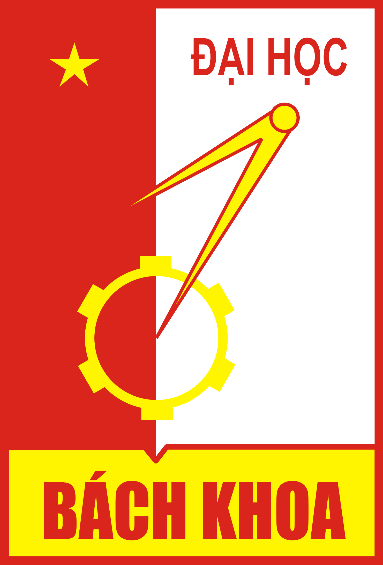
**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

-----🙞🙜🕮🙞🙜-----



**BÁO CÁO BÀI TẬP LỚN**

**PHÁT TRIỂN PHẦN MỀM PHÂN TÁN**

Đề tài:

**Xây dựng hệ thống phân tán thu thập và xử lý Log**

Giảng viên hướng dẫn: Th.s Vũ Thị Hương Giang

Nhóm sinh viên thực hiện: Vũ Văn Lâm 20132231

Nguyễn Đức Quang 20133089

Nguyễn Duy Hoan 20131526

Phạm Nhật Linh 20132307

*Hà Nội, tháng 11/2017*

**MỤC LỤC**

[I. Đặt vấn đề 1](#_Toc497662349)

[1.1. Lý do chọn đề tài 1](#_Toc497662350)

[1.2. Nguyên tắc thiết kế 1](#_Toc497662351)

[1.2.1. Cơ sở toán học 1](#_Toc497662352)

[1.2.2. Mô hình trừu tượng 3](#_Toc497662353)

[1.2.3. Mô hình cụ thể 3](#_Toc497662354)

[1.2.4. Mô hình chuyên biệt 5](#_Toc497662355)

[1.3. Phân công công việc 5](#_Toc497662356)

[II. Mô tả hệ thống 6](#_Toc497662357)

[2.1. Yêu cầu chức năng 6](#_Toc497662358)

[2.2. Yêu cầu phi chức năng 8](#_Toc497662359)

[2.3. Giao diện 8](#_Toc497662360)

[III. Kiến trúc hệ thống 10](#_Toc497662361)

[3.1. Kiến trúc logic 10](#_Toc497662362)

[3.1.1. Biểu đồ Usercase 10](#_Toc497662363)

[3.1.2. Đặc tả usecase 11](#_Toc497662364)

[3.2. Kiến trúc vật lý 12](#_Toc497662365)

[3.2.1. Producer 13](#_Toc497662366)

[3.2.2. Consumer 13](#_Toc497662367)

[3.2.3 Master 14](#_Toc497662368)

[3.3. Ảnh màn hình một số chức năng chính 14](#_Toc497662369)

[IV. Mô hình giao thức và thuật toán truyền thông 16](#_Toc497662370)

[4.1. Mô hình 16](#_Toc497662371)

[4.1.1. Mô hình truyền thông điệp 16](#_Toc497662372)

[4.1.2. Mô hình Publish-subcribe 16](#_Toc497662373)

[4.2. Thuật toán 17](#_Toc497662374)

[4.2.1. Bầu chọn người đứng đầu 17](#_Toc497662375)

[4.2.2. Loại trừ lẫn nhau 18](#_Toc497662376)

[4.3.3. Thuật toán kết thúc 18](#_Toc497662377)

[V. Công nghệ sử dụng 20](#_Toc497662378)

[5.1. JSF 20](#_Toc497662379)

[5.2. Hadoop 20](#_Toc497662380)

[5.3. Zookeeper 20](#_Toc497662381)

[5.4. Hbase 21](#_Toc497662382)

[5.5. Kafka 21](#_Toc497662383)

[VI. Kết luận và hướng phát triển 23](#_Toc497662384)

[VII. Tài liệu tham khảo 24](#_Toc497662385)

# 

# Đặt vấn đề

* 1. Lý do chọn đề tài

Ngày nay, công nghệ thông tin ngày càng phát triển, việc thu thập và xử lý khối lượng thông tin ngày càng lớn, một máy không thể đảm nhiệm hết được. Vì thế, các hệ thống phân tán ngày càng được phát triển để đáp ứng nhu cầu này. Bên cạnh đó, những mối đe dọa trên internet ngày càng nhiều, việc phân tích log là một phần giúp ích không nhỏ cho phát hiện các nguy cơ đó. Vì thế, chúng em chọn đề tài “hệ thống thu thập log phân tán”.

Công nghệ sử dụng trong đề tài của chúng em là JSF(framework ứng dụng web), Hadoop(framework cho phép xử lý phân tán), Zookeeper(service duy trì cấu hình), Kafka(service streaming dữ liệu) và Hbase(hệ cơ sở dữ liệu).

* 1. Nguyên tắc thiết kế
     1. Cơ sở toán học
* Thu thập

Các giải thuật xử lý:

* + - Giải thuật phân loại message theo topic.
    - Xử lý trên bộ nhớ dùng chung
    - Tối ưu hóa xử lý đa tiến trình, cân bằng tải
    - Lọc dữ liệu

Các service message được phân loại theo các topic riêng biệt. Mỗi topic bao gồm các partitions. Mỗi partition được lưu trữ trên mỗi vùng nhớ của cluster broker (Các server chia sẻ bộ nhớ dùng chung). Bao gồm:

* Producer: Các message được gửi đến cluster broker bao gồm (key và value) trong đó thành phần key sẽ xác định việc lưu trữ value trên partition nào trên server. Nếu không xác định key, thành phần value sẽ được hash để xác định partition.
* Consumer: Các consumer subcribe 1 hoặc nhiều topic, đọc dữ liệu từ các partitions lưu trữ trên servers. (Việc phân chia partitions cho consumer dựa vào số lượng của môi consumer khởi tạo trong cùng một group). Mỗi khi 1 consumer add hoặc remove cluster broker sẽ thực hiện reblancing, phân chia lại các partition cho mỗi consumer tồn tại.
* Lưu trữ

Các giải thuật:

* Lưu trữ phân tán
* Xử lý replication
* Xây dựng rowkey hỗ trợ xử lý realtime với database dung lượng lớn lên tới hàng triệu row

Hệ thống database được lưu trữ phân tán trên các server, dữ liệu được lưu trữ theo cơ chế column-orientation. Các table trong database được phân chia thành các regions. Mỗi region trong table lưu trữ column family. Việc xác định dữ liệu được lưu trữ theo trong table dựa vào trật tự alphabe của các rowkey được lưu trữ. Rowkey của các record được xử lý theo trật tự xác định, giúp tối ưu hóa khả năng phân chia dữ liệu vào các region.

