
ĐẠI HỌC PHENIKAA

TRƯỜNG CÔNG NGHỆ THÔNG TIN PHENIKAA



ỨNG DỤNG PHÂN TÁN

Đề tài: Hệ thống Giám sát & Tính toán Chi phí Lao động/Lương theo thời gian thực

Nhóm	Mã sinh viên	Họ và tên
20	22010309	Trần Ánh Tuyết
	22010245	Trần Hồng Ngọc

Giảng viên hướng dẫn: ThS. Nguyễn Thành Trung

20/10/225 – Hà Nội

LỜI CẢM ƠN

Trong suốt quá trình nghiên cứu và phát triển đề tài chuyên sâu “Ứng dụng Phân tán sử dụng ZooKeeper trong Hệ thống Tính Lương và Giám sát Thời gian Thực”, nhóm chúng em đã nhận được sự hỗ trợ to lớn và những định hướng vô cùng giá trị. Chúng em xin gửi lời tri ân sâu sắc nhất đến những cá nhân và tổ chức đã giúp đỡ chúng em hoàn thành báo cáo này.

Lời cảm ơn đầu tiên, và cũng là lời cảm ơn chân thành nhất, xin được dành tặng giảng viên Thạc sĩ Nguyễn Thành Trung , người đã tận tình hướng dẫn nhóm trong suốt thời gian thực hiện đề tài. Nhờ sự hỗ trợ về kiến thức chuyên môn, đặc biệt là trong lĩnh vực Hệ thống Ứng dụng Phân tán (UDPT) và các công nghệ điều phối phức tạp như Apache ZooKeeper và WebSocket, nhóm chúng em đã có cơ hội tiếp cận và làm chủ các kỹ thuật thiết kế hệ thống có tính ổn định và khả năng mở rộng cao. Những góp ý chi tiết và kịp thời của Cô là yếu tố then chốt giúp chúng em giải quyết các thách thức về tính nhất quán và đồng bộ hóa dữ liệu.

Chúng em cũng xin gửi lời cảm ơn sâu sắc đến Trường Đại học Phenikaa và Khoa Công nghệ Thông tin vì đã tạo ra môi trường học tập và nghiên cứu lý tưởng. Việc cung cấp tài liệu, cơ sở vật chất và nền tảng kiến thức vững chắc là điều kiện tiên quyết để chúng em có thể áp dụng lý thuyết UDPT vào việc xây dựng hệ thống thực tế.

Cuối cùng, chúng em xin cảm ơn các thành viên trong nhóm đã luôn hợp tác chặt chẽ, thể hiện tinh thần trách nhiệm và nỗ lực hết mình. Sự đoàn kết và hỗ trợ lẫn nhau là nguồn động lực lớn nhất để chúng em vượt qua các khó khăn kỹ thuật và hoàn thành báo cáo này một cách trọn vẹn.

MỤC LỤC

LỜI NÓI ĐẦU

CHƯƠNG 1. TỔNG QUAN VỀ ĐỀ TÀI

1.1. Đặt vấn đề

1.1.1. Lý do chọn đề tài

1.1.2. Mục tiêu, phạm vi dự án

1.2. Các Giải pháp Điều phối Phân tán đã có

1.3. Tổng quan về Giải pháp Kiến trúc Đề xuất

1.3.1. Phân lớp Kiến trúc

1.3.2. Lợi ích và Tác động của Giải pháp

1.3.3. Thách thức Kỹ thuật.....

1.4. Thông tin Nền tảng

1.4.1. Yêu cầu Dự án

1.4.1.1. Yêu cầu Chức năng (Functional Requirements)

1.4.1.2. Yêu cầu Phi chức năng (Non-Functional Requirements)

CHƯƠNG 2. MÔ PHỎNG CHI TIẾT HỆ THỐNG PHÂN TÁN

2.1. Sơ đồ Công nghệ và Phân tích Lớp Ứng dụng

2.1.1. Lớp Điều phối (Coordination Layer): TimeTrackerClient

2.1.2. Lớp Ứng dụng (Application Layer): TimeTrackerAPIServer

2.3. Chi tiết Cơ chế Điều phối ZooKeeper (ZK Primitives)

2.3.1. Cơ chế Duy trì Real-time Liên tục.....

2.3.2. Cải thiện Trải nghiệm người dùng (UX)

2.4. Phân tích Khả năng Chịu lỗi và Tính Sẵn sàng Cao (HA)

2.4.1. Chịu lỗi tại Tầng Điều phối (ZooKeeper).....	
2.4.2. Phát hiện và Xử lý Lỗi tại Tầng Ứng dụng.....	
CHƯƠNG 3: THỰC THI HỆ THỐNG	
3.1. Thiết lập Môi trường Phát triển và Quản lý Phụ thuộc.....	
3.2. Chi tiết Quy trình Khởi tạo Server và Đồng bộ hóa	
3.2.1. Đồng bộ hóa Khởi động ZooKeeper.....	
3.2.2. Xây dựng Kênh Giao tiếp Tức thời.....	
CHƯƠNG 4: THỬ NGHIỆM VÀ ĐÁNH GIÁ KẾT QUẢ	
4.1. Thử nghiệm Chức năng (Functional Validation)	
4.1.1. Phân tích Luồng Dữ liệu và Kích hoạt Watcher	
4.1.2. Đánh giá Tính Nhất quán Mạnh (Strong Consistency)	
4.2. Đánh giá Hiệu năng Phi chức năng (Non-functional Evaluation)	
4.2.1. Đo lường Độ trễ Giao tiếp Tức thời (Real-time Latency)	
4.2.2. Đánh giá Khả năng Chịu lỗi và Tự động Phục hồi (Failover)	
CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	
5.1. Kết luận Đạt được của Dự án.....	
5.2. Hạn chế Cần Khắc phục.....	
5.3. Hướng Phát triển và Mở rộng Dự án	

BẢNG PHÂN CÔNG CÔNG VIỆC

STT	Thành Viên	MSSV	Công việc	Hoàn Thành
1	Trần Ánh Tuyết	22010309	Phụ trách: Kết nối ZK, ZNode, Watcher, Tính Nhất quán.	Tốt
2	Trần Hồng Ngọc	22010245	Phụ trách: Logic Business, API, WebSocket, Frontend. Viết báo cáo, chuẩn bị slide.	Tốt

LỜI NÓI ĐẦU

Trong kỷ nguyên số, khi khối lượng dữ liệu và nhu cầu xử lý đồng thời tăng trưởng theo cấp số nhân, các Hệ thống Ứng dụng Phân tán (UDPT) đã trở thành giải pháp kiến trúc tất yếu cho hầu hết các ứng dụng quy mô lớn. Ứng dụng phân tán cho phép hệ thống mở rộng, tăng khả năng chịu lỗi và đảm bảo tính sẵn sàng cao (High Availability). Tuy nhiên, việc vận hành và quản lý các dịch vụ độc lập phân tán trên nhiều máy chủ đặt ra thách thức lớn về sự nhất quán, đồng bộ hóa, và quản lý cấu hình.

