Comparing Centralized and Decentralized Distributed Execution Systems

Mustafa Paksoy
mpaksoy1@swarthmore.edu

Javier Prado
jprado1@swarthmore.edu

May 12, 2006

Abstract

We implement two distributed execution systems one centralized and one decentralized. We test these systems in terms of how well they handle a high number of requests, distribute execution, and tolerate heterogeneity. Given the small size and homogeneity of our lab environment we find that the centralized system provides a much simpler yet equally effective alternative to the decentralized system. However, in the simulated environments, we find that the decentralized system has great potential to tolerate heterogeneity and scale to wider area networks. This system needs to be tested with many more nodes in order to effectively test its ability to scale.

1 Introduction

Computation resources on most networks are highly underutilized. At the same time peak usage of individual workstations usually exceeds the workstations' capabilities. It makes sense to move part of this workload to unused workstations and keep used workstations uncluttered.

One way to deal with this problem is by distributed execution. By executing processes on different workstations, we can reduce the load on heavily used workstations and take advantage of underutilized workstations. More specifically, if a user has a job that requires the execution of several programs, an example being an experiment that needs to be run several times, that user could potentially distribute his programs within a network of workstations and considerably cut down on the time it would take the user to complete his job.

As with most tasks in distributed systems, distributed execution can be managed in either a centralized or a decentralized manner. In this paper we compare two distributed execution systems, one being centralized and one being decentralized, and evaluate their performance in different tasks.

The centralized solution is built around a central server that keeps track of registered hosts and distributes them using the round robin algorithm. Hosts and clients talk to the server over TCP making requests and receiving replies.

Our second solution is a system that distributes daemons on a group of workstations. These daemons communicate via UDP broadcasts. A client application communicates with the local daemon through IPC without having to go over the network. This system is distributed in nature because there is no central authority directing the execution of jobs; the decision making process is done by each workstation with the help of its neighbors on the network.

Both the centralized system and the decentralized systems will take advantage of the shared network file system that is available in our laboratory. Our systems run processes remotely using `ssh`. The centralized system distributes execution to workstations in a round-robin fashion. The decentralized system distributes execution of jobs using a least-loaded-workstation heuristic.

2 Related Work

Each of the research papers we have reviewed have implemented some sort of solution to the problem of underutilized workstations. In most cases each solution addresses a network of workstations (NOW) problem and attempts to resolve it.

- **Condor:**
  In the Condor system, the authors propose to use a hybrid of job scheduling coupled with a remote execution program. The authors use this combination of approaches because they want a system that is transparent to the users, and is not a burden upon the users of this system. The main focus of the Condor solution is to have a system where users who want to execute long background jobs could do so at the expense of using underutilized workstations. [1]

- **Butler System:**
  A similar solution to the underutilized workstation problem is presented by [3]. Here, the Butler system makes use of idle workstations in a network of workstations environment. In this Butler system only one remote job can execute per remote machine, although the number of remote jobs is not limited with respect to the user distributing these remote jobs. A distinguishing feature of this solution is that the network of workstations is implemented with a shared file system, which is very similar to our network.

- **Algorithm analysis:**
  Finally there is a proposed solution by [2], where only the gathering of data about a network of workstations is done in order to develop an algorithm for scheduling remote processes. We must be clear that one of the research areas of this paper concerns a distributed job across multiple workstations which we are not attempting to emulate. While this paper does consider parallel jobs, it also examines the general issue of determining how long a given workstation is going to be left idle by its user. From this analysis,

the author presents an ability to make predictions about the capacity of the idle systems from the manager's point of view allowing potential clients of this solution to predict, based on the client's knowledge of its computing demands, whether or not its job can be completed before the remote workstation is claimed by its remote user.

3 Design

3.1 Centralized Server

The centralized system is designed around a central workstation that maintains a list of hosts. Hosts are workstations which are able to accept jobs from neighboring workstations on the network. The central workstation accepts requests from potential hosts and registers them within its table of hosts. After a few hosts have been registered, any workstation that needs to run a remote job can simply query the central workstation and receive an address of one of the previously registered hosts. As a subtle point of reference, the client issuing the request for the use of a host does not have to be a registered user.