Kiến trúc Database server bao gồm:

* **Hmaster**: thành phần phản lý hệ thống, là thành phần trung tâm trong kiến trúc của database
  + HMaster giám sát tất cả các Region Server thuộc cluster.
  + Tất cả các thay đổi liên quan đến metadata đều thực hiện thông qua HMaster  
    Những vai trò quan trọng nhất được thực hiện bởi Hmaster:
    - * Table : createTable, removeTable, enable, disable
      * ColumnFamily : add Column, modify Column
      * Region : move, assign
* **Hregionserver**: quản lý và lưu trữ các region
  + HRegionServer chịu trách nhiệm quản lý các region.  
    Khi Region server nhận writes and read requests từ client, nó sẽ assign request cho region riêng biệt, nơi mà đang chứa column family. Tuy nhiên client có thể trực tiếp liên lạc với Hregion servers, mà không cần sự cho phép của HMaster.
  + Hregions: chứa 2 component chính là Memstore và Hfile
  + HRegions là thành phần kiến trúc cơ sở của Hbase cluster. Bao gồm 2 thành phần chính là Memstore và Hfile.
* **Zookeeper**: lưu trữ các metadata, region info.
  + 1. Mô hình trừu tượng
* Webserver: Sinh dữ liệu (message, log).
* Agents: Lấy log từ webserver, định dạng log, phân loại log theo chủ đề và gửi log lên server thu thập (Xây dựng các producers. Mỗi producer sẽ đảm nhiệm xử lý và gửi các message đã được xử lý theo chủ đề và gửi lên lưu trữ trong các partitions của từng topic).
* Server thu thập:
* Xây dựng consumer và group: Nhận message từ các producers theo từng chủ đề (Xây dựng các consumer và các group. Mỗi group consumer sẽ xử lý message theo các chủ đề khác nhau. Consumers subcribe các topic có nhiệm vụ nhận message từ các partitions). Các message sau khi được nhận sẽ được gửi vào từng queue theo từng loại topic.
* Xây dựng tiến trình InsertTask: Các insertTask có nhiệm vụ lấy dữ liệu từ các queue và commit lên database.
* Xây dựng console hiển thị: Hiển thị realtime các message được gửi tới từ các producer.
* Webserver hiển thị: Hiển thị biểu đồ message theo các dạng khác nhau: chủ đề, server thu thập, trạng thái. Hiển thị tổng số lượng message gửi tới và hỗ trợ query realtime
  + 1. Mô hình cụ thể
* Producer: Bao gồm các định dạng cấu hình: BoostrapHosting, KeySerializer, ValueSerializer, BufferMemory, BatchSize…

Các API:

* + ProducerRecord (string topic, k key, v value): Topic các message được assign, key giá trị key của message (Xác định partitions của topic lưu trữ message được gửi) và value (nội dung của message).
  + Send(topic,key,value): Producer gửi message
* Consumer: Bao gồm các định dạng cấu hình: BoostrapHosting, KeyDeserializer, ValueDeserializer, Auto\_Commit, IntervalCommit, Group.ID, Session\_Timeout…

Các API:

* + Subcribe(String topic): subcribe vào một topic đã được khởi tạo để nhận các message
  + public ConsumerRecords(java.util.Map<TopicPartition,List<Consumer-Record>K,V>>> records)
* DatabaseServer:
  + HBaseConfiguration.create(): Khởi tạo configuration đến database server
  + HBaseAdmin bao gồm các API
    - createTable(HTableDescriptor name): Tạo table mới
    - createTable(HTableDescriptor desc, byte[][] splitKeys): Tạo table mới và các regions
    - deleteColumn(byte[] tableName, String columnName): Xóa một column
    - deleteTable(String tableName): Xóa một table
  + HTableDescriptor(TableName name): Khởi tạo contructor cho một htable
  + HTableDescriptor addFamily(HColumnDescriptor family): Thêm một column family vào talbe
  + Put(), Get(),Scan(),Delete(),Filter()…: Hỗ trợ việc xử lý dữ liệu trong table.
* Webserver hiển thị:
  + HighChart.chart( chart{},title{},subtitle{},xAxis{},yAxis{},Series{}): Xây dựng biểu đồ
    1. Mô hình chuyên biệt

Các class:

* Task: Producer, Consumer, InsertTask
* Entities: Message, Queue, ListCommit, ObjectView
* Helper: ConfigurationHelper, TableHelper, ConfigHelper, MessageHelper
* Controller: MessageController
* ManagedBean(JSF-Webserver): messageManagedBean
* View(xhtml): messageView
  1. Phân công công việc

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| STT | Họ và tên | Nhiệm vụ | % đóng góp | Mức độ hoàn thiện |
| 1 | Vũ Văn Lâm | Xây dụng producer, agent, xử lý dữ liệu | 25% | 100% |
| 2 | Nguyễn Duy Hoan | Cấu hình hệ thống phân tán | 25% | 100% |
| 3 | Phạm Nhật Linh | Lưu trữ dữ liệu phân tán. | 25% | 100% |
| 4 | Nguyễn Đức Quang | Thiết kế hệ thống, nêu phương án | 25% | 100% |

