COMP 8005
Project
Report

Harman Dhillon
A00994245
Dec 3, 2025

Purpose

- A distributed password-cracking system that demonstrates dynamic work distribution, checkpointing, worker failure recovery, and multi-threaded brute-force password cracking.

Requirements

Task	Status
● The server is unaware of the nodes; nodes must register with the server. ● Connects to each node after it has registered. ● Dynamically distributes work and a checkpoint interval (N password guesses) to each node. ● Nodes can register or disappear/disconnect at any time. ● Tracks remaining work and reassigns unfinished tasks when nodes disappear/disconnect. ● Stops distributing work once the password is found, but allows nodes to complete their currently working passwords. ● Prints the cracked password to the console when found. ● Must display whenever a node is registered, disappears/disconnects, what work each node is allocated, and checkpoint information for each node.	Fully implemented
Worker Nodes ● Registers with the server upon startup. ● Waits for the server to connect to it. ● Receives: ○ The hashed password to crack. ○ A range of passwords to attempt. ○ The checkpoint interval (N) from the server. ● Periodically send checkpoint updates every N attempts (sent from the server). ● Continue working until they receive a stop signal from the server. ● Gracefully handle disconnections (e.g., signal handlers).	Fully implemented
● The server must detect failures and redistribute unfinished work.	Fully implemented

● Nodes should be able to join/leave at any time and receive work.	🟩
● The specific message format is up to you (e.g., JSON, plain text, binary). ● The protocol must support: ○ New node connections ○ Work requests ○ Checkpoint updates ○ Node disconnections (this can be by inferring a crash or by an optional disconnection message - optional) ○ Cancel found notifications ○ Password found notifications	🟩 Fully implemented

Platforms

tested on:

- Omarchy 3.1.1

Language

- ISO C17
- Compiles with Cmake

Documents

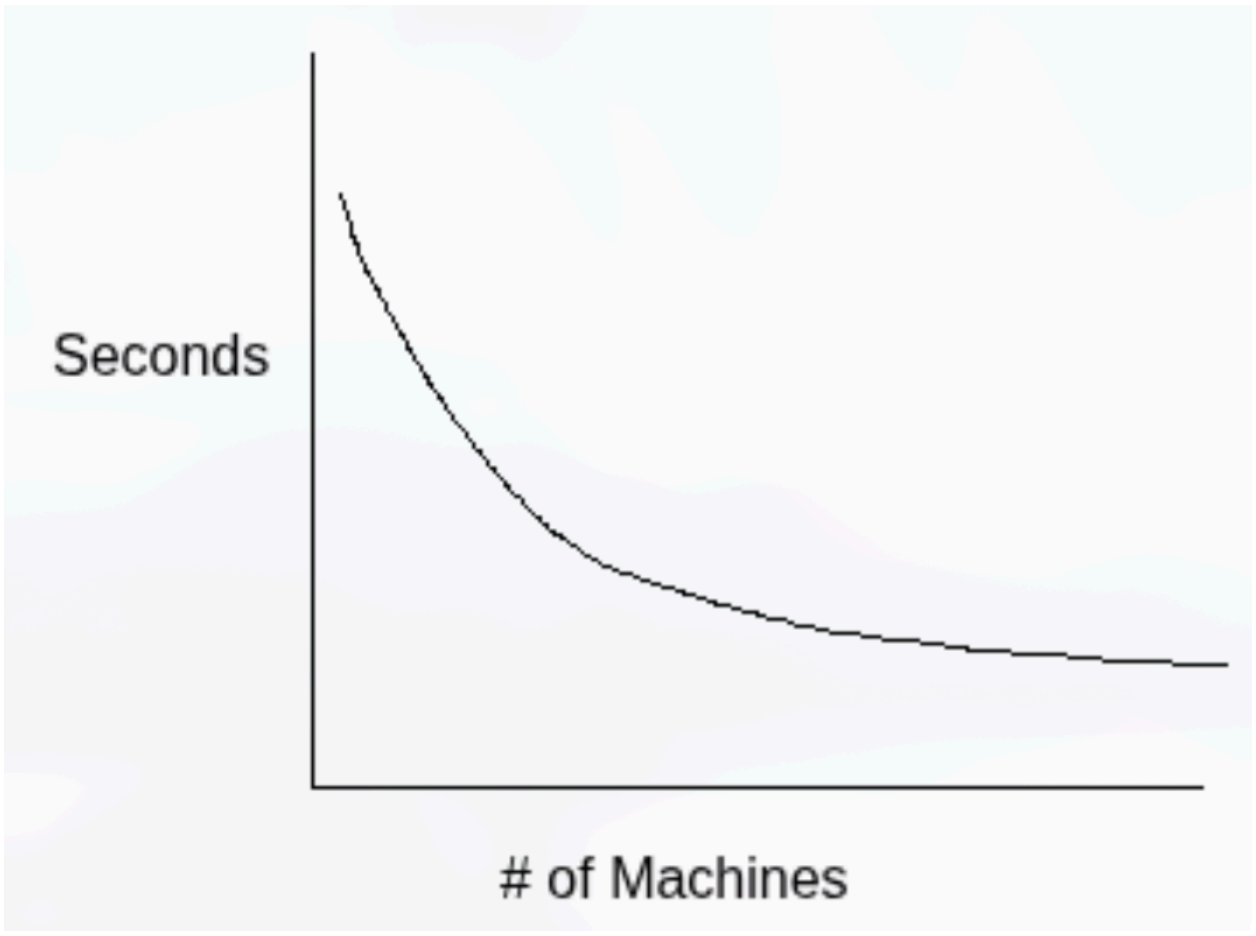
- Design
- Testing
- User Guide

These documents can be found in the report folder of this submission.

Findings

Initial Hypothesis

My hypothesis is that as the number of machines participating in the distributed cracking system increases, the total time required to find the correct password will decrease. This is because the workload is divided across multiple workers, allowing many different password ranges to be tested in parallel instead of a single machine processing the entire search space on its own. In theory, using N # of machines could provide up to an N× speedup, since each machine handles a fraction of the total work. However, the actual speedup will be less than that because distributed systems introduce additional overhead. Communication between the server and workers, network latency, checkpoint updates, work redistribution, and failure handling all contribute to inefficiencies. I also expect that after a certain number of machines, adding more workers will yield diminishing returns. Beyond this point, the system may become only slightly faster or not faster at all because network congestion will begin to outweigh the benefits of additional parallelism.