The way our workstations actually communicate with the central server is through the use of a TCP connection. After a connection has been established, communication is relatively straightforward: a short message is sent to the central server indicating the incoming request as either: register a workstation, get a workstation's address, or unregister a workstation. Once the central server has completed the request of the workstation, it returns a message back to the workstation indicating if the requested action was successful, with an accompanying IP address if the request was for getting a host.

The workstation requesting an IP address executes a `ssh` session with the remote workstation's IP address and executes the job remotely. A remote execution is possible for our system because we have a shared network file system. All the workstation has to provide is a path to the program to be run remotely when running the `ssh` command. Since the job is run over a shared network file system, all standard input/output is the same as if it was run locally. See Figure 1.

3.2 Decentralized System

The second system we implement is completely decentralized. All hosts in the system run daemon processes that periodically broadcast state and listen for broadcasts. In other words this is a Peer To Peer (P2P) system. Clients talk to local daemons using purely local inter-process communication.

Communication amongst daemons is asynchronous. Daemons broadcast system state periodically over UDP. A thread in each daemon listens for these broadcasts and keeps track of hosts in a specialized data structure.

We also implement a load balancing algorithm with the decentralized system. The daemons keep track of percent CPU usage, as well as total and available

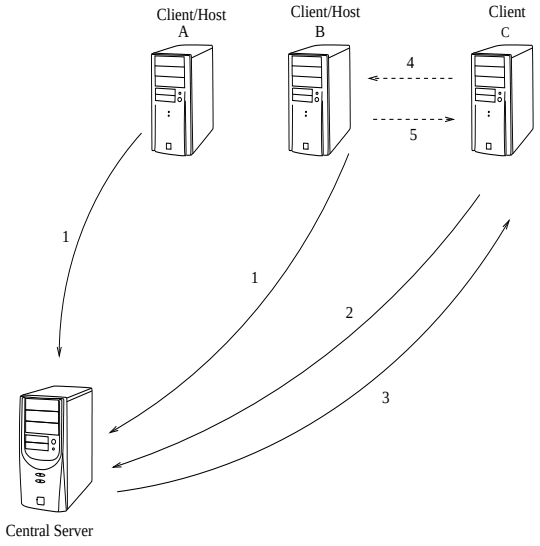

Figure 1: Figure of a simplified centralized system. (1) Hosts A and B register with the server. (2) Client C requests a host from the central server. (3) The server sends a host's IP address to the requesting client. (4) The client executes its job remotely using `ssh` and the host's IP address. (5)The client receives the host's return.

memory sizes. Individual daemons can prioritize these variables as they wish, the load balancing is done purely locally on each daemon.

The daemon maintains a fixed size table of hosts. When a host needs to be added but the table is full, another host needs to be replaced. The daemon uses a least-recently-updated host replacement policy.

Once again we rely on `ssh` and a shared network file system for secure remote execution. Once a client application receives a host from the daemon it `ssh`es to that machine to execute the given code. We have two small programs for interfacing with the daemon. One of these programs is written in C and the other is a shell script. They write a request to an appropriately named pipe and read a host from another named pipe. A thread in the daemon listens to these pipes and returns hosts upon request. See Figure 2 for a description of decentralized system's operation.

3.2.1 Super Queue

As previously stated, each daemon maintains a copy of global state. As such, these daemons need to simultaneously order hosts by the time they have been last updated and by their load. Instead of implementing this functionality using