# Mô tả hệ thống

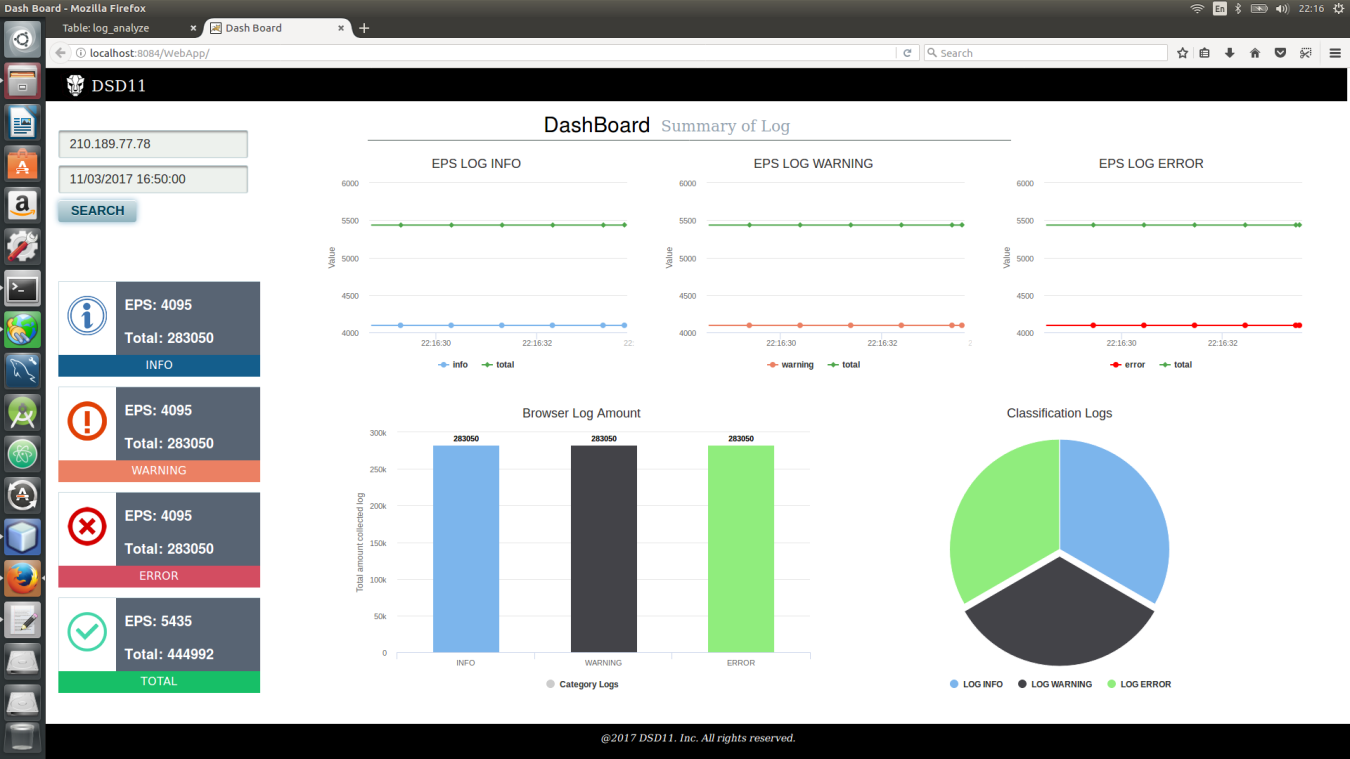
## Yêu cầu chức năng

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nhóm chức năng | Tên chức năng | Đầu vào | Tiến trình | Đầu ra | Ngoại lệ |
| Thu thập | Thiết lập kết nối từ agent | IP, port | Thiết lập kết nối từ agent tới server thông qua địa chỉ IP, port | Kết nối thành công hoặc mã lỗi | - Không có kết nối mạng - Không kết nối được tới địa chỉ ip - Timeout |
| Thu thập | Agent gửi yêu cầu đến các đối tượng giám sát | Thông tin về đối tượng giám sát | Gửi yêu cầu đến đối tượng giám sát | Thành công hoặc trả về mã lỗi |  |
| Thu thập | Lọc dữ liệu | Dữ liệu từ webserver | Lọc bỏ các trường dữ liệu không cần thiết | Dữ liệu đã được lọc | Dữ liệu từ server là null |
| Thu thập | Đóng gói dữ liệu | Dữ liệu đã được lọc | Đóng gói dữ liệu thành object dưới dạng json | Object dưới dạng json |  |
| Thu thập | Gửi dữ liệu đã đóng gói đến server thu thập | Dữ liệu đã được đóng gói | Gửi dữ liệu qua kết nối đã tạo | Gửi dữ liệu thành công hoặc trả về lỗi | Mất kết nối mạng |
| Tiếp nhận | Pooling (sắp xếp các response chờ xử lý) | Dữ liệu gửi từ agent | Tiếp nhận gói tin từ agent gửi đến vào hàng đợi, gói nào tới trước xử lý trước | Hàng đợi sắp xếp |  |
| Xử lý | Chuẩn hóa dữ liệu | Dữ liệu thô trả về từ server (json) | Bóc tách dữ liệu theo các thuộc tính và đóng gói thành các object | Các object với các trường dữ liệu cần thiết |  |
| Hiển thị | Hiện thị nguyên gốc thông tin đã tiếp nhận | Thông tin do agent gửi về | Hiển thị dữ liệu nhận được từ agent dưới dạng text | Text log |  |
| Hiển thị | Hiển thị dữ liệu theo nguồn thu thập | Nguồn thu thập, dữ liệu nhận được từ agent | - Truy xuất dữ liệu theo nguồn thu thập - Hiển thị dữ liệu theo nguồn thu thập | Dữ liệu theo từng nguồn thu thập |  |
| Hiển thị | Hiển thị dữ liệu theo khoảng thời gian | Thời gian bắt đầu, thời gian kết thúc, dữ liệu nhận được | - Truy xuất dữ liệu theo thời gian đầu vào - Hiển thị dữ liệu | Dữ liệu theo khoảng thời gian( sau mỗi giây) |  |
| Lưu trữ | Lưu trữ dữ liệu nhân được | Dữ liệu đã được xử lý | Lưu trữ dữ liệu vào db | Dữ liệu được lưu vào CSDL |  |

