TÀI LIỆU KỸ THUẬT: MODULE XỬ LÝ DỮ LIỆU

Dự án: ITAPIA
Phiên bản: 1.0
Ngày: 20/05/2024

1. Tổng quan và Vai trò

Module Xử lý Dữ liệu (Data Processing Module) là nền tảng của toàn bộ hệ thống ITAPIA. Nó chịu trách nhiệm cho toàn bộ Vòng đời Dữ liệu (Data Lifecycle), từ việc **thu thập** dữ liệu thô từ các nguồn bên ngoài, **xử lý, làm sạch, chuẩn hóa**, cho đến việc **lưu trữ** một cách có cấu trúc và hiệu quả vào các hệ thống cơ sở dữ liệu (PostgreSQL và Redis).

Mục tiêu cốt lõi của module này là cung cấp một **Nguồn Chân lý Duy nhất (Single Source of Truth)**, đáng tin cậy và nhất quán cho tất cả các module cấp cao hơn như API Gateway và AI Service.

Kiến trúc của module được thiết kế theo hướng module hóa, bao gồm hai quy trình chính hoạt động độc lập: **Batch Processing** (Xử lý theo lô) và **Real-time Processing** (Xử lý Thời gian thực).

2. Kiến trúc Lưu trữ Dữ liệu

Hệ thống sử dụng một kiến trúc lưu trữ kết hợp (hybrid) để tối ưu hóa cho các loại dữ liệu khác nhau:

- **PostgreSQL (CSDL Quan hệ):**
    - **Vai trò:** Lưu trữ các dữ liệu có cấu trúc, ít thay đổi và cần tính toàn vẹn cao.
    - **Các bảng chính:**
        - tickers, exchanges, sectors: Các bảng metadata tĩnh, đã được chuẩn hóa để lưu thông tin về mã chứng khoán, sàn giao dịch và nhóm ngành.
        - daily_prices: Lưu trữ dữ liệu lịch sử giá OHLCV hàng ngày. Đây là dữ liệu nền tảng cho các phân tích xu hướng dài hạn và huấn luyện mô hình.
        - relevant_news: Lưu trữ thông tin về các tin tức liên quan đến từng cổ phiếu.
- **Redis (CSDL In-Memory):**
    - **Vai trò:** Lưu trữ dữ liệu có tần suất ghi cao và yêu cầu truy xuất nhanh, cụ thể là dữ liệu giá trong ngày.
    - **Cấu trúc sử dụng: Redis Streams**. Mỗi ticker sẽ có một Stream riêng (ví dụ: intraday_stream:AAPL). Cấu trúc này cho phép ghi dữ liệu mới một cách hiệu

quả và tự động giới hạn kích thước để chỉ lưu giữ dữ liệu của ngày giao dịch gần nhất, tránh tiêu thụ quá nhiều bộ nhớ.

3. Các Quy trình Xử lý Chính

3.1. Quy trình Xử lý theo Lô (Batch Processing Pipelines)

Các quy trình này được thiết kế để chạy định kỳ (ví dụ: mỗi ngày một lần sau khi thị trường đóng cửa) nhằm cập nhật các dữ liệu lịch sử.