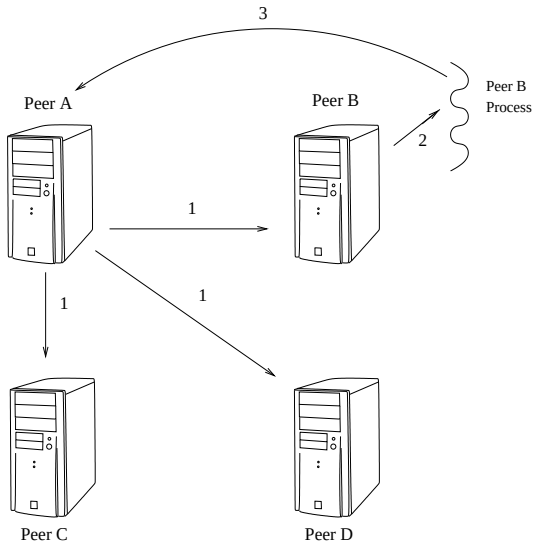

Figure 2: Figure of a simplified decentralized system. (1) Daemon on peer A broadcasts its cpu and memory state to its neighbors. (2) Daemon on peer B, having updated its superQ, determines that Peer A should be the host for the next remote execution. Daemon replies to the requesting process with peer A's address. (3) Peer B connects to Peer A using `ssh` and executes its job.

two separate data structures, we superimpose a priority queue and a FIFO queue on each other. We call the resulting data structure a Super Queue (SQ).

Nodes are placed on a fixed size table in no particular order. These nodes contain two separate sets of pointers, each set implements two separate doubly linked lists. One of these doubly linked lists implements a FIFO queue, the other a priority queue. We chose a doubly linked list because it allows very quick node adding and removal operations. Given a moderately sized group of workstations, a high number of broadcast messages will be going out over the network. A doubly linked list minimizes the load induced by such high turnover. See Figure 3.

4 Experiments

Our main purpose in testing is to establish a basis for comparison between the two systems. We compared the performance of the two systems in several aspects.

1. **Server response time:**
   In this experiment we load a fixed number of hosts on the centralized and

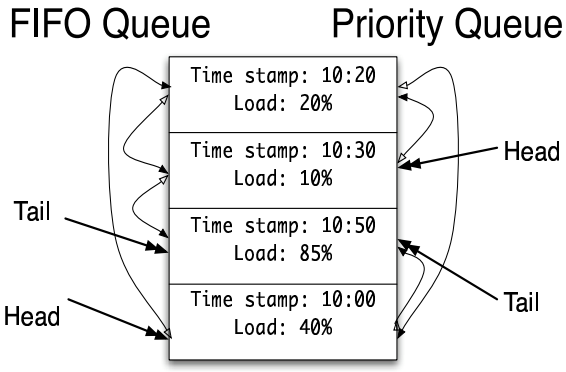

Figure 3: Super Queue structure.

decentralized systems and compare their performance when swamped with a high number of requests. For the centralized system, we also observe the effects of placing the server on a different subnet than the client.

2. **Distributing jobs to homogeneous hosts:**
   Here we test sending out varying numbers of CPU bound jobs to hosts using the two different systems. The hosts in these systems are very similar. This is a basic measure of the effectiveness of the system in distributing execution in order to minimize execution time.

3. **Distributing jobs to heteregeneous hosts:**
   This is the same as distributing jobs to homogeneous hosts except that this time we place CPU bound loads on half of the nodes. The idea here is to simulate the effect of load heterogeneity. Since our decentralized solution includes a simple load balancing mechanism, this will also show us if it has any significant effect.

5 Results and Discussion

In our regular lab environment the decentralized system performed slightly worse than the centralized system. We believe this is primarily due to the small size of our lab environment. However, the decentralized system's advantages in load balancing and response time could be observed in simulated environments. This leads us to believe that the decentralized system needs to be thoroughly examined using a larger scale network.

5.1 Server and Daemon Response

When the centralized server and client are in the same subnet (the Computer Science subnet in this case) response time for the P2P daemon is very close to the response time for the centralized server (Figure 4). The network overhead in the centralized system is not observable. There is no strong trend showing either system responding faster.

However, when we placed the centralized server on a different subnet (the SCCS subnet in this case) the daemon performed observably better than the centralized server. This suggests that with larger scale systems network overhead will be a bigger issue. (Figure 4.)

We run all our experiments using shell scripts and `cron`. It is very likely that shell script overhead makes subtle performance differences between the two applications unnoticeable. In one instance, when we switched from using a shell script to using a C program to interface with the daemon, our response time was reduced by a factor of four. In any case, we need to test these applications on a much larger system to effectively observe the differences in the scalability of each.