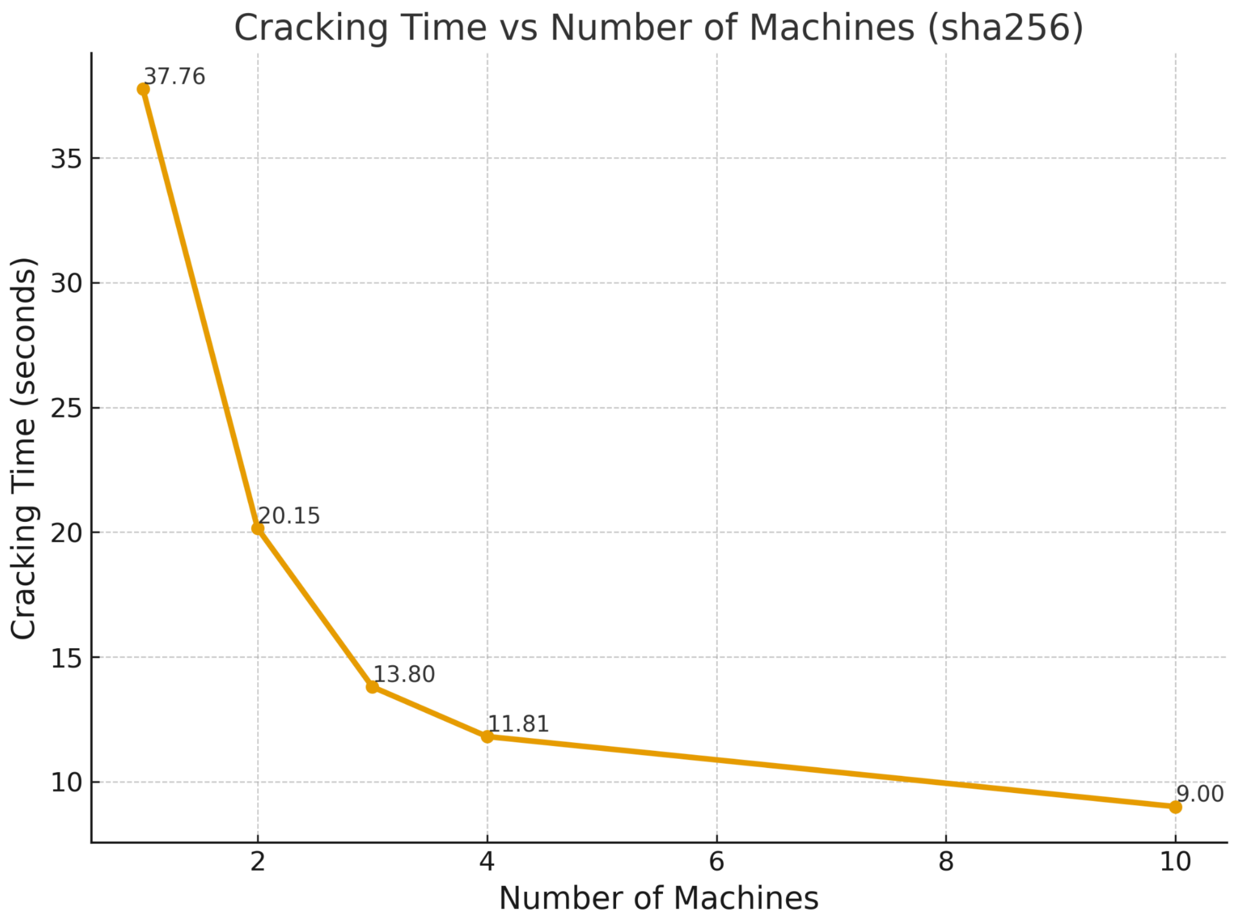


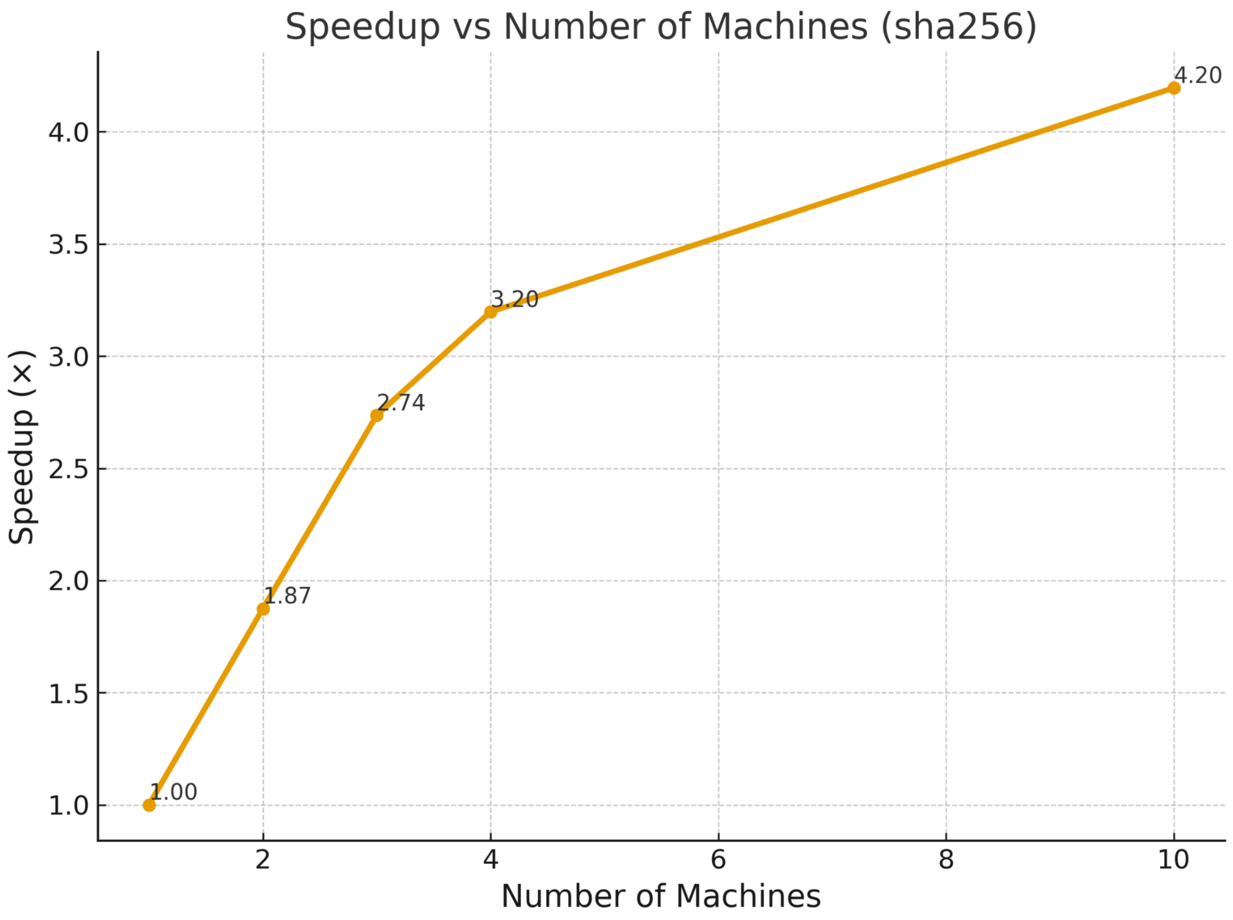
Performance Graph and Analysis

To test my hypothesis, I created hashes for the 3 character password “ABC” using 3 different hashing algorithms (SHA256, SHA512 and yescrypt). I then ran the cracker for each