- **Pipeline 1: Thu thập Lịch sử giá (fetch_history.py)**
    - **Mục tiêu:** Đảm bảo bảng daily_prices luôn được cập nhật với dữ liệu giá của ngày giao dịch gần nhất.
    - **Logic:**
        1. **Xác định Khoảng thời gian:** Truy vấn CSDL để tìm ngày collect_date gần nhất đã được ghi. Ngày bắt đầu thu thập mới sẽ là ngày tiếp theo.
        2. **Thu thập Dữ liệu thô:** Gọi API của yfinance để lấy dữ liệu OHLCV hàng ngày cho *tất cả* các ticker đang hoạt động trong một khoảng thời gian (từ ngày bắt đầu đến ngày hôm qua).
        3. **Tái cấu trúc & Làm sạch:** Dữ liệu trả về từ yfinance cần được tái cấu trúc từ dạng cột-nhóm sang dạng hàng dài (long format). Sau đó, áp dụng các kỹ thuật xử lý giá trị thiếu (ví dụ: forward-fill rồi backward-fill) để đảm bảo dữ liệu liên tục.
        4. **Lưu trữ:** Sử dụng một quy trình bulk_upsert để ghi hàng loạt dữ liệu đã làm sạch vào bảng daily_prices, tự động cập nhật nếu một bản ghi đã tồn tại.
- **Pipeline 2: Thu thập Tin tức (fetch_news.py)**
    - **Mục tiêu:** Cập nhật các tin tức mới nhất liên quan đến các cổ phiếu được theo dõi.
    - **Logic:**
        1. **Lấy danh sách Ticker:** Lấy toàn bộ danh sách các ticker đang hoạt động.
        2. **Thu thập theo Vòng lặp:** Lặp qua từng ticker, gọi API tin tức của yfinance. Có một khoảng nghỉ giữa các lần gọi để tránh bị giới hạn request.

3. **Chuyển đổi Dữ liệu:** Chuẩn hóa dữ liệu JSON thô từ API, trích xuất các thông tin quan trọng (tiêu đề, tóm tắt, nhà cung cấp, link, thời gian xuất bản) và tạo một news_uuid duy nhất.
4. **Lưu trữ:** Sử dụng bulk_insert với tùy chọn on_conflict_do_nothing để ghi các tin tức mới vào bảng relevant_news, bỏ qua nếu tin tức đó đã tồn tại (dựa trên news_uuid).

3.2. Quy trình Xử lý Thời gian thực (fetch_realtime_price.py)

Quy trình này hoạt động liên tục trong suốt thời gian các thị trường mở cửa.

- **Mục tiêu:** Thu thập giá và khối lượng giao dịch gần nhất của các cổ phiếu và ghi vào Redis để phục vụ cho các phân tích intraday.
- **Logic:**
    1. **Lập lịch (Scheduling):** Một bộ điều phối chính (main_orchestrator) sử dụng thư viện schedule để kích hoạt pipeline này chạy định kỳ (ví dụ: mỗi phút hoặc mỗi 15 phút).
    2. **Lấy Metadata từ Cache:** Pipeline bắt đầu bằng việc gọi PostgreDBManager để lấy toàn bộ thông tin metadata của các ticker (đã được cache sẵn trong bộ nhớ).
    3. **Lọc Ticker theo Thị trường Mở cửa:** Lặp qua từng ticker trong cache, sử dụng thông tin timezone và open/close_time để kiểm tra xem thị trường của ticker đó có đang trong phiên giao dịch hay không. Chỉ các ticker "hợp lệ" mới được xử lý tiếp.
    4. **Thu thập Dữ liệu Real-time:** Đối với mỗi ticker hợp lệ, gọi API fast_info của yfinance để lấy giá, khối lượng và các thông tin intraday khác mới nhất.
    5. **Lưu vào Redis Stream:** Ghi dữ liệu vừa thu thập được vào Redis Stream tương ứng với ticker đó.

4. Quản lý Tương tác Cơ sở dữ liệu (db_manager.py)

- **PostgreDBManager:** Đóng vai trò là một lớp DAO (Data Access Object) cho PostgreSQL. Nó trừu tượng hóa toàn bộ logic kết nối và các câu lệnh SQL phức tạp. Đặc biệt, nó triển khai một **cơ chế cache cho metadata** để giảm thiểu số lần truy vấn vào các bảng tĩnh, giúp tăng hiệu năng cho toàn hệ thống.
- **RedisManager:** Một lớp DAO tương tự cho Redis, cung cấp các phương thức đã được trừu tượng hóa để làm việc với các cấu trúc dữ liệu của Redis (cụ thể là Streams), che giấu chi tiết triển khai khỏi các pipeline.