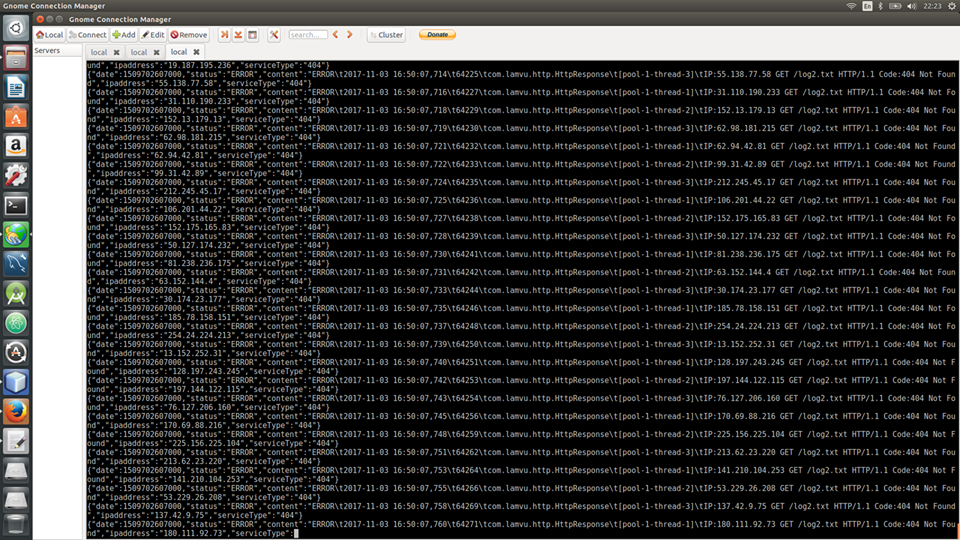
## Yêu cầu phi chức năng

|  |  |  |
| --- | --- | --- |
| STT | Yêu cầu | Thông số cần đáp ứng |
| 1 | Tốc độ xử lý thông điệp của hệ thống | 12000 eps |
| 2 | Tương thích trên nhiều máy tính | Dùng được trên nhiều máy tính |
| 3 | Giao diện ưa nhìn | Đẹp |
| 4 | Dễ sử dụng hệ thống | Người dùng có thể sử dụng sau khi học 20 phút |
| 5 | Độ tin cậy cao | <10 lỗi, khả năng tái hoạt động sau khi gặp sự cố <10 phút |
| 6 | An toàn thông tin | Dữ liệu bảo toàn >95% khi có sự cố mạng |

