## 1. Phát biểu bài toán

Trong học sâu, một bài toán cơ bản là huấn luyện mô hình mạng nơ-ron sâu để tối ưu hóa tham số mô hình nhằm giảm thiểu hàm mất mát (loss function). Bài toán này được biểu diễn như sau:

$$\theta^* = arg\min_{\theta} L(\theta)$$

Trong đó:

- $\theta$: Là các tham số của mô hình
- $L(\theta)$: Là hàm mất mát đo lường độ chênh lệch giữa dự đoán của mô hình và nhãn thực tế.

Thuật toán Stochastic Gradient Descent (SGD) thường được sử dụng để giải quyết bài toán trên bằng cách cập nhật các tham số dựa trên gradient của hàm mất mát theo công thức:



$$\theta \leftarrow \theta - \eta \cdot \nabla L(\theta)$$

Trong đó:

- η: Tốc độ học (learning rate).
- ∇L(θ): Gradient của hàm mất mát.

Thuật toán SGD (tuần tự) hoạt động theo các bước sau:

*Bước 1: Khởi tạo tham số ban đầu với giá trị ngẫu nhiên.*

*Bước 2: Lặp lại qua các epoch:*

- *Chia tập dữ liệu thành các batch nhỏ.*
- *Lặp qua từng batch:*
  *+ Tính gradient trên batch.*
  *+ Cập nhật*

*Bước 3: Kết thúc khi số epoch đạt ngưỡng hoặc hàm mất mát không còn cải thiện.*



Mã giả:

```
Initialize parameters θ
For epoch in range(max_epochs):
    For batch in dataset:
        Compute gradient: grad = ∇L(θ)
        Update parameters: θ = θ - η * grad
    End For
End For
```

Khi áp dụng thuật toán Stochastic Gradient Descent (SGD) để huấn luyện mô hình học sâu, việc tính toán gradient trên từng mini-batch giúp giảm tải so với Gradient Descent toàn bộ dữ liệu, nhưng vẫn đòi hỏi tài nguyên tính toán đáng kể. Đặc biệt, với các tập dữ liệu lớn và mô hình phức tạp, quá trình huấn luyện kéo dài do số lượng phép tính gradient tăng lên, làm chậm quá trình hội tụ và ảnh hưởng đến hiệu suất tổng thể.

Một máy tính đơn lẻ khó có thể đáp ứng được yêu cầu tính toán cao trong thời gian hợp lý, do hạn chế về năng lực xử lý và khả năng khai thác tài nguyên phần cứng. Do đó, cần có phương pháp song song hóa SGD để phân chia khối lượng công việc giữa nhiều tiến trình hoặc máy tính, tận dụng hiệu quả tài nguyên tính toán và giảm thời gian huấn luyện.

Báo cáo này tập trung nghiên cứu hai thiết kế song song chính trong song song hóa dữ liệu (Data Parallelism)—thiết kế tập trung và thiết kế phân tán. Mục tiêu là đánh giá hiệu suất huấn luyện, độ chính xác mô hình và khả năng mở rộng trong môi trường tính toán hiệu năng cao (HPC), từ đó cung cấp cái nhìn chi tiết về hai cách tiếp cận này.

## 2. Thuật Toán Song Song

Để tối ưu hóa thuật toán Stochastic Gradient Descent (SGD) trong môi trường phân tán, hai thiết kế song song chính được triển khai: Thiết kế tập trung (Centralized Design) và Thiết kế phân tán (Decentralized Design). Cả hai phương pháp nhằm mục tiêu giảm thời gian huấn luyện và đảm bảo độ chính xác mô hình.

### 2.1. Thiết kế tập trung (Centralized Design)

Thiết kế tập trung dựa trên việc sử dụng một tiến trình trung tâm (master node) để quản lý toàn bộ quá trình huấn luyện. Master chịu trách nhiệm khởi tạo và điều phối các tham số mô hình, đồng thời nhận kết quả từ các tiến trình còn lại (worker nodes). Các worker thực hiện tính toán gradient cục bộ dựa trên phần dữ liệu được phân công, sau đó gửi kết quả này về master. Master sẽ tổng hợp thông tin từ các worker, cập nhật tham số mô hình, và phát truyền lại tham số đã cập nhật để tiếp tục vòng huấn luyện. Cách tiếp cận này giúp duy trì tính nhất quán của mô hình và đảm bảo quy trình huấn luyện được đồng bộ hóa.