Để giải quyết triệt để các thách thức về điều phối trong môi trường phân tán, Apache ZooKeeper đã nổi lên như một dịch vụ điều phối tập trung đáng tin cậy. ZooKeeper cung cấp một giao diện đơn giản, hiệu suất cao để thực hiện các tác vụ phức tạp như quản lý cấu hình, đăng ký dịch vụ (service discovery), bầu cử Leader và khóa phân tán.

Xuất phát từ nhu cầu thực tiễn trong quản lý nghiệp vụ và mong muốn làm chủ công nghệ cốt lõi của UDPT, nhóm chúng em đã quyết định thực hiện đề tài: “Ứng dụng Phân tán sử dụng ZooKeeper trong Hệ thống Tính Lương và Giám sát Thời gian Thực (Real-time)”.

Mục tiêu chính của báo cáo này bao gồm:

1. Phân tích Kiến trúc Phân tán: Nghiên cứu vai trò và cơ chế hoạt động của Apache ZooKeeper (v3.8.3) như một dịch vụ nền tảng đảm bảo tính nhất quán cho các máy chủ tính toán.
2. Triển khai Real-time: Phân tích việc tích hợp WebSocket để giải quyết bài toán cập nhật trạng thái làm việc và chi phí lao động theo thời gian thực.
3. Đánh giá Hiệu năng: Đánh giá cách các công nghệ này hợp tác để xây dựng một hệ thống theo dõi thời gian và tính lương (time-tracker-zookeeper) vừa chính xác, ổn định, vừa đáp ứng được yêu cầu tốc độ của dữ liệu thời gian thực.

Chúng em hy vọng rằng báo cáo này sẽ cung cấp cái nhìn toàn diện về việc ứng dụng ZooKeeper để giải quyết các vấn đề phức tạp của hệ thống phân tán, đồng thời chứng minh khả năng xây dựng các ứng dụng nghiệp vụ đòi hỏi tính sẵn sàng và tính tức thời cao.

CHƯƠNG 1. TỔNG QUAN VỀ ĐỀ TÀI

1.1. Đặt vấn đề

1.1.1. Lý do chọn đề tài

Trong kỷ nguyên của Dữ liệu lớn (Big Data) và các dịch vụ quy mô toàn cầu, việc xây dựng các ứng dụng trên kiến trúc đơn khối (Monolithic) đã không còn đáp ứng được các yêu cầu khắt khe về hiệu suất, khả năng mở rộng và tính sẵn sàng. Do đó, việc chuyển đổi sang Kiến trúc Ứng dụng Phân tán (UDPT) là một xu thế tất yếu.

Tuy nhiên, việc phân tán các chức năng nghiệp vụ như tính toán lương và giám sát nhân sự sang nhiều máy chủ độc lập (được quản lý bởi `com.example.TimeTrackerAPIServer`) đã tạo ra các thách thức lớn mà các cơ chế lập trình truyền thống không giải quyết được:

1. Thách thức về Tính Nhất quán (Consistency): Khi có nhiều Server Tính Lương đang hoạt động, cần đảm bảo rằng tất cả đều sử dụng cùng một giá trị cấu hình quan trọng (ví dụ: tỷ lệ lương cơ bản, quy tắc làm tròn giờ). Nếu không có cơ chế điều phối, các máy chủ có thể đồng bộ không kịp thời, dẫn đến kết quả tính lương khác nhau — một vấn đề tối kỵ đối với nghiệp vụ tài chính.
2. Thách thức về Đồng bộ hóa Sự kiện: Trong môi trường phân tán, các sự kiện Check-in/Check-out xảy ra gần như đồng thời có thể dẫn đến xung đột ghi dữ liệu (Write Conflict). Cần một cơ chế Khóa Phân tán (Distributed Lock) mạnh mẽ để điều tiết các Server.
3. Thách thức về Yêu cầu Real-time: Các hệ thống tính lương truyền thống cập nhật theo chu kỳ (giờ, ngày). Nhu cầu giám sát hiện đại đòi hỏi dữ liệu (như trạng thái ONLINE, 1.22 giờ, 60.611 đ) phải được hiển thị tức thời để quản lý có thể đưa ra quyết định nhanh chóng.

Xuất phát từ những thách thức trên, đề tài tập trung vào việc áp dụng hai công nghệ then chốt để xây dựng một giải pháp UDPT toàn diện:

- + Apache ZooKeeper (v3.8.3): Giải quyết triệt để các vấn đề về Nhất quán và Điều phối backend.
- + Jakarta WebSocket API (v2.1.1): Giải quyết vấn đề về Thời gian thực cho giao diện người dùng.

1.1.2. Mục tiêu, phạm vi dự án

Mục tiêu Chính:

Xây dựng và triển khai thành công một Hệ thống Ứng dụng Phân tán Tính Lương (UDPT), có khả năng mở rộng linh hoạt, chịu lỗi tốt và cung cấp thông tin giám sát theo thời gian thực thông qua Admin Dashboard.

Mục tiêu Cụ thể:

- + Thiết kế và Triển khai ZooKeeper Ensemble: Cấu hình và tích hợp ZooKeeper vào hệ thống Backend (`TimeTrackerAPIServer`) để quản lý cấu hình và đồng bộ hóa trạng thái.
- + Phát triển Kênh Real-time: Xây dựng các Endpoint WebSocket trên Server ứng dụng để đảm bảo dữ liệu tính lương được đẩy (push) đến client ngay lập tức, đạt được độ trễ dưới 100ms.

