# Định luật Amdahl về tối ưu hóa hiệu suất hệ thống

## Định nghĩa

Định luật Amdahl là một công thức toán học mô tả mối quan hệ giữa tốc độ thực thi của một chương trình và tỷ lệ phần trăm thời gian được dành cho một phần cụ thể của chương trình đó.

## Công thức:

Tăng tốc tối đa = 1 / (1 - (Phần được tối ưu hóa / 100))

**Giải thích:**

* **Tăng tốc tối đa:** Tốc độ tối đa có thể đạt được sau khi tối ưu hóa.
* **Phần được tối ưu hóa:** Tỷ lệ phần trăm thời gian được dành cho phần chương trình được tối ưu hóa.

**Ví dụ:**

Giả sử một chương trình được thực thi trong 100 giây và 20% thời gian (20 giây) được dành cho một phần cụ thể của chương trình. Nếu chúng ta có thể tối ưu hóa phần này để nó chạy nhanh gấp 5 lần, thời gian thực thi sẽ giảm xuống còn 80 giây.

**Tăng tốc tối đa:**

Tăng tốc tối đa = 1 / (1 - (20 / 100)) = 1.25

**Thời gian thực thi sau khi tối ưu hóa:**

Thời gian thực thi mới = Thời gian thực thi ban đầu / Tăng tốc tối đa = 100 giây / 1.25 = 80 giây

**Lưu ý:**

* Định luật Amdahl chỉ là một ước tính và không phải lúc nào cũng chính xác.
* Việc tối ưu hóa một phần nhỏ của chương trình có thể không mang lại lợi ích đáng kể nếu phần đó chỉ chiếm một phần nhỏ thời gian thực thi.
* Cần xác định các phần chương trình chiếm nhiều thời gian thực thi nhất để tập trung tối ưu hóa.

## Ứng dụng định luật Amdahl vào tối ưu hóa hệ thống:

* **Xác định các phần mềm hoặc phần cứng chiếm nhiều thời gian nhất trong hệ thống.**
* **Tập trung tối ưu hóa các phần mềm hoặc phần cứng đó.**
* **Có thể sử dụng các công cụ profiling để xác định các phần mềm hoặc phần cứng chiếm nhiều thời gian nhất.**
* **Có thể áp dụng các kỹ thuật tối ưu hóa khác nhau như nâng cấp phần cứng, cài đặt phần mềm mới hoặc tối ưu hóa cấu hình hệ thống.**

**Ví dụ:**

* **Nếu hệ thống của bạn thường xuyên bị nghẽn mạng, bạn có thể nâng cấp băng thông mạng hoặc tối ưu hóa cấu hình mạng.**
* **Nếu hệ thống của bạn thường xuyên bị chậm khi truy cập cơ sở dữ liệu, bạn có thể tối ưu hóa truy vấn cơ sở dữ liệu hoặc nâng cấp phần cứng cơ sở dữ liệu.**

**Lưu ý:**

* Việc tối ưu hóa hệ thống là một quá trình liên tục và cần được thực hiện thường xuyên để đảm bảo hiệu suất tối ưu cho hệ thống.
* Nên tham khảo ý kiến của chuyên gia để được tư vấn và hỗ trợ tối ưu hóa hệ thống hiệu quả.

## Các yếu tố quyết định tối ưu hóa phần nào

Để quyết định cải thiện phần nào của hệ thống thì còn phải dựa vào:

* Vùng ảnh hưởng
* Chi phí
* Những lợi ích và cải tiến đạt được
* Rủi ro có thể gặp phải

## Các bước thực hiện

Lập bảng liệt kê tuần tự và xác định mỗi bước chiếm bao nhiêu thời gian

|  |  |  |
| --- | --- | --- |
| Bước | Thời gian thực thi (đơn vị: s) | % (tổng thời gian) |
| A |  |  |
| B |  |  |
| C |  |  |

Áp dụng công thức Amdahl + đánh giá vùng ảnh hưởng, chi phi, những lợi ích và cải tiến đạt được:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bước | Tăng tốc tối đa | Thời gian thực thi mới | Vùng ảnh hưởng | Chi phí | Lợi ích và cải tiến đạt được | Rủi ro |
| A |  |  |  |  |  |  |
| B |  |  |  |  |  |  |
| C |  |  |  |  |  |  |