## Giao diện



*Hình 2.1. Màn hình hiển thị biểu đồ log*

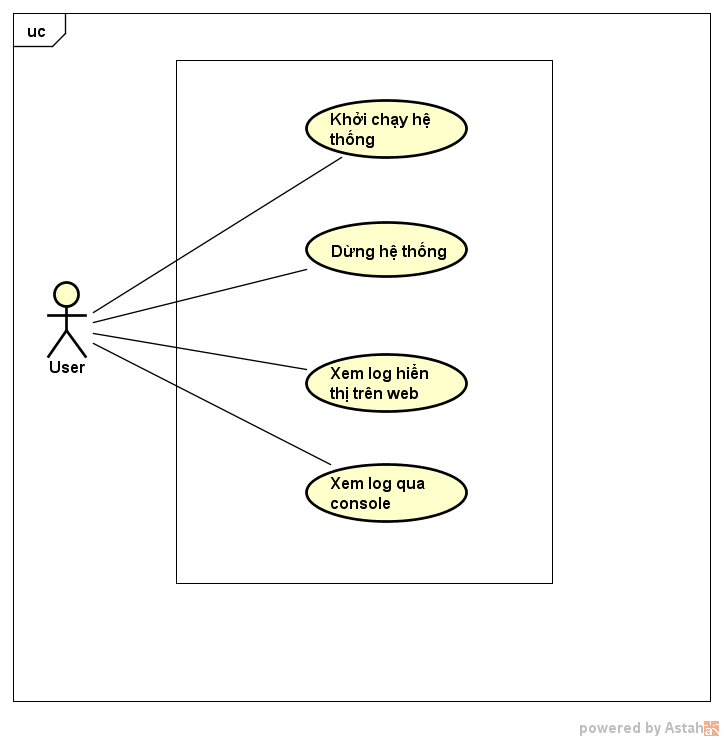


*Hình 2.2. Màn hình hiển thị log trên console*

# Kiến trúc hệ thống

## Kiến trúc logic

### Biểu đồ Usercase



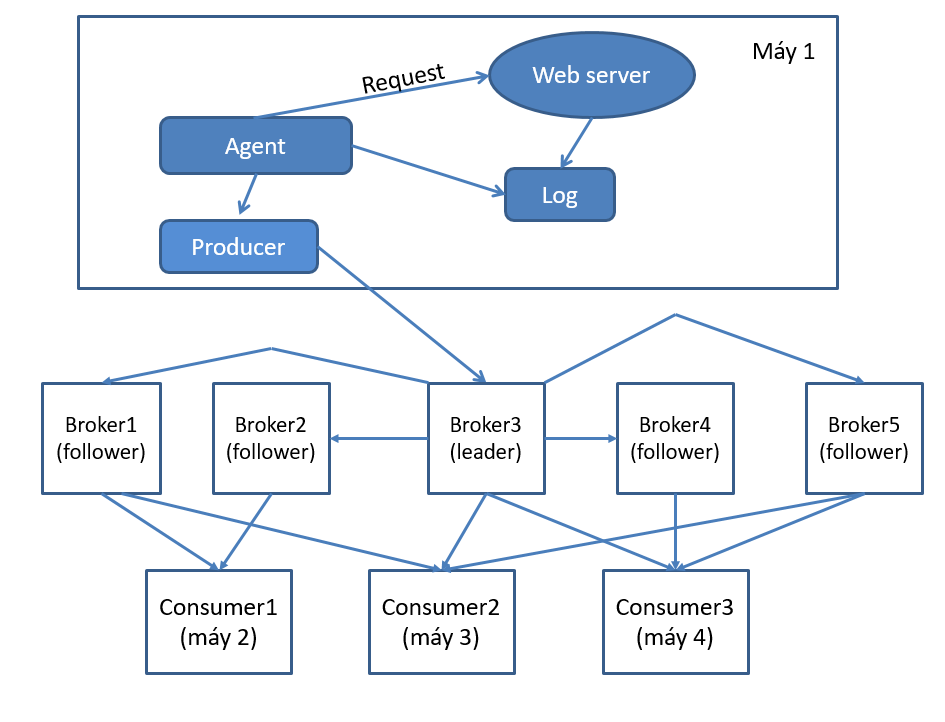
Biểu đồ 3.1: Biểu đồ Usecase

Người dùng hệ thống có thể khởi chạy hệ thống, dừng hệ thống, xem log qua web hoặc console.

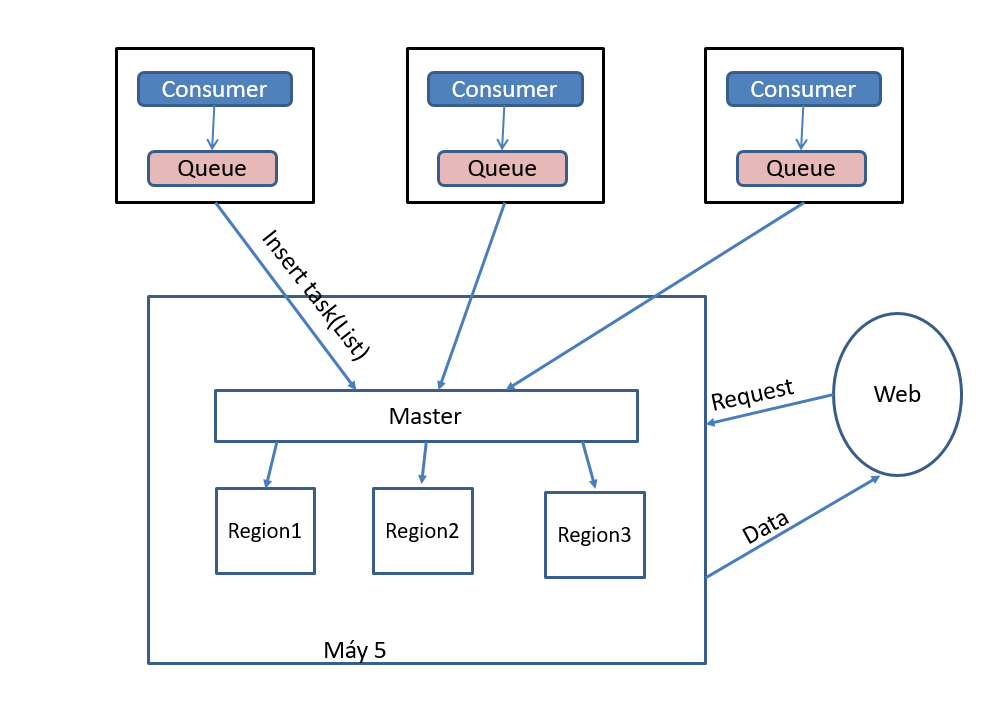
### Đặc tả usecase

* Khởi chạy hệ thống: Người dùng có thể khởi chạy hệ thống bằng một trong các cách sau:
  + Console
  + Chạy trực tiếp phần mềm
* Dừng hệ thống: Người dùng có thể dừng hệ thống bất kì lúc nào bằng:
* Console
* Trực tiếp bằng phần mềm
* Xem log hiển thị trên web: người dùng có thể xem số lượng, biểu đồ hiển thị các loại log trên giao diện web.
* Xem log qua console: người dùng có thể xem log chạy trực tiếp qua console.

## Kiến trúc vật lý



Hình 3.1. Consumer và producer



*Hình 3.2. Cosumer và master*

Hệ thống bao gồm 1 producer, 3 consumer và 1 master(5 máy tính vật lý)

### Producer

Producer đảm nhận vai trò của Agent, sẽ gửi request đến webserver và nhận các log gửi về được đưa vào 1 thư mục. Sau đó sẽ phân tích và xử lý các log và gửi về các broker.

### Consumer

Các consumer sẽ subscribe vào các topic để lấy log. Mỗi khi các broker chứa các log nằm trong các topic mà các consumer đã subscribe thì các broker sẽ gửi các log về cho các consumer này.

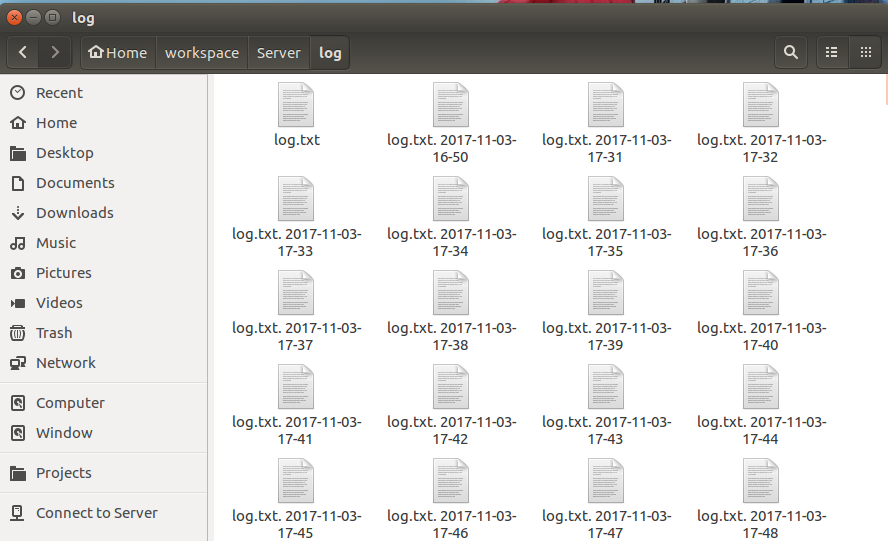
Các broker(5 broker tương ứng với 5 máy tính vật lý) sẽ bầu ra 1 broker leader đảm nhiệm vai trò điều khiển gửi log vào consumer.