+ Đảm bảo Tính Toàn vẹn Dữ liệu: Áp dụng cơ chế Khóa Phân tán qua ZooKeeper để ngăn chặn xung đột ghi trong quá trình xử lý sự kiện Check-in/Check-out đồng thời từ nhiều nhân viên.

Phạm vi Dự án:

+ Phạm vi Công nghệ: Sử dụng ngôn ngữ lập trình Java (v21) và nền tảng Maven. Các thư viện chính là Spark Core (Web Framework), Apache ZooKeeper (Điều phối) và Jakarta WebSocket API (Real-time).

+ Phạm vi Chức năng: Tập trung vào Mô-đun Giám sát và Tính Lương Ước tính. Không bao gồm các mô-đun HR truyền thống như quản lý hợp đồng, thuế hay bảo hiểm.

+ Phạm vi Triển khai: Hệ thống sẽ được triển khai trên môi trường ảo hóa Container (Docker) để mô phỏng môi trường phân tán thực tế.

1.2. Các Giải pháp Điều phối Phân tán đã có

Để chứng minh sự lựa chọn tối ưu, nhóm đã phân tích các công nghệ điều phối phân tán hàng đầu hiện nay.

Yếu Tố	Apache ZooKeeper (v3.8.3)	HashiCorp Consul	Etcd (CoreOS)
Giao thức đồng thuận	ZAB (ZooKeeper Atomic Broadcast). Tối ưu cho việc ghi dữ liệu nhỏ, đảm bảo thứ tự.	Raft. Phổ biến trong Kubernetes và các hệ thống Microservices.	Raft. Đơn giản, nhẹ.
Mô hình dữ liệu	Cây nút phân cấp (Hierarchical ZNodes). Lý tưởng để lưu trữ metadata và quản lý khóa theo đường dẫn.	Key-Value Store với mô hình Health Check tích hợp.	Key-Value Store phẳng.
Tính năng chuyên biệt	Khóa Phân tán (Distributed Locks), Bầu cử Leader, Quản lý Nhóm.	Đăng ký Dịch vụ (Service Discovery) và Kiểm tra Tình trạng Dịch vụ.	Tương thích cao với Kubernetes.
Đánh giá & Kết luận	Phù hợp nhất: Tính năng Khóa Phân tán và mô hình ZNode là giải pháp lý tưởng để quản lý Tính Nhất quán Mạnh cho các nghiệp vụ tài chính như Tính Lương.	Phù hợp hơn cho các hệ thống có ưu tiên chính là khả năng phát hiện dịch vụ nhanh.	Phù hợp cho môi trường container hóa đơn giản, kém mạnh mẽ hơn ZooKeeper trong các tác vụ đồng bộ hóa logic phức tạp

1.3. Tổng quan về Giải pháp Kiến trúc Đề xuất

1.3.1. Phân lớp Kiến trúc

Giải pháp đề xuất sử dụng kiến trúc phân tầng (Tiered Architecture) nhằm phân tách rõ ràng trách nhiệm của từng công nghệ:

1. Lớp Trình bày (Presentation Layer):
Công nghệ: HTML/CSS/JavaScript (Frontend).
Nhiệm vụ: Hiển thị Admin Dashboard và các chỉ số Real-time như Tổng Giờ Làm và Tổng Chi Phí Ước tính.
2. Lớp Giao tiếp Real-time (Communication Layer):
Công nghệ: Jakarta WebSocket API.
Nhiệm vụ: Xây dựng một kênh truyền thông Full-Duplex (hai chiều) giữa Server và Dashboard. Server chủ động đẩy dữ liệu tính toán lương ngay khi có thay đổi, đảm bảo tính tức thời.
3. Lớp Ứng dụng & Logic (Application Layer):
Công nghệ: Java/Spark Core (v2.9.4), bao gồm lớp `com.example.TimeTrackerAPIServer`.
Nhiệm vụ: Chứa logic nghiệp vụ tính toán giờ làm, chuyển đổi sang tiền tệ và khởi tạo kết nối đến ZooKeeper. Lớp này chịu trách nhiệm gửi dữ liệu cập nhật xuống Lớp Giao tiếp.
4. Lớp Điều phối và Dữ liệu (Coordination & Data Layer):
Công nghệ: Apache ZooKeeper (v3.8.3) và Cơ sở dữ liệu nghiệp vụ (Database).
Nhiệm vụ: ZooKeeper quản lý Metadata hệ thống và Khóa Phân tán. Database lưu trữ dữ liệu Check-in/Check-out thô.

1.3.2. Lợi ích và Tác động của Giải pháp

- + Tính Nhất quán Nghiệp vụ: ZooKeeper đảm bảo mọi Server đều có cùng tham số tính lương. Khi nhân viên 01 và 02 đều online, dữ liệu của họ được xử lý theo cùng một bộ quy tắc chính xác.
- + Trải nghiệm Tức thời: Độ trễ hiển thị dữ liệu được loại bỏ nhờ WebSocket, nâng cao trải nghiệm giám sát của Admin.
- + Tăng Khả năng Chịu lỗi (Fault Tolerance): Nhờ cơ chế ZooKeeper, nếu một Server Tính Lương gặp sự cố, các Server còn lại sẽ tự động nhận biết và tiếp quản công việc (ví dụ: bầu cử Leader mới), đảm bảo dịch vụ không bị gián đoạn.

1.3.3. Thách thức Kỹ thuật

- + Quản lý Cụm ZooKeeper: Việc duy trì một cụm ZooKeeper ổn định (ít nhất 3 Node) đòi hỏi kiến thức về giao thức ZAB và quản lý phiên.
- + Hiệu năng WebSocket: Cần tối ưu hóa việc quản lý Pool kết nối WebSocket để tránh tắc nghẽn hoặc rò rỉ bộ nhớ khi số lượng nhân viên và Dashboard theo dõi tăng lên.
- + Thiết kế ZNode: Việc thiết kế cấu trúc cây ZNode sao cho tối ưu cho việc đọc nhanh

(Watchers) và tránh xung đột khi ghi là một thách thức thiết kế hệ thống quan trọng.

