P2P-LLM: Collaborative LLM Inference

Andrew Boessen BOESSENA@BC.EDU

Abstract

1 Introduction

In recent years, large language models (LLMs) have demonstrated remarkable capabilities across diverse tasks, exhibiting unprecedented rates of improvement within a relatively brief developmental timeline. This extraordinary progress can be primarily attributed to the immense computational resources allocated to training these models on datasets comprising trillions of tokens, resulting in systems that subsequently demand substantial computing power for deployment. The computational intensity of these models has resulted in significant investment, with billions of dollars directed toward the construction of specialized super-clusters housing hundreds of thousands of GPUs dedicated to the training and serving of LLMs. Given the substantial memory and computational requirements of state-of-the-art LLMs, most personal computing devices lack the necessary specifications to run or train these models independently. Consequently, access to LLM capabilities is predominantly channeled through services offered by large technology corporations, which may present concerns regarding privacy, autonomy, and accessibility.

Crowdsourcing methodologies offer a promising alternative to address these limitations. The crowdsourcing approach has demonstrated considerable success across various domains, including knowledge aggregation (Wikipedia), real-time traffic information collection (Waze), and venture capital formation (Kickstarter). These same foundational principles can be effectively applied to computational resource allocation and utilization.

My work explores fully decentralized crowd computing architectures, where users establish a peer-to-peer network in which each participant can leverage the collective computational resources for individual tasks. This structure shares conceptual similarities with BitTorrent file-sharing networks. While current state-of-the-art language models exceed the capacity of individual personal computers, the aggregation of distributed computational resources through a decentralized network could potentially enable users to host their own LLMs, thereby democratizing access to advanced artificial intelligence capabilities.

2 Realted Work

2.1 Crowd Computing

The idea of pooling compute power from many computers to form a distributed supercomputer is not new. This concept of Crowd Computing has been used ever since large networks of computer were formed. The predominant use of this method is to fascilitate academic research. Researchers publish problem that that are working on, and anyone can help contribute by volunteering to donate computing power. Projects that have successfully used

crowd computing include SETI@Home (Anderson et al., 2002) and Einstein@Home (Steltner et al., 2021). This approach has proven to be successful in the past, but it has become less popular as the computational power of computers increased and access to supercomputers increased. However, the use of idle compute power still has untapped applications, and the principles from crowd computing can be applied to modern tasks like LLM inference.

2.2 P2P Networks

LLM inference is a perfect candidate for P2P network because it has technical requirments that prohibit most personal computers from being able to run large models with billions of parameters, and it can easilly be split into small chunks that can be run on several computers at the same time. BitWorker (Durand et al., 2015) is a protocol designed to compute any parallel computing task across a peer-to-peer network. Although it is possible to apply LLM inference to this protocol, BitWorker is designed for general tasks and would not able to efficiently scale to the demands of many users. The most notable research being done in this area is Petals (Borzunov et al., 2023) which is a BitTorrent like peer-to-peer network for LLM inference and finetuning. Petals creates a network with client and servers. Clients send requests that will be processed by the servers on the network. The model is split into multiple sequential blocks of layers, and each server in the network is capable of serving one or multiple of these blocks. In order to run an inference step, a client will find a optimal path of servers in the network thats forms the complete model and minimizes time required. In my work I explore a similar approach with a network of distributed computers that serve a specific part of the whole LLM, but I expand upon this and add incentive mechanisms that reward nodes for donating their computational power. This incentive mechanism also fucilitates optimal allocation of nodes to maximize throughput, by each node maximizing expected revenue individually.

3 Peer-to-Peer Network

Fundamentally, a peer-to-peer network is a decentralized system where each devices acts as a client and a sever. Each devices can send and recieve data which eliminates the need for a centralized server. The central benefit of a P2P network is that recources are pooled and shared between peers, so the combined resources available to an individual peer is greater than that nodes own resources. A peer-to-peer network can be built to share any type of resource. In the case of BitTorrent, bandwidth and files are shared between peers. For the case of LLM inference, peers will need to ba able to share bandwidth, data, and computational power.