### 3.2.3 Master

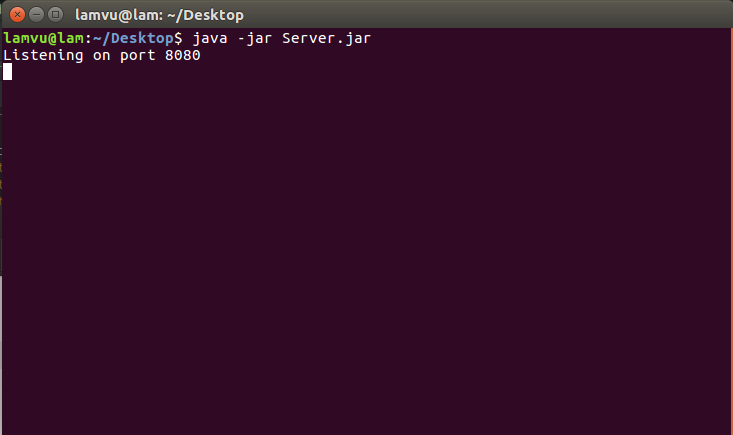
Các consumer sau khi đã có log sẽ đẩy log vào queue, từ đó thực hiện insert log vào DataNote(Master). Master có nhiều region, mỗi region có nhiều partition để đảm nhiệm vai trò lưu trữ của mình.

Mỗi khi web hiển thị request lấy dữ liệu thì các partition sẽ tiến hành gửi dữ liệu lên web theo cơ chế đã cài đặt.

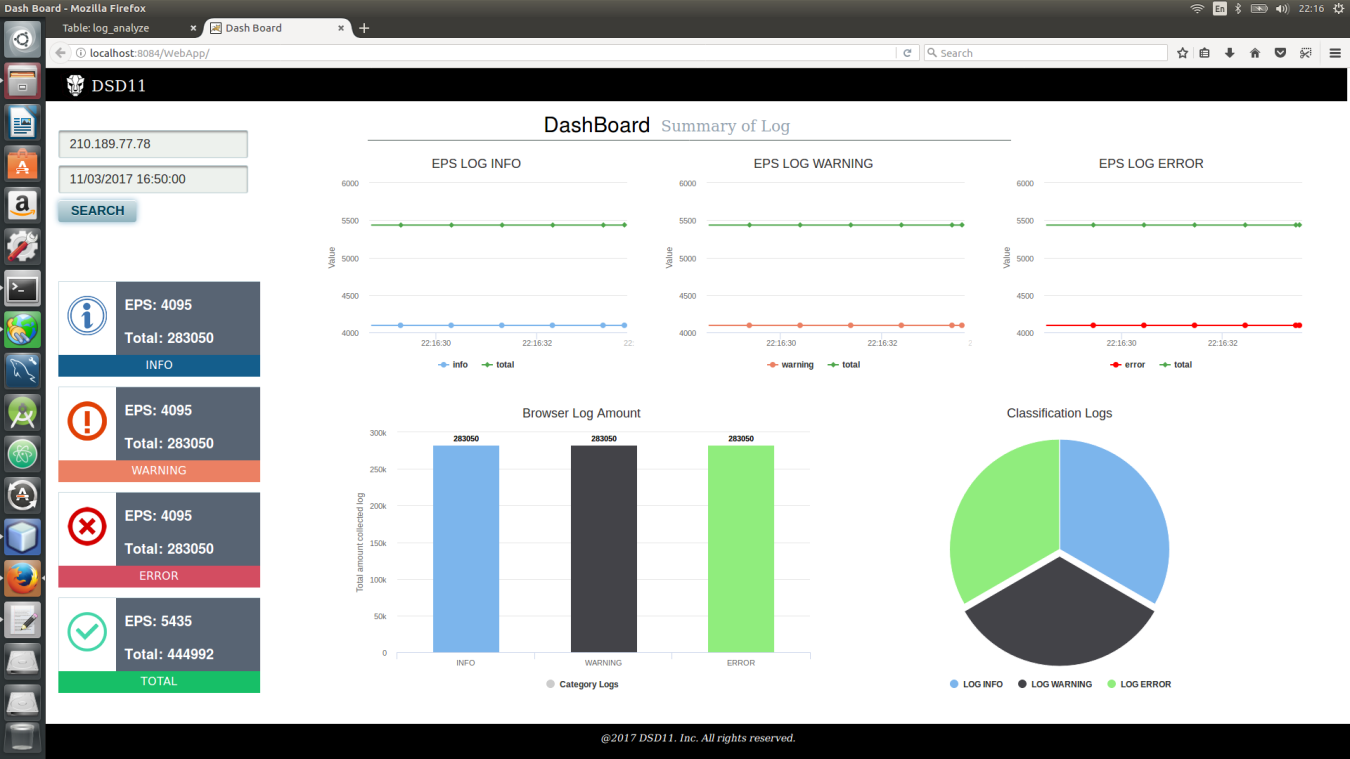
## 3.3. Ảnh màn hình một số chức năng chính

****

*Hình 3.3. Lưu file log mỗi phút*

****

*Hình 3.4. Khởi chạy hệ thống*

****

*Hình 3.5. Hiển thị log trên nền web*

# Mô hình giao thức và thuật toán truyền thông

## Mô hình

* + 1. Mô hình truyền thông điệp

1. Tổng quan

Mỗi tiến trình tham gia vào quá trình giao tiếp truyền nhận thông điệp là một đối

tượng riêng biệt trong hệ thống. Mỗi tiến trình này có bộ nhớ riêng và năng lực xử lí