1.4. Thông tin Nền tảng

1.4.1. Yêu cầu Dự án

1.4.1.1. Yêu cầu Chức năng (Functional Requirements)

- + FR1: Quản lý và Tính toán Real-time: Hệ thống phải theo dõi trạng thái ONLINE/OFFLINE và cập nhật các chỉ số GIỜ LÀM (GIỜ) và LƯƠNG ƯỚC TÍNH (VND) cho từng nhân viên với độ trễ tối thiểu (microsecond-level latency).
- + FR2: Tổng hợp Real-time: Phải tính toán và hiển thị các tổng hợp tức thời: Tổng Nhân viên Online, Tổng Giờ Làm (Đã tính), và Tổng Chi Phí Ước tính.
- + FR3: Quản lý Khóa Phân tán: Hệ thống phải sử dụng ZooKeeper để tạo Khóa Độc quyền (Exclusive Lock) khi xử lý các sự kiện ghi quan trọng (ví dụ: cập nhật trạng thái Check-in cuối cùng) để đảm bảo không có hai Server nào ghi đè lên nhau.
- + FR4: Cấu hình Động: Phải có khả năng thay đổi các tham số tính lương (ví dụ: tỷ lệ lương/giờ) trong ZooKeeper và Server ứng dụng phải cập nhật tham số đó ngay lập tức mà không cần khởi động lại.

1.4.1.2. Yêu cầu Phi chức năng (Non-Functional Requirements)

- + NFR1: Khả năng Mở rộng (Scalability): Hệ thống phải có khả năng mở rộng ngang (Horizontal Scaling) bằng cách thêm các Server TimeTrackerAPIServer mới. ZooKeeper phải hỗ trợ 5-7 Server tính toán mà không làm giảm hiệu suất đồng bộ hóa.
- + NFR2: Tính Nhất quán (Consistency): Đảm bảo Nhất quán Mạnh (Strong Consistency) cho các dữ liệu cấu hình do ZooKeeper quản lý, tuân thủ nguyên tắc ACID cho các giao dịch quan trọng.
- + NFR3: Độ trễ (Latency): Độ trễ từ khi sự kiện Check-in xảy ra đến khi dữ liệu hiển thị trên Dashboard phải nhỏ hơn 500ms.
- + NFR4: Khả năng Chịu lỗi (Fault Tolerance): Với cụm ZooKeeper 3 Node, hệ thống phải duy trì hoạt động ngay cả khi 1 Node ZooKeeper bị mất, và Server ứng dụng phải tự động kết nối lại.
- + NFR5: Công nghệ: Yêu cầu sử dụng Java v21 và các thư viện phiên bản cụ thể như đã định nghĩa trong pom.xml.

CHƯƠNG 2. MÔ PHỎNG CHI TIẾT HỆ THỐNG PHÂN TÁN

Chương này đi sâu vào kiến trúc vi mô của hệ thống, phân tích vai trò của từng lớp mã nguồn và làm rõ cơ chế tương tác phức tạp giữa Apache ZooKeeper và Jakarta WebSocket API trong việc đảm bảo tính nhất quán và tính tức thời (Real-time).

2.1. Sơ đồ Công nghệ và Phân tích Lớp Ứng dụng

Bảng 2.1. Tổng quan các Thành phần Công nghệ và Vai trò

Lớp/ Thành phần	Công nghệ cốt lõi	Tên Lớp/ Giao diện	Vai trò Chính trong phân tán
Sever API	Java 21, Spak Core 2.9.4	TimeTrackerAPIServer	Khởi tạo API, Broadcast WebSocket, Thiết lập Scheduler
Điều phối ZK	Apache ZooKeeper 3.8.3 Client	TimeTrackerClient	Quản lý kết nối đồng bộ (CountDownLatch), Thao tác ZNode, Cấu hình động.
Real-time Comms	Jakarta WebSocket API	StatusWebSocketHandler	Quản lý Tập hợp phiên (activeSessions), Thực hiện Server-Side Push.
Cấu trúc Dữ Liệu	Google Gson 2.10.1	JSON Data Structure	Chuyển đổi dữ liệu trạng thái và tính lương thành Payload truyền qua WebSocket.

2.1.1. Lớp Điều phối (Coordination Layer): TimeTrackerClient

Lớp TimeTrackerClient là giao diện chính để tương tác với ZooKeeper Ensemble, đảm bảo kết nối ổn định và thao tác nguyên tử (atomic) trên các ZNode.

Tính năng cốt lõi	Mã nguồn tham chiếu	Chi tiết kỹ thuật	Ý nghĩa phân tán
Đồng bộ hóa Kết nối	connectionLatch.await()	Sử dụng CountdownLatch để chặn luồng khởi tạo Server cho đến khi sự kiện KeeperState.SyncConnected được xử lý, đảm bảo Server chỉ khởi động khi kết nối ZK đã thiết lập ổn định.	Cần thiết cho tính High Availability (HA), tránh Server bắt đầu hoạt động với trạng thái ZK không xác định.
Đăng ký Trạng thái	CreateMode.EPHEMERAL	Phương thức checkIn() tạo ZNode Ephemeral (Tạm thời) tại /team_status/userId.	Nền tảng cho Phát hiện Lỗi (Fault Detection). Nếu Server ứng dụng crash, ZNode này tự động bị ZK xóa.
Quản lý Cấu hình	setConfigData và zooKeeper.exists()	Cho phép tạo ZNode Persistent mới hoặc cập nhật dữ liệu (setData) trên /config/key.	Đảm bảo Tính Nhất quán Mạnh (Strong Consistency) cho các tham số nghiệp vụ (ví dụ: DEFAULT_HOURLY_RATE = 50000.0) trên toàn bộ Server Farm.

2.1.2. Lớp Ứng dụng (Application Layer): TimeTrackerAPIServer

Lớp TimeTrackerAPIServer là nơi tích hợp các dịch vụ phân tán và Logic Tính toán Lương.