A peer-to-peer network is defined as having n nodes in the network. There is an LLM model with m total layers which will be devided between nodes. Each node N in the network is a tuple $N(\ell, l, c, p)$ where

- $\ell_i = (x, y) : x, y \in [0, m); y > x$ is the range of layers the node is able to serve
- $l_{ij} \in \mathbb{R}$ is the network latency between nodes i and j,
- $c_i \in \mathbb{R}$ is the computational cost of the node, measured in miliseconds
- $p_i \in \mathbb{R}$ is the price the nodes charges for its compute.

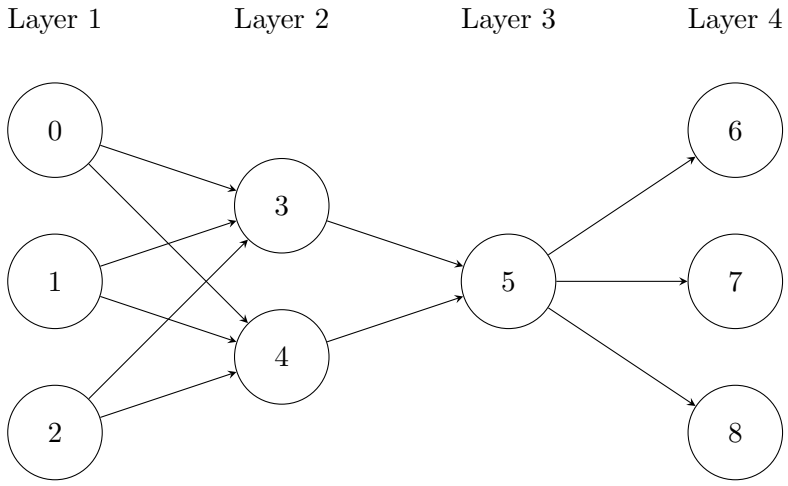

Figure 1: P2P-LLM Network Directed Graph

3.1 Path Finding

In order to use the network to run the LLM, a node must first find a sequence of nodes in the graph that form a complete model. Because the LLM layers must be computed sequentially, the node must find a path through the network that will ideally compute the LLM in the least amount of time possible and at the lowest price. This problem can be solved by viewing the network as a directed graph where weights correspond to the computational cost of nodes. The edges of the graph correspond the compatability between the layers that each node in serving. Specifically, every node is connected to all nodes that are serving the consecutive layer of the model. A vizualization of this directed graph is shown in Figure 1.

Due to this property, the P2P network graph is a directed acyclical graph (DAG). This means that there are efficient algorithms to find the shortest path between two nodes. The first step is to perform a topological sort on the nodes in the network. This is an ordering of the nodes such that all neighbors of a node come after the node in the ordering. For any graph $G = (V, E)$ a totological ordering is an order $\prec$ on V such that if $(u \to v) \in E$ then $u \prec v$. For example, a topological ordering of the DAG in Figure 1 is $[0, 1, 2, 3, 4, 5, 6, 7, 8]$. The algorithm for topological sort is shown in Appendix A. With the topological order, dynamic programming can be used to find the shortest path in the network. This algorithm will iterate over the ordering, and search all the node's neighbors for the one with the lowest weight. The full algorithm is shown in Appendix B. Most importantly, both the topological sort and path finding algorithm run in linear time with a computational complexity of $O(|V| + |E|)$.

3.2 Pareto Effeciency

The optimal path is well defined when weights for nodes are clearly defined, but there is not a clear way to define weights for nodes in this case. In the P2P-LLM network there are two competing criterea that define the weight of node: computational cost and price.