hash using 1, 2, 3, 4, 10 machines. The workers each ran with 4 threads. The checkpoint was at 200 while the work size was 1000. The timeout was 600 but that did not matter because no worker timed out.

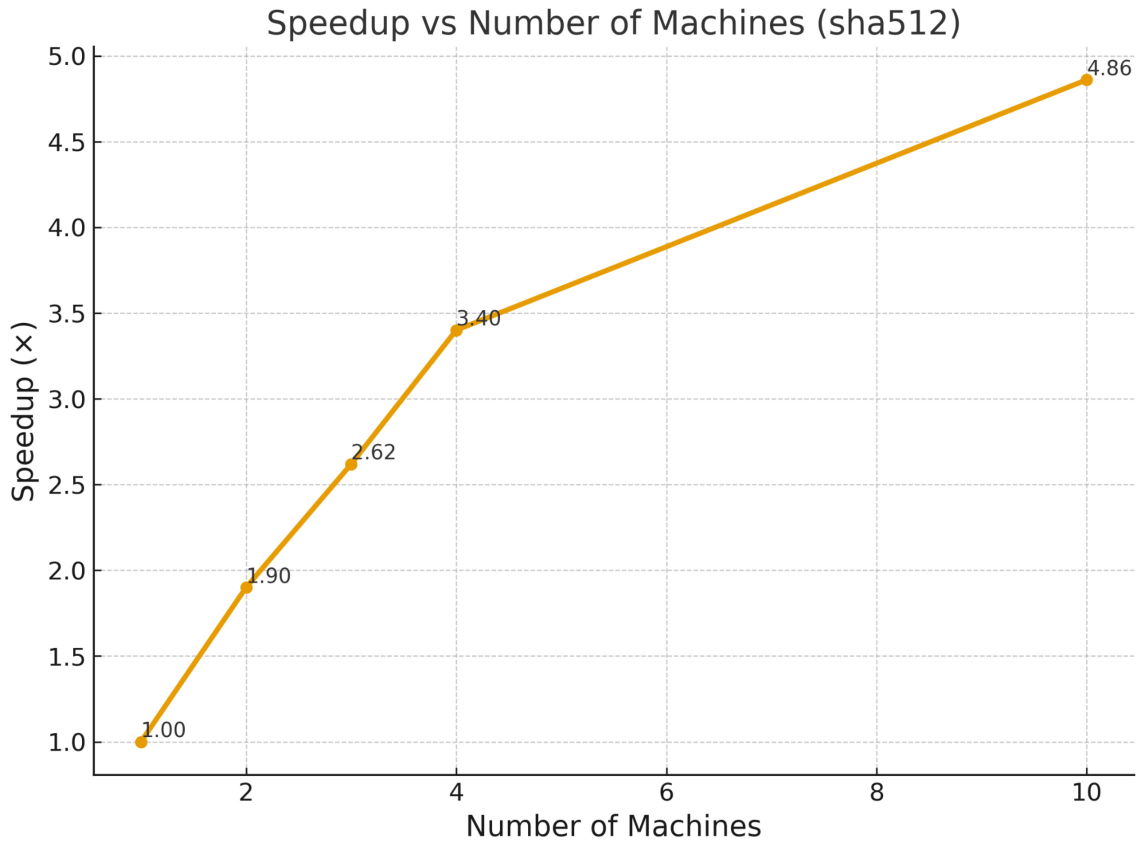
Graphs of Time VS Number of Machines For Each Hash

