to store all the primes instead of just up to sqrt (limit) and run performance analysis on that.
- Output being sent to an output server which does not stress the coordinator
  - It's not really possible to send the output back to the coordinator since it would increase overhead to a massive degree, but having the output sent to another server specifically designed to receive that output could simulate stress from data transmission on the client side and reduce the speed of calculation. The problem is that the sending of output to the output server would continue until long after the actual calculations are done and the main server quits.
- Finding mathematically optimal values for all variables
  - The graph of chunk size shows that there can be an optimal value for certain variables. Find a mathematically provable optimal value for number of servers and chunk size which takes into account network speed, speed of the processors on the EDC machines, and
- Run trials across geographically diverse machines
  - Network overhead was minor because our cluster was working across a 100 megabit LAN connection in the EDC. Running some trials with the coordinator off the grid could give us some more significant, if more random and less meaningful, values for network overhead.

7 Attached Documents
    a Code
        i SingleClientSolution
            1 SingleClientCalculator.class
            2 SingleClientSolution.class
            3 Makefile
        ii DistributedPrimeClient
            1 DistributedPrimeCalculator.class
            2 DistributedPrimeClient.class
            3 Makefile
        iii DistributedPrimeServer
            1 DistributedPrimeCalculator.class
            2 DistributedPrimeServer.class
            3 Client.class
            4 Makefile
    b README

    c Test Data
        i Spreadsheet of output data