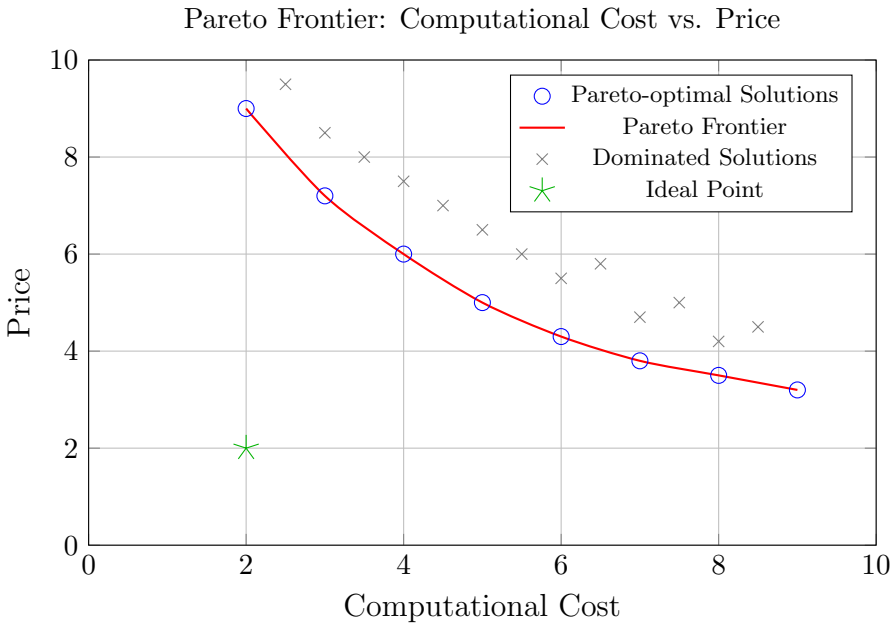

Figure 2: Pareto Frontier

The algorithm described above uses a single objective (SO), while the optimal path in the network is a multi-objective (MO) optimization problem. There are several way to find optimal solutions to MO optimization problems (Ngatchou et al., 2005). The simplest way is to convert it to a SO optimization problem, and since the shortest path algorithm is designed for SO objective, it is the most practical solution for this task. In order to reframe the problem as a single objective, a weighed aggreagte of the objetives is used. In the case for the peer-to-peer network, the SO is a weighed sum of computational cost c_i and price p_i. The objective, or weight, of a node, w_i is defined as.

$$w_i = \alpha \cdot p_i + (1 - \alpha) \cdot c_i \tag{1}$$

However, this assumes that the weight α is known a priori. In general, there are many equivilent optimal solutions in a MO optimzation problem. These are known and Pareto-optimal solutions. They are solutions such that there is no way to improve one objective without degrading any other objectives. A solution is said to be Pareto-optimal if there are no other solutions that dominate it. A solution x_1 dominates x_2 if

1. $\forall i, f_i(x_1) \leq f_i(x_2)$ and
2. $\exists i, f_i(x_1) < f_i(x_2)$

where $f_i()$ is the utility for the ith objective.

The set of all Pareto-optimal solutions forms a frontier (Figure 2) of equivilent solutions, but there is not a clear choice on which one should be used. In the case of the SO weighted aggreagte used in the P2P-LLM network, the parameter α decides where on the Pareto frontier the optimal solution lies. Due to this, each node can independently decide that importance of price and computational cost in the optimal path through the network.