SHA 256

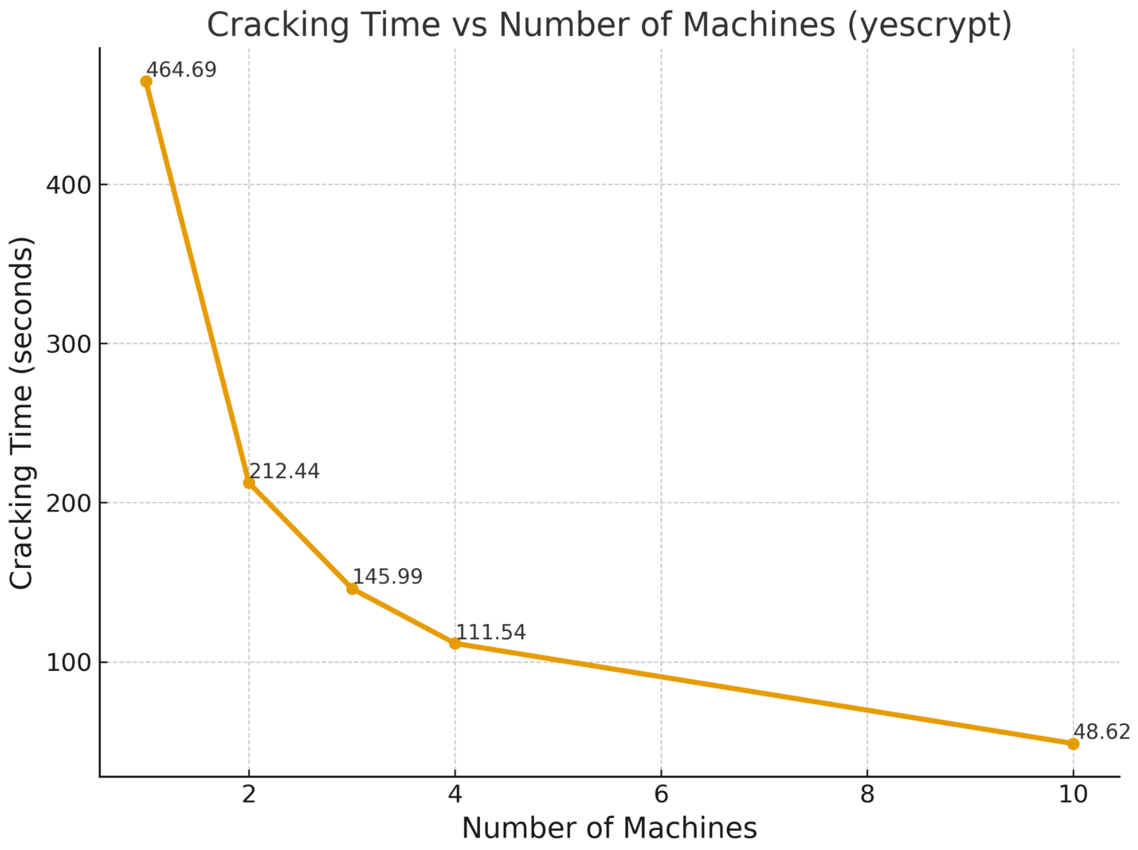



Speedup vs Number of Machines (sha256)
Speedup (×)
Number of Machines
1.00
1.87
2.74
3.20
4.20
1.0
1.5
2.0
2.5
3.0
3.5
4.0
2
4
6
8
10