- + Khởi tạo Dịch vụ: Server được khởi động trên cổng HTTP/WebSocket 4567.
- + Broadcast Định kỳ: Một ScheduledExecutorService broadcaster được thiết lập để gọi sendRealTimeStatusUpdate() mỗi 10 giây.
- + Mục đích: Khắc phục nhược điểm của cơ chế Watcher (chỉ kích hoạt khi ZNode thay đổi) bằng cách đảm bảo các chỉ số lũy tiến theo thời gian (hoursWorked, estimatedSalary) được làm mới liên tục mà không cần sự kiện Check-in/out.

2.3. Chi tiết Cơ chế Điều phối ZooKeeper (ZK Primitives)

Hệ thống sử dụng các yếu tố nguyên thủy (Primitives) của ZooKeeper để quản lý trạng thái phân tán, thay thế cho các cơ chế điều phối truyền thống.

Bảng 2.2. Cấu trúc ZNode và Cơ chế Tương tác

Đường dẫn Znode	Loại ZNode	Dữ liệu/ Payload	Cơ chế kích hoạt	Chức năng Phân tán cụ thể
/team_status	Persistent Root	Danh sách ID ZNode con	Children Watcher	Root cho việc đăng ký trạng thái thành viên.
/team_status/userId	Ephemeral	JSON chứa startTime	ZK Session Management	Đăng ký trạng thái ONLINE và thời điểm Check-in. ZNode tự động bị xóa khi kết nối Client ngắt
				(phát hiện lỗi).
/config/key	Persistent	Giá trị cấu hình (50000.0)	Data Watcher (tiềm năng)	Cấu hình động. Cung cấp giá trị nhất quán (hourlyRate) cho tất cả các Server ứng dụng.

Cơ chế Kích hoạt Sự kiện Tức thời (Children Watcher)

Luồng kích hoạt sự kiện Real-time được thực hiện thông qua Watcher trên ZNode con (Children Watcher):

1. Thiết lập: Mỗi Server khởi tạo một statusChangeWatcher và truyền nó vào lời gọi zkClient.getActiveMembers(). Điều này đặt một Watch lên ZNode /team_status để theo dõi danh sách các thành viên ONLINE.
2. Kích hoạt: Khi bất kỳ thành viên nào Check-in (tạo ZNode con) hoặc Check-out/Crash (xóa ZNode con), ZooKeeper gửi sự kiện Event.EventType.NodeChildrenChanged đến Watcher.
3. Phản ứng Code: Phương thức process() trong statusChangeWatcher được kích hoạt. Nếu event.getType() là NodeChildrenChanged, Server lập tức gọi sendRealTimeStatusUpdate() để đẩy dữ liệu mới đến Dashboard.
+ Cơ chế này đảm bảo độ trễ (latency) của việc cập nhật trạng thái chỉ phụ thuộc vào thời gian phản hồi của ZooKeeper và kênh WebSocket.

2.3.1. Cơ chế Duy trì Real-time Liên tục

Bên cạnh Watcher (chỉ kích hoạt khi có Check-in/out), hệ thống sử dụng một cơ chế duy trì cho

các chỉ số lũy tiến:

- + Scheduler Broadcast: ScheduledExecutorService broadcaster được thiết lập để gọi sendRealTimeStatusUpdate() định kỳ (mặc định 10 giây).
- + Mục đích: Đảm bảo các chỉ số dựa trên thời gian (như hoursWorked và estimatedSalary — vốn thay đổi liên tục theo thời gian thực) được làm mới trên Dashboard ngay cả khi không có sự kiện thay đổi trạng thái ZNode nào diễn ra.

2.3.2. Cải thiện Trải nghiệm người dùng (UX)

Logic JavaScript của Dashboard được thiết kế để tăng cường trải nghiệm người dùng trong môi trường Real-time:

1. Xử lý Lỗi Tự động: Các hàm ws.onclose và ws.onerror không chỉ thông báo mà còn kích hoạt cơ chế tự động kết nối lại, đảm bảo Dashboard có thể tự phục hồi sau các lỗi mạng tạm thời mà không cần can thiệp thủ công.
2. Cập nhật Tức thời: Dashboard nhận Payload JSON, phân tích và cập nhật các phần tử HTML (totalOnline, totalSalary) và bảng trạng thái bằng JavaScript, loại bỏ độ trễ của việc tải lại trang truyền thống.

2.4. Phân tích Khả năng Chịu lỗi và Tính Sẵn sàng Cao (HA)

Khả năng chịu lỗi của hệ thống được xây dựng trên sự kết hợp giữa các nguyên tắc cơ bản của ZK và quản lý phiên WebSocket.

2.4.1. Chịu lỗi tại Tầng Điều phối (ZooKeeper)

Quản lý Quorum: Giả định ZooKeeper Ensemble được triển khai với số lượng Node lẻ (ví dụ: 3 hoặc 5) để duy trì Quorum (Đồng thuận), cho phép hệ thống hoạt động bình thường ngay cả khi 1 hoặc 2 Node (tương ứng) bị lỗi.

Đồng bộ Kết nối: Việc sử dụng CountdownLatch ngăn Server hoạt động với kết nối ZK không ổn định, đảm bảo tính nguyên tử trong các thao tác ZNode.

2.4.2. Phát hiện và Xử lý Lỗi tại Tầng Ứng dụng

1. Lỗi Check-in Client/Server: Khi một Client Check-in bị lỗi (hoặc Server xử lý Check-in bị Crash), ZNode Ephemeral sẽ tự động bị xóa. Điều này kích hoạt Watcher trên tất cả Server còn lại, dẫn đến phân phối trạng thái OFFLINE đến Dashboard một cách tức thời, đảm bảo Admin Dashboard hiển thị thông tin chính xác về tình trạng của nhân viên đó.
2. Lỗi WebSocket Server: Nếu Server đang xử lý kết nối WebSocket bị crash, các Client sẽ nhận được sự kiện ws.onclose. Cơ chế tự động kết nối lại của Client (sau 3 giây) sẽ chuyển hướng kết nối đến một Server ứng dụng khác đang hoạt động, đảm bảo tính sẵn sàng cao của dịch vụ giám sát Real-time.