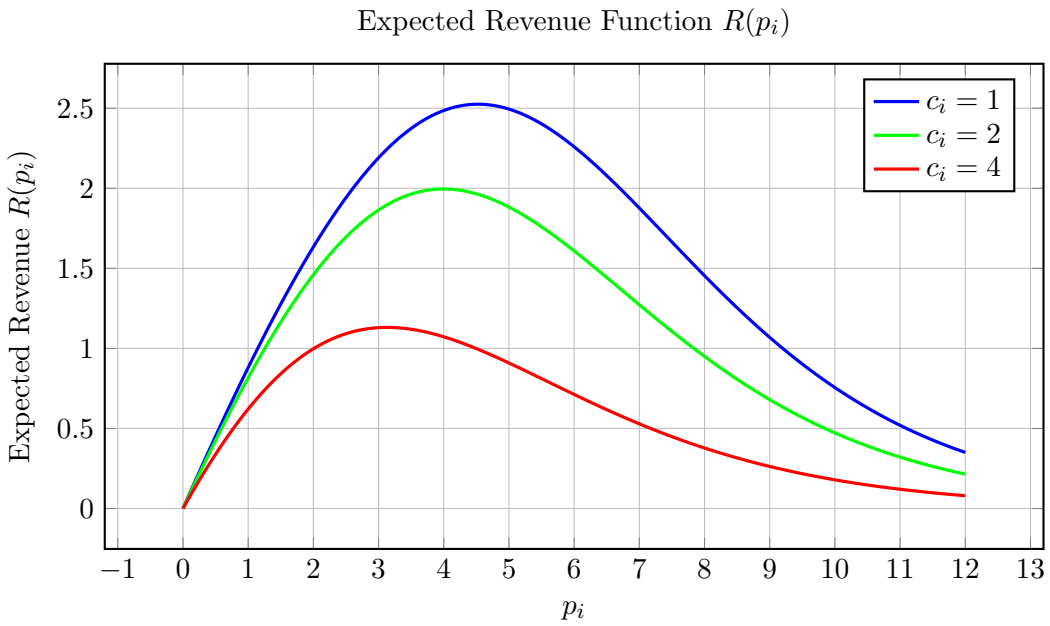

Figure 3: Effect of Computational Cost on Expected Revenue

3.3 Revenue Optimization

Each peer additionally has to choose a price to charge for other peers to use their computation. Ideally each node will act in its self-iterest and choose the price that maximizes revenue. If the price is too high, peers will choose equivlent node serving the same layers, while if the price is too low, the total revenue will be less than ideal. The price they choose should reflect the current state of the network, and adapt to conditions. In this context, price optimization become difficult as it is a dynamic environment and a uniform pricing rule would not result in optimal revenue. In the P2P-LLM network, a peer only has to compete with other peers serving the same layers. This can be exploited to narrow down the complexities of this problem. Working with this leads to the conclusion that a nodes price should be determined by its own price and computational cost and the price and computational cost of all other peers serving the same layers.

If we can define a revenue function based on these parameters, then the optimal price can be found by maximizing this function with respect to p_i. The revenue a node recieves for a single peer using the network would be its price multiplied by the probability of being chosen in the optimal path.

$$R(p_i) = p_i \cdot \mathbb{P}(w_i) \tag{2}$$

The probability that a node is chosen is related to its weight (Equation 1). In order to get a distribution across all weights in the node's layer, the Boltzmann distribution (Boltzmann, 1868) can be used. This distribution give the probability that a system will be in certain state.

$$\mathbb{P}(\epsilon_i) = \frac{e^{-\epsilon_i/kT}}{\sum_{j=1}^{M} e^{-\epsilon_j/kT}} \tag{3}$$

where ϵ_i is the energy for state i, k is the Boltzmann constant, and T is the temperature of the system. For the purposes of the P2P-LLM network, k and T are dropped and ϵ_i is w_i (Equation 1). Using this method gives a smooth expected revenue function, and prices will reflect how much each nodes contributes to the network. Specifcally, if a nodes contributes more compute power to the network, demonstrated as a lower computational cost c_i, the optimal price they will charge will be higher, and they will be rewarded with a higher expected revenue. A vizualization of how c_i effects the expected revenue curve is shown in Figure 3.

Now with the expected revnue function, the optimal price can be found by finding the price that maximizes $R(p_i)$. Unfortunately, there is not a closed form solution to this problem, but there are methods that can be used. Iterative pricing mechanism (Saari, 1985) are used to find optimal prices by making iterative changes to the price until it converages at an optimal point. These methods work well for dynamic environment where the state of the market is contiously changing. For example, gradient ascent has been used for forex investing (Murtza et al., 2022). The gradient ascent algorithm is very similar to the gradient descent (Cauchy, 1847) algorithm commonly used in machine learning. The key differnce is that it seeks to maximize a function's value rather minimize it. Gradient ascent is used to maximize the expected revenue function, $R(p_i)$ with respect to price. It is defined as

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \eta \nabla f(\mathbf{x}_t) \tag{4}$$

where $\mathbf{x}_t$ is the parameter vector at iteration t ,$\mathbf{x}_{t+1}$ is the updated parameter vector, η is the learning rate (step size) , and $\nabla f(\mathbf{x}_t)$ is the gradient of the objective function f at point $\mathbf{x}_t$.

Applying this to the revenue function, $R(p_i)$, we can get and update function for the price. The gradient of R with respect to p is

$$\frac{\partial R}{\partial p} = \mathbb{P}(w)(1 + \alpha p \cdot (\mathbb{P}(w) - 1))$$

Using this with the gradient ascent algorithm Equation 4, the price update rule is

$$p_{t+1} = p_t + \eta \frac{\partial R}{\partial p_t} \tag{5}$$

As all nodes continuly update their prices, they will converge on the price that gives maximum revenue for each node individually.