Dựa vào đó để quyết định sẽ tối ưu hóa phần nào

## Tài liệu tham khảo

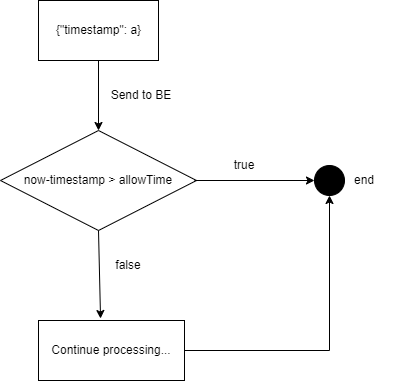
**[Thinking Clearly about Performance](https://queue.acm.org/detail.cfm?id=1854041)**

**[Improving the performance of complex software is difficult, but understanding some fundamental principles can make it easier](https://queue.acm.org/detail.cfm?id=1854041).**

# Các cách bảo mật API

## Tokens (JWT)

## Timestamp



## Chữ ký (signature generation)

Cơ chế:

* Bên gửi tạo public key, private key để ký và chỉ gửi đi public key
* Bên nhận: nhận public key để verify

## Nonce (number once – số dùng một lần)

Dùng để tránh reply attack

## Rate limits

## Ip access limits

## Blacklist

## Whitelist

# Xử lý Request Đồng thời

## Khóa lạc quan

### Giả định

Dữ liệu ít có khả năng bị xung đột khi truy cập đồng thời.

### Cách thức hoạt động

* Luồng truy cập dữ liệu sẽ **không** yêu cầu khóa trước khi truy cập.
* Luồng sẽ **kiểm tra** xem dữ liệu đã bị thay đổi bởi luồng khác hay chưa trước khi thực hiện thay đổi.
* Nếu dữ liệu **không** bị thay đổi, luồng sẽ thực hiện thay đổi và **cập nhật** dữ liệu.
* Nếu dữ liệu **bị** thay đổi, luồng sẽ **thử lại** hoặc **báo lỗi**.

### Ưu điểm

* Hiệu suất cao hơn vì không yêu cầu khóa trước khi truy cập dữ liệu.
* Tăng khả năng đồng thời vì nhiều luồng có thể truy cập dữ liệu cùng lúc.

### Nhược điểm

* Có thể xảy ra xung đột dữ liệu nếu nhiều luồng cố gắng thay đổi dữ liệu cùng lúc.
* Cần có cơ chế để giải quyết xung đột dữ liệu.

## Khóa bi quan

### Giả định

 Dữ liệu có khả năng cao bị xung đột khi truy cập đồng thời.

### Cách thức hoạt động

* Luồng truy cập dữ liệu sẽ **yêu cầu** khóa trước khi truy cập.
* Nếu khóa được cấp, luồng sẽ **truy cập** và **thay đổi** dữ liệu.
* Sau khi thay đổi xong, luồng sẽ **giải phóng** khóa.

### Ưu điểm

* Đảm bảo không xảy ra xung đột dữ liệu.
* Dễ dàng triển khai.

### Nhược điểm

* Hiệu suất thấp hơn vì luồng phải chờ đợi khóa trước khi truy cập dữ liệu.
* Giảm khả năng đồng thời vì chỉ có một luồng có thể truy cập dữ liệu cùng lúc.

## Lựa chọn phương pháp nào phù hợp

### Khóa lạc quan

Phù hợp cho các tình huống:

* Dữ liệu ít có khả năng bị xung đột.
* Hiệu suất cao là quan trọng.
* Có thể chấp nhận nguy cơ xung đột dữ liệu.

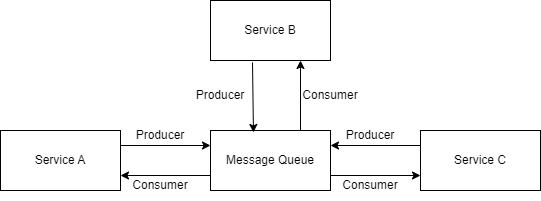
### Khóa bi quan

 Phù hợp cho các tình huống:

* Dữ liệu có khả năng cao bị xung đột.
* Tính toàn vẹn dữ liệu là quan trọng.
* Hiệu suất không phải là yếu tố quan trọng nhất.