5.2 Homogeneous Distribution

In order to judge the actual performance of our systems we tested how well they performed in distributing the execution of a given set of jobs. In each case 16 hosts are registered on the system. The client workstation seeding the jobs sends out jobs in powers of 2 up to 128. The total time for each batch is recorded.

Data suggests (Figure 5.) that the centralized solution actually performs (and scales) *better* than the decentralized solution in this case. Given the sophistication and load balancing capabilities of the decentralized solution, this is unexpected. As previously stated we believe this is because of the particular character of our lab environment.

5.3 Heterogeneous Distribution

The two systems perform very similarly in a heterogeneous environment (Figure 6). The decentralized system performs better given a smaller number of hosts. The daemon for the decentralized system marks a host as having full load once it assigns a job to it. This is to ensure that jobs are distributed as much as possible while still being sent to the least loaded hosts. So once a daemon sends out jobs to all known hosts, it will simply distribute jobs in a round robin fashion until it receives new state information about hosts.

When swamped, the daemon will quickly mark many of its nodes as having full load. If the number of requests sent is less than the number of known hosts, the daemon will adhere to its least-loaded-first policy. However, when the number of requests exceeds the number of known hosts, the daemon will quickly mark all hosts as having full load. As such, it can only use load state information

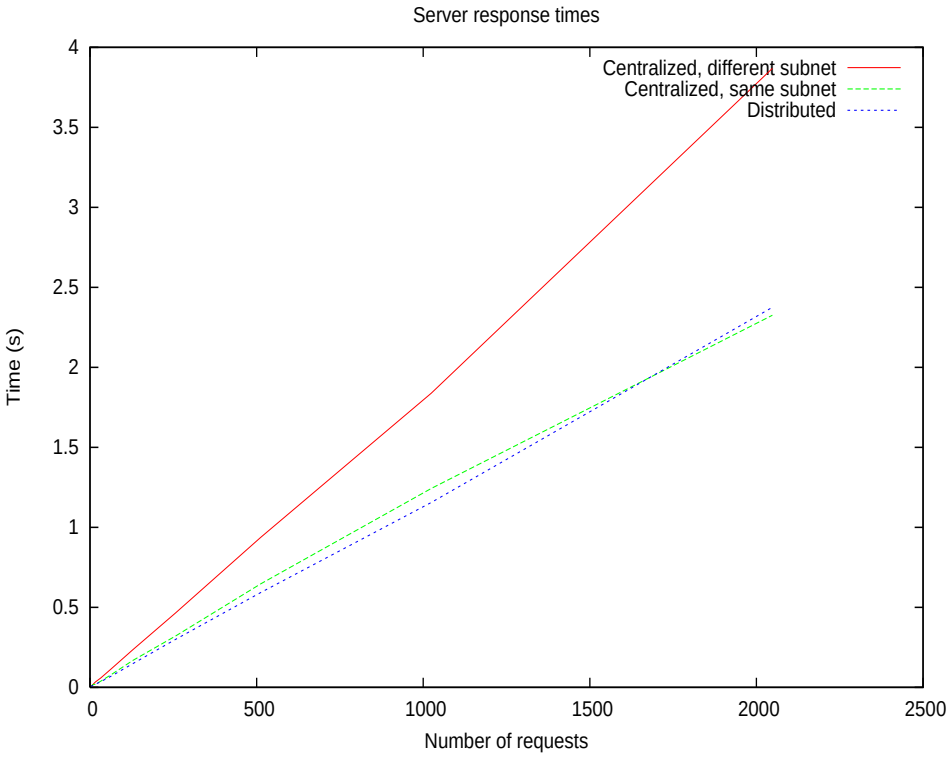

Figure 4: Server response graph.

if it receives a broadcast right in between the requests. Daemons broadcast every second, so a daemon will be able to make intermittent informed decisions when swamped. As such, the load balancing algorithm works best when requests come in small bursts. It still provides a smaller advantage otherwise.

5.4 Homogeneous vs Heterogeneous Performance

Heterogeneity impacts the centralized and decentralized systems differently. Both systems become slower due to half of the nodes becoming heavily loaded, however the impact on the decentralized system is considerably less than the centralized system. We can see on Figure 7 that the centralized system is consistently made slower by heterogeneity. This is as expected; the round robin scheduling algorithm does not respond to load on certain nodes in any way.