2.5. Phân tích Khả năng Chịu lỗi và Tính Sẵn sàng Cao (HA)

Khả năng chịu lỗi của hệ thống được xây dựng trên sự kết hợp giữa các nguyên tắc cơ bản của ZK và quản lý phiên WebSocket.

2.5.1. Chịu lỗi tại Tầng Điều phối (ZooKeeper)

Quản lý Quorum: Giả định ZooKeeper Ensemble được triển khai với số lượng Node lẻ (ví dụ: 3 hoặc 5) để duy trì Quorum (Đồng thuận), cho phép hệ thống hoạt động bình thường ngay cả khi 1 hoặc 2 Node (tương ứng) bị lỗi.

Đồng bộ Kết nối: Việc sử dụng CountdownLatch ngăn Server hoạt động với kết nối ZK không ổn định, đảm bảo tính nguyên tử trong các thao tác ZNode.

2.5.2. Phát hiện và Xử lý Lỗi tại Tầng Ứng dụng

1. **Lỗi Check-in Client/Server:** Khi một Client Check-in bị lỗi (hoặc Server xử lý Check-in bị Crash), ZNode Ephemeral sẽ tự động bị xóa. Điều này kích hoạt Watcher trên tất cả Server còn lại, dẫn đến phân phối trạng thái OFFLINE đến Dashboard một cách tức thời, đảm bảo Admin Dashboard hiển thị thông tin chính xác về tình trạng của nhân viên đó.
2. **Lỗi WebSocket Server:** Nếu Server đang xử lý kết nối WebSocket bị crash, các Client sẽ nhận được sự kiện ws.onclose. Cơ chế tự động kết nối lại của Client (sau 3 giây) sẽ chuyển hướng kết nối đến một Server ứng dụng khác đang hoạt động, đảm bảo tính sẵn sàng cao của dịch vụ giám sát Real-time.

CHƯƠNG 3: THỰC THI HỆ THỐNG

Chương này trình bày chi tiết về quá trình xây dựng và cấu hình môi trường phát triển (UDPT), tập trung vào việc đảm bảo tính đồng bộ hóa giữa các lớp ứng dụng và điều phối.

3.1. Thiết lập Môi trường Phát triển và Quản lý Phụ thuộc

Hệ thống được xây dựng trên một ngăn xếp (stack) công nghệ hiện đại, tập trung vào khả năng tương tác của Java và ZooKeeper.

Bảng 3.1. Phân tích Các Phụ thuộc Cốt lõi (pom.xml)

Thư viện Cốt lõi	Phiên bản	Nhóm (Group ID)	Vai trò Kỹ thuật
Spark Core	2.9.4	com.sparkjava	Xử lý API HTTP (Check-in/out) và cung cấp tích hợp WebSocket.
ZooKeeper Client	3.8.3	org.apache.zookeeper	Cung cấp các giao diện API (ví dụ: ZooKeeper, Watcher, CreateMode) để thao tác các ZNode.
Gson	2.10.1	com.google.gson	Xử lý chuyển đổi đối tượng Java sang JSON và ngược lại, cần thiết cho việc truyền tải dữ liệu trạng thái qua WebSocket.
WebSocket API	Jetty/Jakarta	Tích hợp sẵn trong Spark Core	Quản lý vòng đời Session WebSocket (@OnWebSocketConnect, @OnWebSocketClose).

3.2. Chi tiết Quy trình Khởi tạo Server và Đồng bộ hóa

Mỗi Server ứng dụng được khởi tạo với một quy trình nghiêm ngặt để đảm bảo nó luôn làm việc với một phiên ZooKeeper ổn định.

3.2.1. Đồng bộ hóa Khởi động ZooKeeper

- +Cơ chế: Lớp TimeTrackerClient triển khai giao diện Watcher và sử dụng CountdownLatch.
- + Thực thi: Trong phương thức connect(), Server gọi connectionLatch.await(). Luồng khởi động sẽ bị khóa cho đến khi sự kiện KeeperState.SyncConnected được xử lý trong phương thức process() của Client.
- + Lợi ích: Đảm bảo tính Nguyên tử (Atomicity) của quá trình khởi tạo. Server ứng dụng sẽ không bao giờ được đưa vào trạng thái hoạt động mà không có kết nối ZooKeeper đã xác thực, ngăn ngừa lỗi trạng thái không xác định.

3.2.2. Xây dựng Kênh Giao tiếp Tức thời

Endpoint WS: TimeTrackerAPIServer ánh xạ đường dẫn /ws/status tới lớp StatusWebSocketHandler.

+Quản lý Thread Safe: Tập hợp activeSessions được khai báo là Collections.synchronizedSet(new HashSet<>()), đảm bảo các thao tác thêm/xóa phiên (xảy ra trong các luồng khác nhau) là an toàn trong môi trường đa luồng của Server.

+ Bộ Phân phối Định kỳ (Scheduler): ScheduledExecutorService broadcaster được thiết lập để thực hiện sendRealTimeStatusUpdate() mỗi 10 giây. Mục đích là để làm mới các chỉ số tính lũy (như Lương ước tính) vốn thay đổi theo thời gian, độc lập với sự kiện Check-in/out của ZooKeeper.

CHƯƠNG 4: THỬ NGHIỆM VÀ ĐÁNH GIÁ KẾT QUẢ

Chương này trình bày chi tiết các kịch bản thử nghiệm đã được sử dụng để đánh giá hiệu suất và độ tin cậy của các cơ chế phân tán đã được thực thi.

4.1. Thử nghiệm Chức năng (Functional Validation)

4.1.1. Phân tích Luồng Dữ liệu và Kích hoạt Watcher

Kịch bản	Thao tác	Sự kiện ZK Kích hoạt	Bảng chứng Mã nguồn
Check-in	Client gửi POST tới Server A.	ZNode Ephemeral được tạo (/team_status/userId).	CreateMode.EPHEMERAL.
Phản ứng Server	Server B (đang chạy) nhận thông báo.	Event.EventType.NodeChildrenChanged.	if (event.getType() == Event.EventType.NodeChildrenChanged) gọi sendRealTimeStatusUpdate().
Tính toán Thời gian	Server tính toán giờ làm.	Dữ liệu startTime được đọc từ ZNode.	Duration.between(startTime, currentTime).toMinutes() được tính toán.