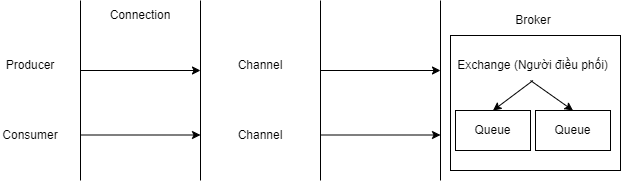
# Kiến trúc Message Queue

## Tại sao phải sử dụng message queue?



* Service gọi đến MQ => tách biệt các thành phần của hệ thống => giảm sự phụ thuộc giữa các service và tăng cường tính bảo mật.
* Xử lý bất đồng bộ => Cứ gửi lên MQ rồi không cần quan tâm => tăng hiệu suất
* Là nơi trung chuyển, điều phối => tăng khả năng chịu tải
* Message đẩy lên MQ => có các cơ chế bảo toàn và khôi phục => Đảm bảo hệ thống thông suốt
* Có thể sắp xếp độ ưu tiên của message => có thể lên lịch để thực hiện các message => linh hoạt và tăng hiệu suất

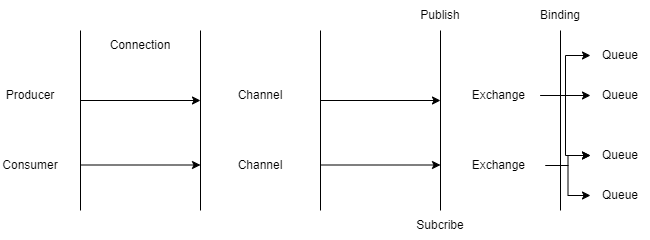
## Kiến trúc (RabbitMQ)



## Một số tham số

* Durable:
  + True: ghi queue vào rom, restart vẫn còn
  + False: ghi queue vào ram
* Persistant: True thì ghi vào rom
* noAck:
  + true: Gửi thì xóa luôn message
  + false: consumer phải gửi xác nhận thì mới xóa
* Ttl: Thời gian sống của một message

## Mô hình publish, subcribe



Exclusive: Nếu không có consumer nào subcribe thì clear hàng đợi

## Các lại exchange

### Direct: 1-1 (gửi đến queue có routingkey = routingkey)

* Định tuyến tin nhắn dựa trên **routing key** chính xác.
* Tin nhắn sẽ được gửi đến **queue** có **binding key** khớp chính xác với **routing key** của tin nhắn.
* **Ví dụ:**
  + **Routing key:** "news.sports"
  + **Binding key:** "news.sports"
  + **Queue:** "sports-news"

### Fanout: 1- All (không quan tâm routingkey)

* Gửi tin nhắn đến **tất cả** các **queue** đã được **binding** với **exchange**.
* **Routing key** không được sử dụng để định tuyến tin nhắn.
* **Ví dụ:**
  + **Routing key:** "any"
  + **Binding key:** "any"
  + **Queue:** "queue1"
  + **Binding key:** "any"
  + **Queue:** "queue2"

### Topic: 1- nhiều (routingkey và pattern)

* Định tuyến tin nhắn dựa trên **pattern matching** với **routing key**.
* Tin nhắn sẽ được gửi đến **queue** có **binding key** khớp với **routing key** của tin nhắn bằng **wildcard matching**.

#### \* (single wildcard)

* Phù hợp với bất kỳ ký tự nào tại vị trí được đặt.
* Ví dụ:
  + Routing key: "news.\*"
  + Binding key: "news.sports" -> Khớp
  + Binding key: "news.world" -> Khớp
  + Binding key: "news.sports.football" -> Không khớp

#### # (multi wildcard)

* Phù hợp với bất kỳ số lượng ký tự nào, bao gồm cả thư mục con.
* Ví dụ:
  + Routing key: "news.#"
  + Binding key: "news" -> Khớp
  + Binding key: "news.sports" -> Khớp
  + Binding key: "news.sports.football" -> Khớp

#### Kết hợp các wildcard

* Ví dụ:
  + Routing key: "\*.sports.#"
  + Binding key: "news.sports" -> Khớp
  + Binding key: "news.world" -> Không khớp
  + Binding key: "news.sports.football" -> Khớp

#### Lưu ý

* Wildcard chỉ được sử dụng trong binding key, không được sử dụng trong routing key.
* Wildcard không phân biệt hoa thường.

#### Bảng so sánh

|  |  |  |
| --- | --- | --- |
| Wildcard | Mô tả | Ví dụ |
| \* | Phù hợp với bất kỳ ký tự nào | news.\*, news.sports, news.world |
| # | Phù hợp với bất kỳ số lượng ký tự nào | news.#, news, news.sports, news.sports.football |

### Header: Dựa trên đặc tính của header

* Định tuyến tin nhắn dựa trên các **header** được đính kèm với tin nhắn.
* **Routing key** không được sử dụng để định tuyến tin nhắn.
* **Ví dụ:**
  + **Header:** "type": "news"
  + **Binding key:** {"type": "news"}
  + **Queue:** "news-queue"