**Luồng hoạt động:**

- Master khởi tạo các tham số mô hình và phát truyền chúng đến các worker.
- Mỗi worker huấn luyện trên dữ liệu của mình, tính toán gradient cục bộ và gửi kết quả về master.
- Master tổng hợp gradient từ các worker, cập nhật tham số mô hình và truyền lại cho các worker.
- Quá trình lặp lại cho đến khi hoàn tất số lượng epoch hoặc đạt điều kiện dừng.

**Mã giả**:

```
Master Node:
1. Initialize model parameters θ.
2. Broadcast θ to all worker nodes.
3. Repeat for each epoch:
    a. Gather gradients {∇L(θ)_i} from all worker nodes.
    b. Compute weighted average of gradients:
                ∇L(θ) = Σ(w_i * ∇L(θ)_i) / Σ(w_i)
    c. Update parameters: θ = θ - η * ∇L(θ).
    d. Broadcast updated θ to all worker nodes.
4. Output final model θ.

Worker Node:
1. Receive initial model parameters θ from master.
2. Repeat for each epoch:
    a. Compute gradient ∇L(θ)_i on local dataset.
    b. Send ∇L(θ)_i to master.
    c. Receive updated θ from master.
3. Terminate.
```

Thiết kế tập trung có ưu điểm lớn ở tính quản lý tập trung, giúp dễ triển khai và theo dõi trạng thái của toàn bộ hệ thống. Đây là phương pháp phù hợp khi số lượng worker nhỏ, do việc xử lý và tổng hợp thông tin ở master không gây quá tải. Tuy nhiên, nhược điểm chính là master dễ trở thành điểm cổ chai (bottleneck) khi số lượng worker lớn, dẫn đến hiệu suất giảm. Đồng thời, chi phí truyền thông giữa master và các worker tăng đáng kể khi số lượng tiến trình tăng, gây ảnh hưởng đến khả năng mở rộng của hệ thống.

## 2.2. Thiết kế phân tán (Decentralized Design)

Thiết kế phân tán (Decentralized Design) loại bỏ hoàn toàn vai trò của master node, đảm bảo tất cả các tiến trình (nodes) tham gia bình đẳng vào quá trình tính toán và tổng hợp thông tin. Các nodes khởi tạo tham số mô hình giống nhau và thực hiện huấn luyện trên dữ liệu cục bộ. Thông qua cơ chế `Allgather`, các nodes chia sẻ gradient hoặc trọng số với nhau, sau đó mỗi node tính toán giá trị trung bình từ thông tin thu thập được để cập nhật tham số mô hình. Cách tiếp cận này tạo ra một quy trình đồng đẳng, trong đó các tiến trình hoạt động độc lập nhưng vẫn duy trì tính nhất quán của mô hình.

**Luồng hoạt động**:

1. Tất cả các nodes khởi tạo tham số mô hình giống nhau.
2. Mỗi node huấn luyện trên tập dữ liệu cục bộ, tính toán gradient.
3. Các nodes sử dụng `Allgather` để chia sẻ gradient hoặc trọng số.
4. Mỗi node cập nhật tham số mô hình dựa trên giá trị trung bình thu thập được.
5. Quá trình lặp lại cho đến khi hoàn tất.

**Mã giả**:

```
All Nodes:
1. Initialize model parameters θ.
2. Repeat for each epoch:
    a. Compute gradient ∇L(θ)_i on local dataset.
    b. Share gradients using Allgather.
    c. Compute average gradient:
                ∇L(θ) = Σ(∇L(θ)_i) / N
                (where N is the number of nodes).
    d. Update parameters: θ = θ - η * ∇L(θ).
3. Output final model θ.
```

Thiết kế phân tán có ưu điểm vượt trội trong việc loại bỏ điểm cổ chai (bottleneck) thường gặp ở master node, giúp cải thiện khả năng mở rộng khi số lượng nodes tăng. Các nodes hoạt động đồng đẳng, không phụ thuộc vào một trung tâm, từ đó tăng tính linh hoạt và khả năng chịu lỗi của hệ thống. Tuy nhiên, nhược điểm lớn nhất của thiết kế này là chi phí truyền thông cao hơn, do tất cả các nodes đều phải chia sẻ thông tin với nhau. Đồng thời, việc triển khai và đồng bộ hóa các nodes cũng phức tạp hơn, đặc biệt trong các hệ thống lớn và đa dạng về tài nguyên.