Không cần phải giống nhau. Các tiến trình giao tiếp với nhau bằng cách gửi và nhận thông điệp:

- Tiến trình A gửi thông điệp yêu cầu đến tiến trình B

- B xử lý thông điệp và gửi thông điệp trả lời A

- Thông điệp trả lời có thể kích hoạt thông điệp khác, cứ thế tiếp diễn

B) Ưu, nhược điểm

* Ưu điểm: Dễ quản lý tài nguyên
* Nhược điểm: Mất thời gian chờ response, gây lãng phí tài nguyên

C) Lý do chọn

Áp dụng vào hệ thống: Hệ thống thu thập log sử dụng agent. Agent gửi yêu cầu đến server, server sẽ trả về thư mục log, từ đó, agent có thể đọc từ thư mục đó và thực hiện các việc tiếp theo.

* + 1. Mô hình Publish-subcribe

1. Tổng quan

Publisher sinh ra dữ liệu dưới dạng các sự kiện, subscriber đăng ký nhận dữ liệu theo chủ đề hoặc nội dung. Thực thể trung gian (mediator) đóng vai trò thông báo các sự kiện tới subscriber mỗi khi có sự thay đổi.

1. Ưu, nhược điểm

* Ưu điểm: Phân bổ tài nguyên, năng lực xử lý tốt
* Nhược điểm: Phải thiết lập và quản lý nhiều kết nối

1. Lý do chọn

Áp dụng vào hệ thống: xử lý và lưu trữ log.

Agent đóng vai trò là Producer, tiến hành gửi message lên các Cluster broker theo topic, các Consumer sẽ nhận được các message từ Cluster theo topic đã đăng kí từ trước.

## Thuật toán

* + 1. Bầu chọn người đứng đầu

1. Tổng quan

Đây là thuật toán chọn ra một tiến trình để điều khiển các tiến trình khác. Trước khi hệ thống khởi chạy, các nút của hệ thống chưa biết nút nào là chủ đạo điều khiển các nút khác. Sau khi thuật toán được thực hiên, các nút sẽ chọn ra một nút làm “người đúng đầu” để điều khiển hoạt đông của hệ thống.

1. Áp dụng vào hệ thống

* Hệ thống phải xây dựng database phân tán, nhiều server cùng lưu trữ dữ liệu. Cần xây dựng leader để phục vụ request từ client (Scan, get…) trên nhiều server cùng lúc và yêu cầu backup khi leader xảy ra lỗi.
* Xây dựng leader (database) bao gồm các metadata (bao gồm các thông tin từ các server lưu trữ dữ liệu khác nhau), leader (manager) quản lý và điều phối tài nguyên từ các server khác khi có yêu cầu request từ nhiều server cùng lúc. Xây dựng namenode để thực hiện quản lý các server lưu trữ (datanode) bao gồm các thông tin cơ bản của các server: link, block size. Xây dựng resourcemanager để kiểm soát và điều phối các nguồn tài nguyên hiện có (các nodemanager gửi thông tin tài nguyên hiện có cho resourcemanager). Xây dựng secondarynamenode backup cho namenode.
* Cần xây dựng leader để kiểm soát và quản lý message được gửi đến server thu thập từ agent để cân bằng tải giữa các server đảm nhiệm vụ nhận message.
* Các agent gửi message đến broker lưu trữ trung gian (Lưu trữ trong các partitions trên từng server). Request được gửi đến leader, leader sẽ kiểm tra tình trạng các server thu thập hiện tại đang hoạt động và quyết định việc gửi message có thành công hay không hoặc message được gửi vào partition của server nào.
* Leader bị chết trong quá trình thực hiện quản lý và điều phối message gửi đến broker trung gian từ agent.
* Bầu cử lại leader, leader mới được bầu cử dựa vào id có giá trị gần nhất với id của leader trước đó (Các giá trị id của server được lưu trữ trong file cấu hình).
  + 1. Loại trừ lẫn nhau

1. Tổng quan

Khi các tiến trình được khởi chạy trên hệ thống phân tán, đôi khi chúng xung đột về tài nguyên và thời gian. Thuật toán loại trừ lân nhau sẽ cho duy nhất một tiến trình được sử dụng tài nguyên tại một thời điểm và loại trừ những tiến trình khác.

1. Áp dụng vào hệ thống

* Các tiến trình đảm nhiệm một nhệm vụ tương tự nhau ( nhận dữ message từ agent, đẩy dữ liệu lên database) cùng tranh chấp cùng một tài nguyên ( queue message, queue commit).
* Thiết lập cơ chế lock và unlock trên từng đối tượng tài nguyên
  + 1. Thuật toán kết thúc

1. Tổng quan

Khi muốn xử lý những ngoại lệ không mong muốn của hệ thống, chúng ta có thể kết thúc một tiến trình. Thuật toán kết thúc được thiết kế để kết thúc các tiến trình không phù hợp với hệ thống.

1. Áp dụng vào hệ thống

* Các consumer subcribe một topic message để tiến hành thu nhận message quá lâu
* Thiết lập timeout cho consumer
* Message không được gửi thành công trong lần gửi đầu tiên
* Thiết lập cơ chế tùy chọn retry để gửi lại hoặc bỏ qua message bị gửi lỗi
* Sử dụng queue để lưu trữ message được gửi đến để tiến hành đẩy lên database thông chu kỳ 0.5s. Tuy nhiên số lượng message lớn vượt quá giới hạn cho phép kích thước của queue
* Tiến hành đẩy luôn lên database hoặc lock tiến trình nhận message lại để xử lý theo chu kỳ. Sau đó clear buffer
* Khi một consumer subcribe topic message để nhận message đột ngột bị lỗi
* Kiểm tra tất cả các consumer đang hoạt động tiến hành phân chia lại tài nguyên
* Agent gửi yêu cầu kết nối đến web server và chờ phản hồi quá lâu do mạng yếu
* Thiết lập timeout cho agent