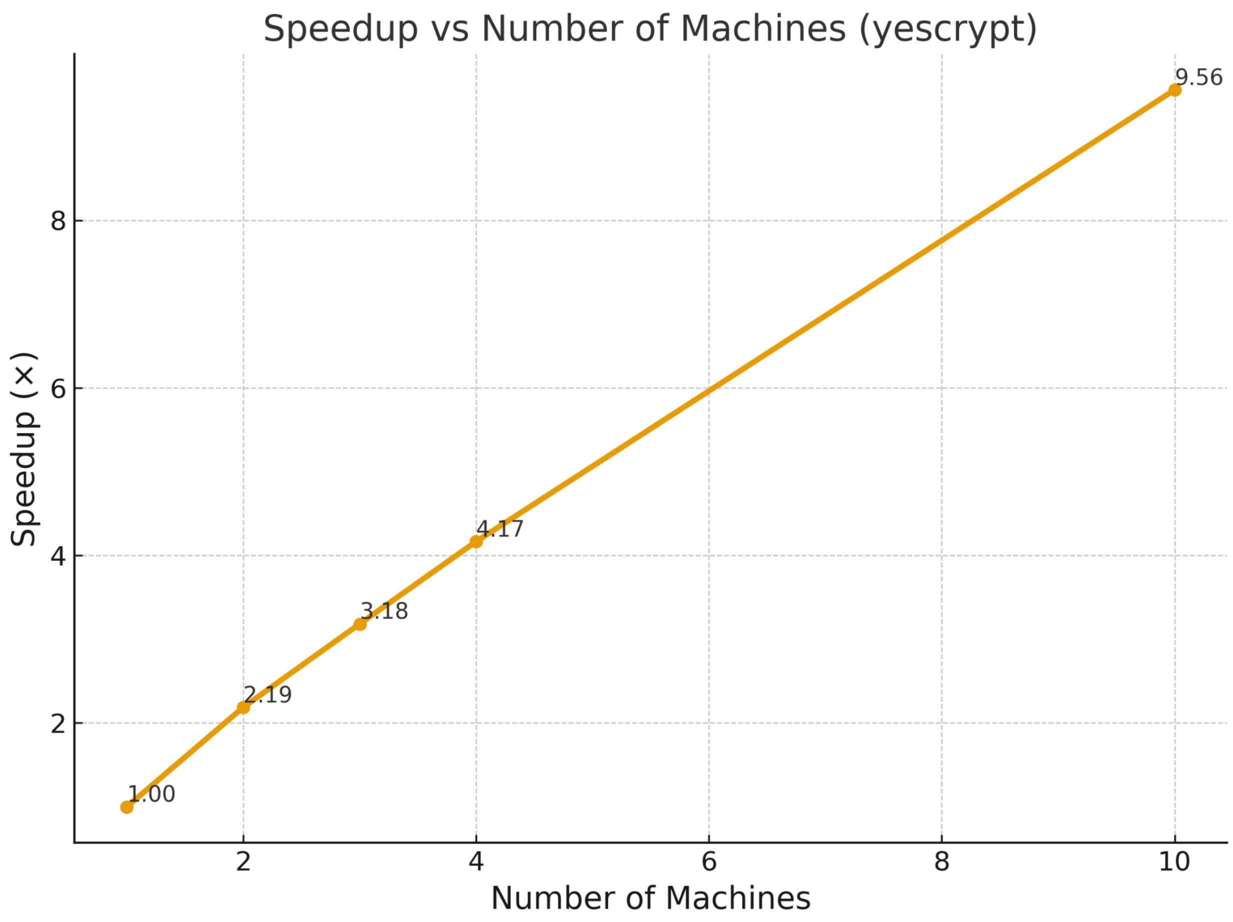
SHA 512


Speedup vs Number of Machines (sha512)
Speedup (×)
Number of Machines
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
2
4
6
8
10
1.00
1.90
2.62
3.40
4.86