Appendix A. Topological Sort

Algorithm 1 Topological Sort of a Directed Acyclic Graph (DAG)

Require: A directed acyclic graph $G = (V, E)$

Ensure: A linear ordering of vertices such that for every directed edge (u, v), vertex u comes before vertex v

result[] ▷ An empty list to store the sorted vertices
visited[v] →false for all $v \in V$ ▷ A map to track visited vertices
temp[v] → false for all$v \in V$ ▷ A map to track vertices in the current recursion stack
for each vertex $v \in V$ **do**
    if visited[v] = false **then**
        DFS-Visit(G, v, visited, temp, result)
    end if
end for
return Reverse(result)
function DFS-Visit(G, u, visited, temp, result)
    temp[u] ← true ▷ Mark current vertex as being processed
    for each vertex v such that $(u, v) \in E$ **do**
        DFS-Visit(G, v, visited, temp, result)
    end for
    temp[u] ← false ▷ Mark u as processed
    visited[u] ← true ▷ Mark u as visited
    Append u to result
end function

Appendix B. Shortest Path

References

David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11): 56–61, November 2002. ISSN 0001-0782. doi: 10.1145/581571.581573. URL `https://doi.org/10.1145/581571.581573`.

Ludwig Boltzmann. Studien über das gleichgewicht der lebendigen kraft zwischen bewegten materiellen punkten. *Wiener Berichte*, 58:517–560, 1868. [Studies on the balance of living force between moving material points].

Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Max Ryabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, and Colin Raffel. Petals: Collaborative inference and fine-tuning of large models, 2023. URL `https://arxiv.org/abs/2209.01188`.

Augustin-Louis Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. *Comptes Rendus de l'Académie des Sciences*, 25:536–538, 1847. Early description of the gradient descent method.

Algorithm 2 Shortest Path in a DAG

Require: Graph $G = (V, E)$ with edge weights w, source vertex s, target vertex t, topological ordering $topo[]$ of G

Ensure: Length of shortest path from s to t and the path itself

```
Initialize dist[v] ← ∞ for all v ∈ V
Initialize prev[v] ← nil for all v ∈ V
dist[s] ← 0
for each vertex u in topological order topo[] do
    for each outgoing edge (u, v) ∈ E do
        if dist[v] > dist[u] + w(u, v) then
            dist[v] ← dist[u] + w(u, v)
            prev[v] ← u
        end if
    end for
end for
path ← ∅                                   ▷ Reconstruct the shortest path
curr ← t
while curr ≠ nil do
    Prepend curr to path
    curr ← prev[curr]
end while
return dist[t], path
```


Arnaud Durand, Mikael Gasparyan, Thomas Rouvinez, Imad Rafic Aad, Torsten Braun, and Tuan Anh Trinh. Bitworker, a decentralized distributed computing system based on bittorrent, May 2015. URL https://boris-portal.unibe.ch/handle/20.500.12422/134610.

Iqbal Murtza, Ayesha Saadia, Rabia Basri, Azhar Imran, Abdullah Almuhaimeed, and Abdulkareem Alzahrani. Forex investment optimization using instantaneous stochastic gradient ascent—formulation of an adaptive machine learning approach. *Sustainability*, 14(22), 2022. ISSN 2071-1050. doi: 10.3390/su142215328. URL https://www.mdpi.com/2071-1050/14/22/15328.

P. Ngatchou, A. Zarei, and A. El-Sharkawi. Pareto multi objective optimization. In *Proceedings of the 13th International Conference on, Intelligent Systems Application to Power Systems*, pages 84–91, 2005. doi: 10.1109/ISAP.2005.1599245.

Donald G. Saari. Iterative price mechanisms. *Econometrica*, 53(5):1117–1131, 1985. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1911014.

B. Steltner, M. A. Papa, H.-B. Eggenstein, B. Allen, V. Dergachev, R. Prix, B. Machenschalk, S. Walsh, S. J. Zhu, O. Behnke, and S. Kwang. Einstein@home all-sky search for continuous gravitational waves in ligo o2 public data. *The Astrophysical Journal*, 909(1):79, March 2021. ISSN 1538-4357. doi: 10.3847/1538-4357/abc7c9. URL http://dx.doi.org/10.3847/1538-4357/abc7c9.