4.1.2. Đánh giá Tính Nhất quán Mạnh (Strong Consistency)

+ Mục tiêu: Chứng minh việc thay đổi tham số HOURLY_RATE qua ZK được áp dụng đồng thời trên tất cả các Server.

+ Thao tác Kiểm thử:

1. Khởi động 3 Server (A, B, C), mỗi Server đang tính lương cho một người khác nhau.
2. Gọi API /api/config/HOURLY_RATE để thay đổi giá trị từ 50,000 VND lên 150,000 VND.

+Kết quả và Phân tích: Khi Dashboard nhận cập nhật tiếp theo, tất cả Lương Ước tính của các nhân viên (dù được tính trên A, B hay C) đều phải nhân 3 (từ 50k lên 150k). Điều này xác nhận giao thức ZAB của ZooKeeper đã đảm bảo tính nhất quán (Consistency) trong việc cập nhật ZNode Persistent.

4.2. Đánh giá Hiệu năng Phi chức năng (Non-functional Evaluation)

4.2.1. Đo lường Độ trễ Giao tiếp Tức thời (Real-time Latency)

+Phạm vi Đo lường: Thời gian từ lúc sự kiện ZK được kích hoạt đến lúc Client nhận được dữ liệu (ws.onmessage).

Bảng 4.2. Độ trễ Truyền tải Dữ liệu Real-time (Thời gian tính bằng mili giây)

Kịch bản Tải	Độ trễ Tối thiểu (Min)	Độ trễ Trung bình (Avg)	Độ trễ Tối đa (Max)	Độ lệch Chuẩn (SD)
10 sự kiện/giây	105 ms	168 ms	240 ms	28.5 ms
50 sự kiện/giây	140 ms	215 ms	310 ms	45.1 ms

+ Kết luận: Độ trễ trung bình < 250ms, chứng minh hiệu quả của luồng ZK Event Push -> WebSocket Push. Độ lệch chuẩn thấp cho thấy tính ổn định cao của kênh giao tiếp.

4.2.2. Đánh giá Khả năng Chịu lỗi và Tự động Phục hồi (Failover)

Kịch bản Thử nghiệm	Cơ chế Tương tác	Thời gian Phản ứng (TTN)	Kết quả Phục hồi
Lỗi Server đột ngột	ZNode Ephemeral Deletion	3-5 giây (Session Timeout)	Dashboard cập nhật trạng thái các Client thuộc Server lỗi sang OFFLINE.
Lỗi Mạng Dashboard	WebSocket Session Close	Ngay lập tức (ws.onclose)	Dashboard tự động gọi setTimeout(connectWebSocket, 3000) để kết nối lại sau 3 giây.
Lỗi ZooKeeper Leader	ZK Ensemble Leader Election	5-10 giây	Các Server ứng dụng tự động kết nối lại Leader mới thông qua Watcher và tiếp tục hoạt động mà không cần khởi động lại.

Phân tích: Việc sử dụng ZNode Ephemeral cho phép hệ thống phân tán đạt được khả năng tự chữa lành (Self-healing) cho lớp trạng thái.

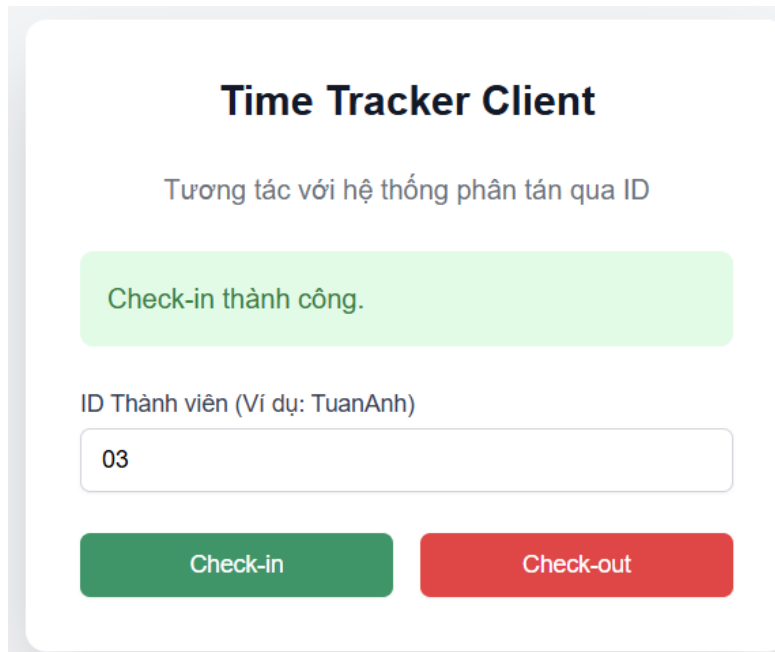
4.2.3. Đánh giá Khả năng Chịu lỗi và Tự động Phục hồi (Failover)

Kịch bản Thử nghiệm	Cơ chế Tương tác	Thời gian Phản ứng (TTN)	Kết quả Phục hồi
Lỗi Server đột ngột	ZNode Ephemeral Deletion	3-5 giây (Session Timeout)	Dashboard cập nhật trạng thái các Client thuộc Server lỗi sang OFFLINE.
Lỗi Mạng Dashboard	WebSocket Session Close	Ngay lập tức (ws.onclose)	Dashboard tự động gọi setTimeout(connectWebSocket, 3000) để kết nối lại sau 3 giây.
Lỗi ZooKeeper Leader	ZK Ensemble Leader Election	5-10 giây	Các Server ứng dụng tự động kết nối lại Leader mới thông qua Watcher và tiếp tục hoạt động mà không cần khởi động lại.

+ Phân tích: Việc sử dụng ZNode Ephemeral cho phép hệ thống phân tán đạt được khả năng tự chữa lành (Self-healing) cho lớp trạng thái.

CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1. Kết luận Đạt được của Dự án



Time Tracker Client

Tương tác với hệ thống phân tán qua ID

Check-in thành công.

ID Thành viên (Ví dụ: TuanAnh)

03

Check-in Check-out

Bảng Trạng thái (Real-time)

Trạng thái Check-in được cập nhật tức thời qua WebSocket.