yescrypt



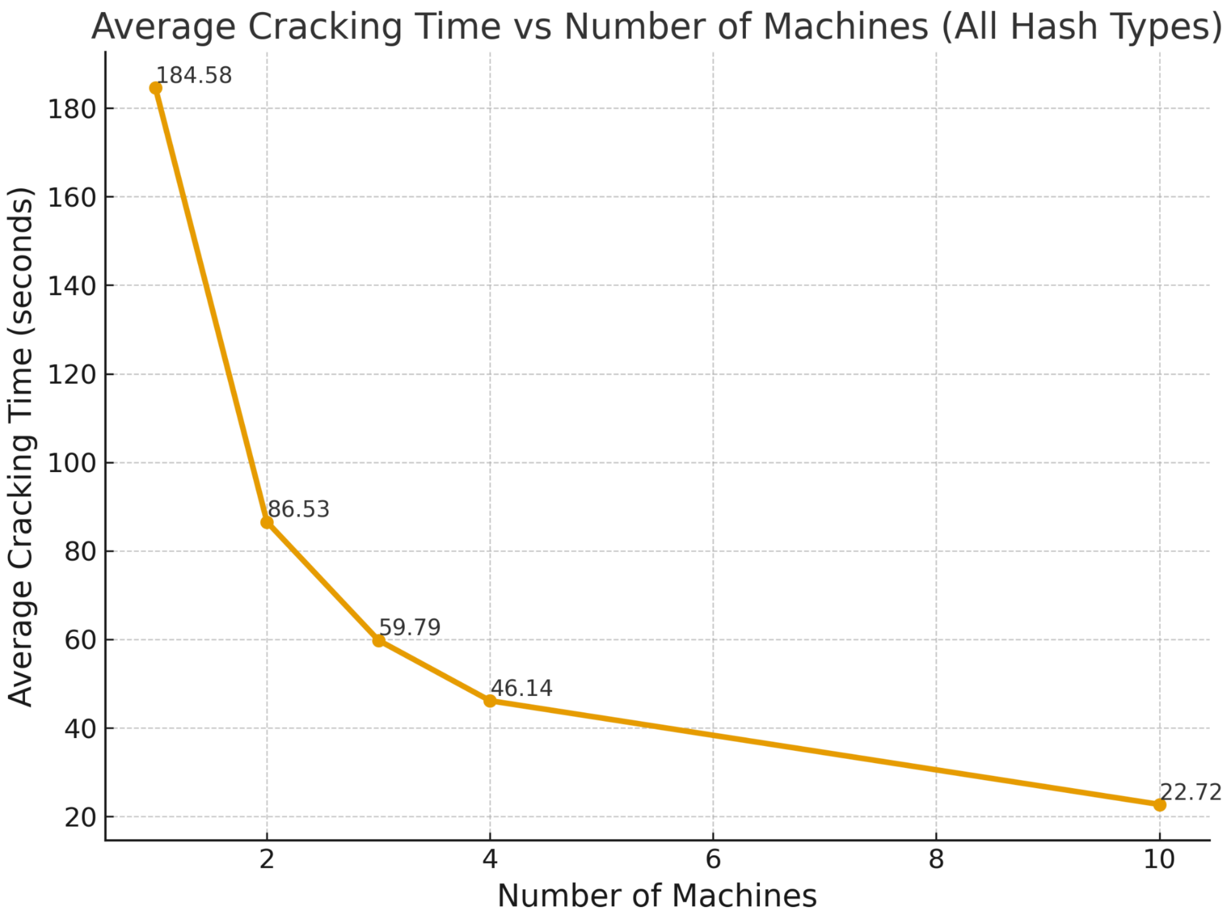
Cracking Time vs Number of Machines (yescrypt)
Cracking Time (seconds)
Number of Machines
464.69
212.44
145.99
111.54
48.62
100
200
300
400
2
4
6
8
10