On the other hand we can see in Figure 8 that the decentralized system, with its load balancing algorithm, tolerates heterogeneity much more gracefully. The

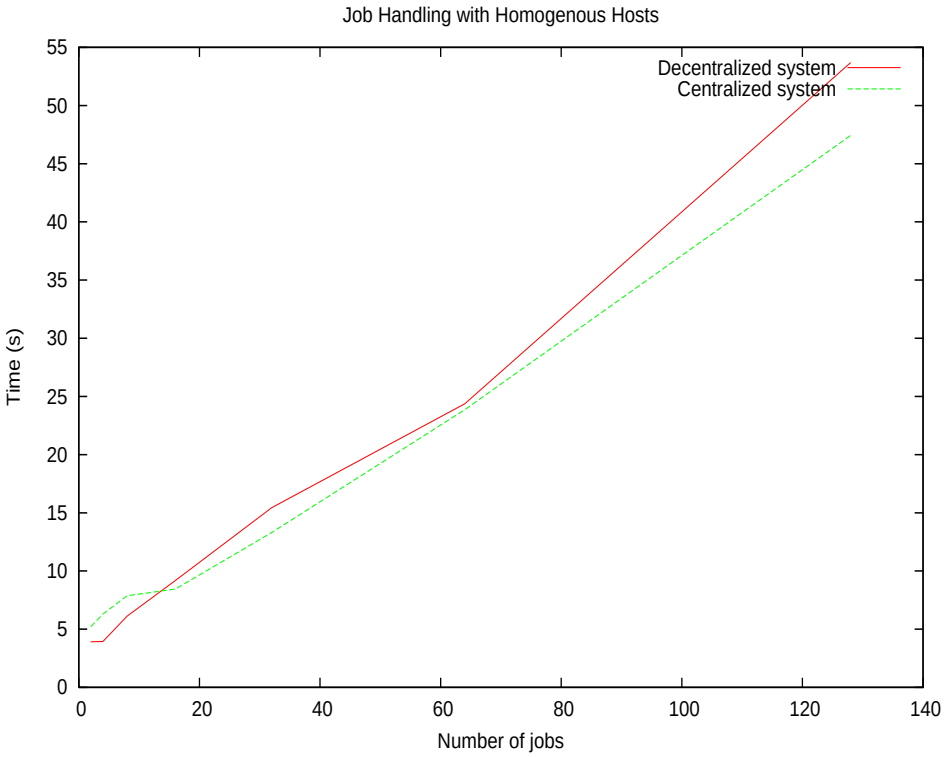

Figure 5: Homogeneous response graph.

gap between the performance of the heteregeneous and homogeneous systems is actually decreasing with the number of jobs seeded.

6 Conclusion

Given the small number of hosts we tested this system on, it is understandable that we did not make considerable gains from a decentralized approach. On the whole, our lab environment has a fast network and is generally homogeneous. As the date suggests, the gains made from a decentralized solution are hard to observe on such a system.

However, we can still see the advantages of our decentralized solution in the simulated environments we created. The load balancing algorithm of our decentralized solution is clearly superior in tolerating heterogeneity. Furthermore, once outside the immediate CS network, the costs of going over the network make local host lookup more advantageous.