# Công nghệ sử dụng

## JSF

JSF là viết tắt của JavaServer Faces, là một [bộ khung](https://vi.wikipedia.org/w/index.php?title=B%E1%BB%99_khung&action=edit&redlink=1) (framework) phát triển các ứng dụng Web viết bằng Java nhằm làm đơn giản hóa quá trình [phát triển](https://vi.wikipedia.org/wiki/C%C3%B4ng_ngh%E1%BB%87_ph%E1%BA%A7n_m%E1%BB%81m) [giao diện người dùng](https://vi.wikipedia.org/wiki/Giao_di%E1%BB%87n_ng%C6%B0%E1%BB%9Di_d%C3%B9ng) cho các ứng dụng [J2EE](https://vi.wikipedia.org/wiki/J2EE). Để tạo ra giao diện hiển thị, JSF dùng dạng cấu trúc cây của các thẻ, mỗi thẻ là một thành phần giao diện (component) và FacesServlet servlet sẽ thực hiện công đoạn chuyển đổi ra giao diện tương ứng cho người dùng với định dạng HTML.

Lý do chọn vào hệ thống: tiện lợi, giúp cho quá trình phát triển ứng dụng web nhanh chóng hơn. Việc xử lý, hiển thị của hệ thống được dùng công nghệ này.

* 1. Hadoop

Hadoop là một Apache framework mã nguồn mở được viết bằng java, cho phép xử lý phân tán (distributed processing) các tập dữ liệu lớn trên các cụm máy tính (clusters of computers) thông qua mô hình lập trình đơn giản. Hadoop được thiết kế để mở rộng quy mô từ một máy chủ đơn sang hàng ngàn máy tính khác có tính toán và lưu trữ cục bộ (local computation and storage).

Lý do chọn:Các server có thể được thêm vào hoặc gỡ bỏ từ cluster một cách linh hoạt và vẫn hoạt động mà không bị ngắt quãng, vì vậy, khi một máy trong hệ thống ngưng hoạt động, hệ thống vẫn có thể hoạt động.

Một lợi thế lớn của Hadoop ngoài mã nguồn mở đó là khả năng tương thích trên tất cả các nền tảng do được phát triển trên Java, do đó, dễ sử dụng với người dùng Java bọn em.

* 1. Zookeeper

Zookeeper là 1 dịch vụ tập trung để duy trì thông tin cấu hình, đặt tên, cung cấp sự đồng bộ phân tán , và cung cấp các dịch vụ nhóm. Nói cách khác, Zookeeper là 1 dịch vụ đồng bộ hóa nhân rộng với sự nhất quán cuối cùng.

Lý do chọn: Nó tương thích rất tốt với Hadoop, cụ thể như sau:

Zookeeper quản lý toàn bộ quy trình (workflow) việc khởi động (start) và dừng (stop) các nodes khác nhau trong cluster của Hadoop

Trong Hadoop cluster khi bất kỳ xử lý nào cần cấu hình để hoàn thành tác vụ. Zookeeper đảm bảo node đó được cấu hình nhất quán.

Trong trường hợp master node lỗi, Zookeeper có thể gán master node mới và đảm bảo cluster làm việc bình thường.

* 1. Hbase

Là 1 hệ cơ sở dữ liệu mã nguồn mở được xây dựng dựa trên BigTable. HBase cung cấp khả năng lưu trữ dữ liệu lớn lên tới hàng tỷ dòng, hàng triệu cột khác nhau cũng như hàng petabytes dung lượng. HBase là 1 NoSQL điển hình bởi vậy các tables của HBase không có 1 schemas cố định và không có các quan hệ giữa các bảng, không cung cấp phép join giữa các bảng. Hadoop có khả năng mở rộng tốt và được thiết kế để mở rộng theo chiều ngang (Scale horizontally).

Lý do chọn Hbase: Do Hbase có thể lưu trữ được một lượng dữ liệu rất lớn. Giao thức truy vấn nhanh dựa trên key/value, nhanh hơn rất nhiều so với mySQL trong việc lưu trữ dữ liệu lớn.

* 1. Kafka

Kafka có thể hiểu là một hệ thống logging, nhằm lưu lại các trạng thái của hệ thống, nhằm phòng tránh mất thông tin.

Lý do chọn: Kafka rất phù hợp với mô hình Publish-Subcribe đã thiết kế, cụ thể như sau:

* Kafka lưu, phân loại message theo topics
* Kafka sử dụng producers để publish message vào các topics ở trên
* Kafka sử dụng consumers để subscribe vào topics, sau đó xử lý các message lấy được theo một logic nào đó
* Kafka thường được chạy dưới dạng cluster, khi đó mỗi server trong đó sẽ được gọi là broker

# Kết luận và hướng phát triển

Nhóm đã xây dựng một web server đơn giản để xử lý Request và ghi log và một web server log hoàn chỉnh để hiển thị log. Bên cạnh đó, nhóm đã phát triển rất tốt việc xử lý, lưu trữ dữ liệu phân tán với khả năng đáp ứng tốt yêu cầu đề bài, đạt tốc độ yêu cầu. Hệ thống có cơ chế cân bằng tải tốt.

Tuy nhiên, hệ thống còn có nhiều hạn chế: chưa đảm bảo mã hóa dữ liệu khi truyền thông điệp, chưa đảm bảo khả năng dung lỗi, chưa đảm bảo độ ổn định của tốc độ xử lý, chưa có server backup dữ liệu.

Hướng phát triển sản phẩm: trong tương lai, nhóm em sẽ phát triển sản phẩm thêm nhiều server, đảm bảo an toàn thông tin và đảm bảo sự ổn định của hệ thống.

# Tài liệu tham khảo

1. <https://www.tutorialspoint.com/apache_kafka/>
2. <https://www.tutorialspoint.com/hadoop/>
3. <https://www.tutorialspoint.com/zookeeper/>
4. <https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html>
5. Slide phát triển phần mềm phân tán