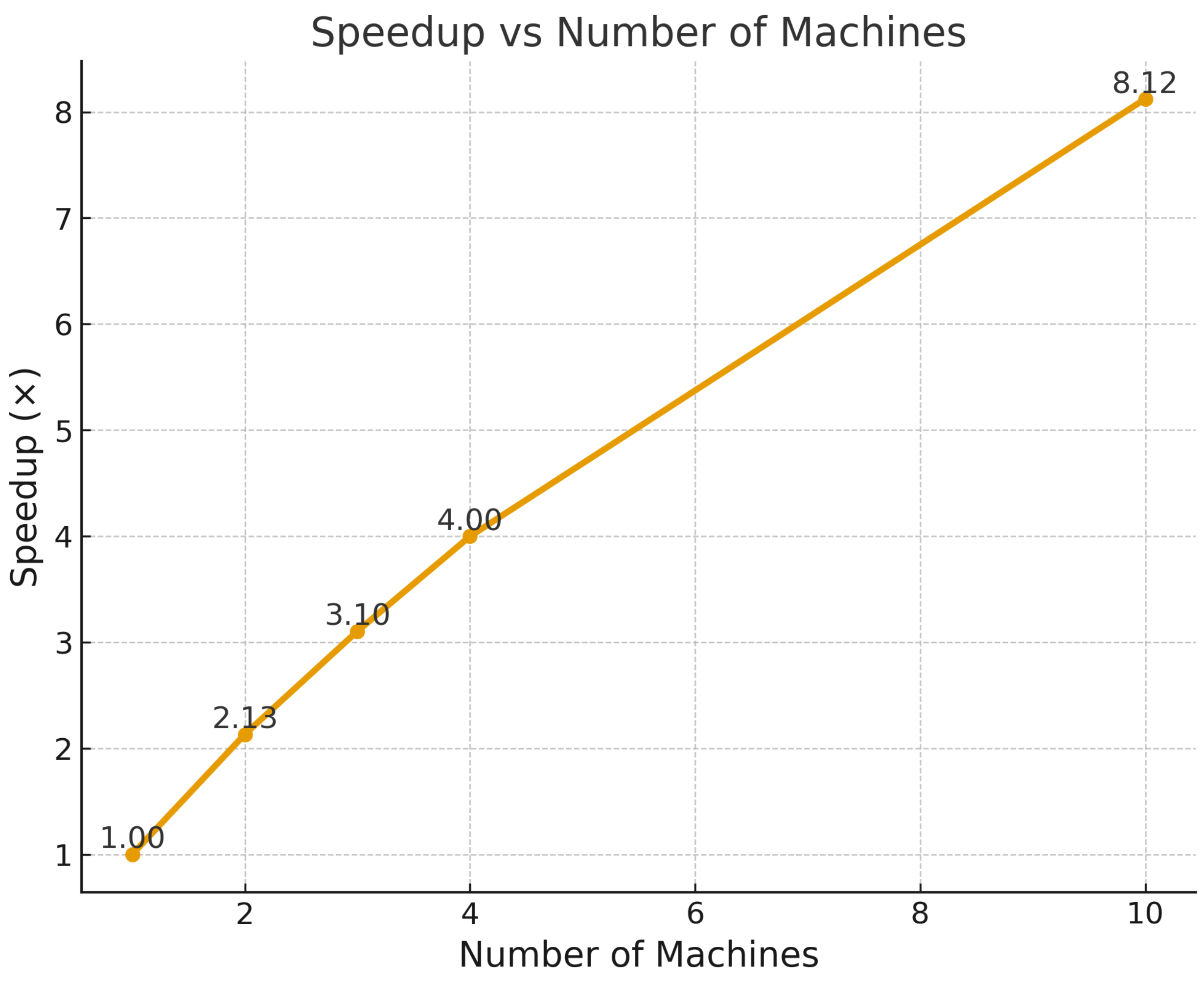

Speedup vs Number of Machines (yescrypt)
Speedup (×)
Number of Machines
1.00
2.19
3.18
4.17
9.56
2
4
6
8
10

Analysis

The results show that increasing the number of machines in the distributed system consistently reduces the wall-clock time required to crack a password for all hashing algorithms. The most significant improvements occur when scaling from 1 to 4 machines, where each additional worker provides a large reduction in total cracking time. At 10 machines the speedup continues to improve, but at a slower rate, indicating diminishing returns due to distributed system overhead.

Number of Machines	Wall Time (s)	Wall Speedup (×)
1	184.5766667	1
2	86.5266667	2.132
3	59.4566667	3.104
4	46.1433333	3.998
10	22.7233333	8.123


Average Cracking Time vs Number of Machines (All Hash Types)
Average Cracking Time (seconds)
Number of Machines
184.58
86.53
59.79
46.14
22.72
20
40
60
80
100
120
140
160
180
2
4
6
8
10



Yescrypt is almost a linear speedup because it is computationally expensive, indicating that it would be more cpu bound rather than io bound. Each password attempt takes longer to compute so the most cost is the pure hashing time rather than the communication over the network. That's why adding more machines increases the total compute power.

In contrast sha256 and sha512 are much faster algorithms, they are io bound rather than cpu bound. Which means that since the hashing operation is fast, most of the cost is in the communication over the network. So that's why the speedup percentage slows down when more machines are involved; As more machines are added, there are more I/O operations making the speedup graph for these algorithms flatten.

To speed up the more I/O bound hashes, I would increase the work size and the checkpoint because that would mean there would be less communication and network congestion. Also workers would not have to wait for the server to assign them more work.

Overall, the data confirms the hypothesis: adding more machines reduces total cracking time, but the speedup is not perfectly linear, and diminishing returns appear as distributed overhead increases. The system performs efficiently up to around 4 machines, with noticeable performance benefits even at 10 machines, though with reduced efficiency.

Hash	Number of Machines	Threads	Time (s)	Speedup
yescrypt	1	4	464.69	1
yescrypt	2	4	212.44	2.19
yescrypt	3	4	145.99	3.18
yescrypt	4	4	111.54	4.17
yescrypt	10	4	48.62	9.56
sha256	1	4	37.76	1
sha256	2	4	20.15	1.87
sha256	3	4	13.8	2.74
sha256	4	4	11.81	3.2
sha256	10	4	9	4.2
sha512	1	4	51.28	1
sha512	2	4	26.99	1.9
sha512	3	4	19.58	2.62
sha512	4	4	15.08	3.4
sha512	10	4	10.55	4.86