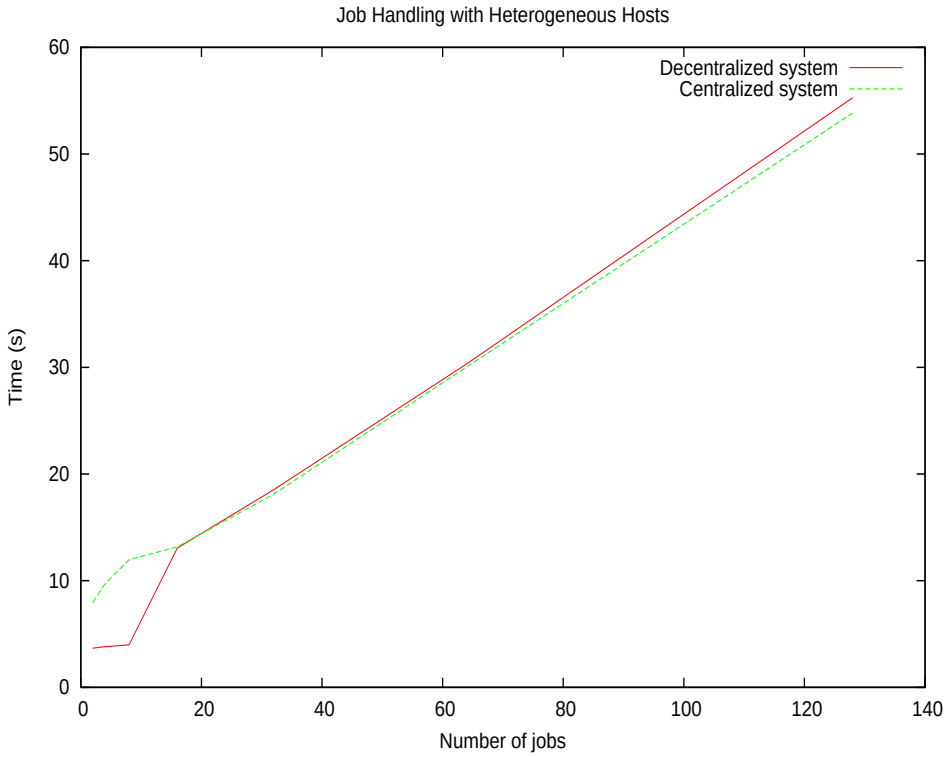

Figure 6: Heterogeneous response graph.

The effectiveness of an application is constrained by the demands of the environment. Even though our decentralized solution was more sophisticated than our centralized solution on many counts, it was not well suited for such a tame environment.

7 Future Work

For both systems, in a batch of 128 jobs with total execution time of around 50 seconds, time spent getting hosts accounted for around 0.3 seconds. This suggests that more time can be spent trying to develop more elaborate load balancing schemes. Profiling nodes in the system before or during execution is a viable option that would further guide load balancing algorithms.

Another option is developing a more powerful remote execution system. `ssh` is not the ideal tool for distributed execution. It provides functionality such as transferring terminal i/o over the network, secure authentication, and encryp-

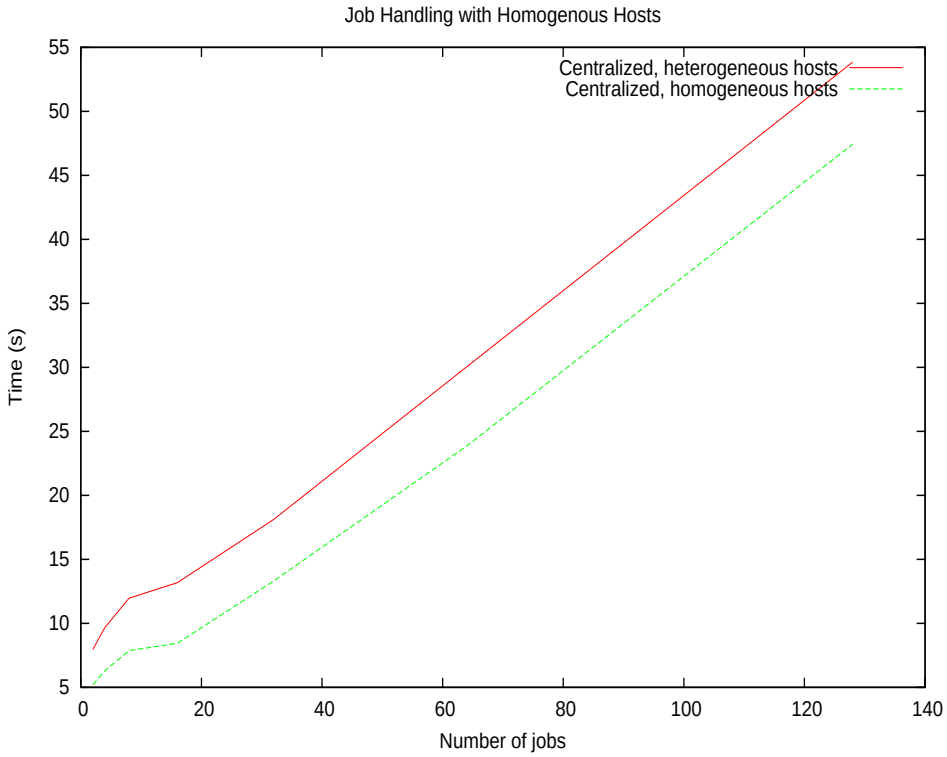

Figure 7: Heterogeneous response graph for our centralized solution.

tion. A remote execution mechanism tailored for distributed execution would greatly streamline communication over the network. Such a system would also free us from the requirement for a shared network file system, which is greatly limiting.

8 Acknowledgements

We based our UDP broadcaster and listener code on Beej's samples located at http://www.ecst.csuchico.edu/ beej/guide/net_old/. We thank Beej for his extremely helpful tutorial on network programming.

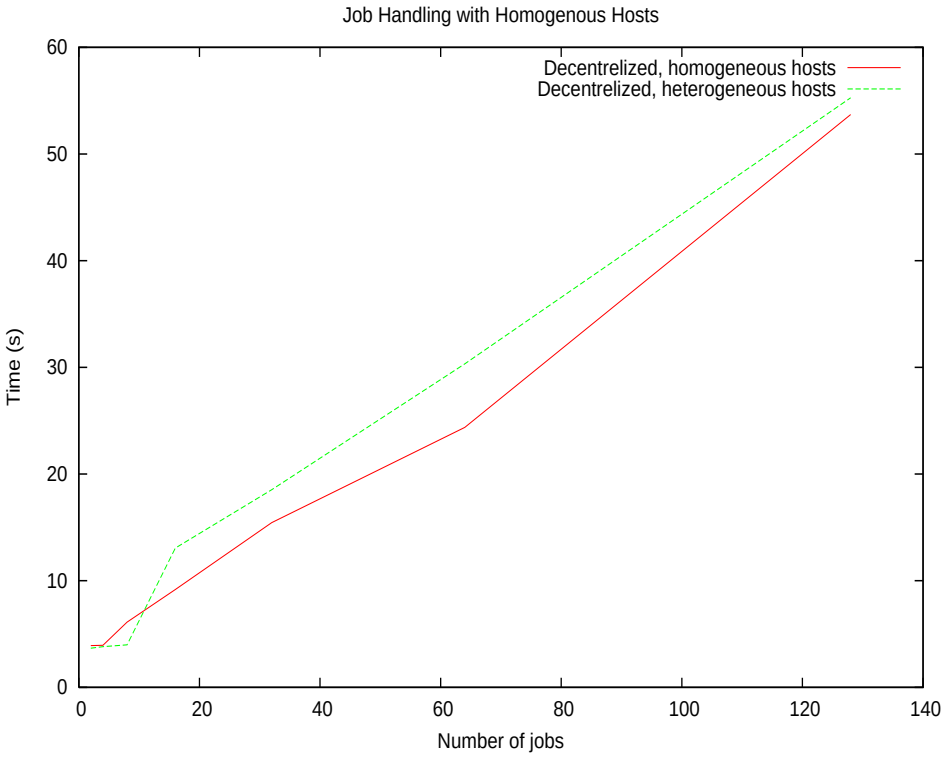

Figure 8: Heterogeneous response graph for our decentralized solution.

References

[1] Michael Litzkow. Condor—a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, 1988.

[2] Matt W. Mutka. An examination of strategies for estimating capacity to share among private workstations. In *SIGSMALL '91: Proceedings of the 1991 ACM SIGSMALL/PC symposium on Small systems*, pages 41–49, New York, NY, USA, 1991. ACM Press.

[3] D. Nichols. Using idle workstations in a shared computing environment. In *SOSP '87: Proceedings of the eleventh ACM Symposium on Operating systems principles*, pages 5–12, New York, NY, USA, 1987. ACM Press.