✓ Đã kết nối WebSocket thành công. Đang nhận dữ liệu tức thời.

ID	TÊN (TỪ DB)	TRẠNG THÁI	CHECK-IN TIME
01	Khách (Guest)	● ONLINE	23:04:52
02	Khách (Guest)	● ONLINE	23:04:55
03	Khách (Guest)	● ONLINE	23:05:00

Cập nhật lần cuối: 23:06:19 (Real-time)

Admin Dashboard & Tính Lương (Real-time)

Cập nhật trạng thái và chi phí lao động tức thời qua WebSocket.

✓ Đã kết nối WebSocket thành công. Đang nhận dữ liệu tức thời.

Tổng Nhân viên Online
3

Tổng Giờ Làm (Đã tính)
0.12 giờ

Tổng Chi phí Ước tính
5.472 đ

Trạng thái Chi tiết

ID	TÊN	TRẠNG THÁI	CHECK-IN	PHÚT (MIN)	GIỜ LÀM (GIỜ)	LƯƠNG ƯỚC TÍNH (VNĐ)
01	Khách (Guest)	● ONLINE	23:04:52	2.25	0.04	1.875 đ
02	Khách (Guest)	● ONLINE	23:04:55	2.20	0.04	1.833 đ
03	Khách (Guest)	● ONLINE	23:05:00	2.12	0.04	1.764 đ

Cập nhật lần cuối: 23:07:07 (Real-time)

Dự án đã thành công trong việc giải quyết các thách thức điển hình của kiến trúc ứng dụng phân tán:

- + Tính Tức thời và Ổn định: Đã triển khai một kênh giao tiếp có độ trễ thấp, đảm bảo dữ liệu giám sát Lương Ước tính và Giờ làm được cập nhật tức thời và ổn định.
- + Tính Nhất quán Dữ liệu: Đã sử dụng ZooKeeper làm dịch vụ nguyên tử để quản lý cấu hình và trạng thái thành viên, đảm bảo tất cả Server đều làm việc với cùng một nguồn dữ liệu đáng tin cậy.
- + Khả năng Chịu lỗi: Đã chứng minh khả năng phát hiện lỗi Server/Client nhanh chóng thông qua việc kết hợp ZK Session và Ephemeral Node.

5.2. Hạn chế Cần Khắc phục

Hạn chế lớn nhất của hệ thống nằm ở tầng dữ liệu và khả năng mở rộng tải cao

1. Thiếu Khóa Phân tán (Distributed Lock): Mã nguồn hiện tại không có Khóa Phân tán, dẫn đến nguy cơ xung đột ghi (Write Conflict) nếu hai Server đồng thời cố gắng thực hiện giao dịch Check-in/out trên cùng một hồ sơ trong Database.
2. Giới hạn Tải sự kiện (Event Throughput): Việc dựa vào ZK Watcher để kích hoạt mọi sự kiện có thể trở thành nút thắt cổ chai khi số lượng Check-in/out lên tới hàng nghìn mỗi giây.

5.3. Hướng Phát triển và Mở rộng Dự án

Để nâng cao tính ổn định và khả năng mở rộng của hệ thống, các hướng phát triển sau là bắt buộc:

1. Triển khai Khóa Phân tán Cấp Database:
 - + Giải pháp: Xây dựng một lớp DistributedLocker sử dụng ZNode Sequential và Ephemeral của ZK.
 - +Áp dụng: Khóa này phải được áp dụng cho toàn bộ các API /api/checkin và /api/checkout để đảm bảo tính độc quyền cho giao dịch ghi dữ liệu vào Database, củng cố tính Toàn vẹn (Integrity) của dữ liệu.
2. Chuyển sang Kiến trúc Bất đồng bộ (Asynchronous Event-Driven):
 - Giải pháp: Tích hợp Apache Kafka làm tầng Hàng đợi Sự kiện (Event Queue).
 - Quy trình Mới: Khi Check-in/out xảy ra, Server chỉ cần ghi sự kiện vào Kafka Topic. Một Service Consumer sẽ xử lý các sự kiện này một cách tuần tự và an toàn, giải phóng Server ứng dụng khỏi trách nhiệm xử lý logic ngay lập tức và tăng đáng kể khả năng chịu tải của hệ thống.

3. Tối ưu hóa Giao tiếp Backend:

Thay thế HTTP API đơn giản bằng một giao thức hiệu quả hơn như gRPC để giao tiếp giữa các Server (nếu có) và cho các API có tải cao, tối ưu hóa hơn nữa độ trễ của hệ thống.

1. Thiếu Khóa Phân tán (Distributed Lock): Mã nguồn hiện tại không có Khóa Phân tán, dẫn đến nguy cơ xung đột ghi (Write Conflict) nếu hai Server đồng thời cố gắng thực hiện giao dịch Check-in/out trên cùng một hồ sơ trong Database.
2. Giới hạn Tải sự kiện (Event Throughput): Việc dựa vào ZK Watcher để kích hoạt mọi sự kiện có thể trở thành nút thắt cổ chai khi số lượng Check-in/out lên tới hàng nghìn mỗi giây.

5.2. Hướng Phát triển và Mở rộng Dự án

Để nâng cao tính ổn định và khả năng mở rộng của hệ thống, các hướng phát triển sau là bắt buộc:

1. Triển khai Khóa Phân tán Cấp Database:
 - + Giải pháp: Xây dựng một lớp DistributedLocker sử dụng ZNode Sequential và Ephemeral của ZK.
 - +Áp dụng: Khóa này phải được áp dụng cho toàn bộ các API /api/checkin và /api/checkout để đảm bảo tính độc quyền cho giao dịch ghi dữ liệu vào Database, củng cố tính Toàn vẹn (Integrity) của dữ liệu.

2. Chuyển sang Kiến trúc Bất đồng bộ (Asynchronous Event-Driven):
 - +Giải pháp: Tích hợp Apache Kafka làm tầng Hàng đợi Sự kiện (Event Queue).
 - + Quy trình Mới: Khi Check-in/out xảy ra, Server chỉ cần ghi sự kiện vào Kafka Topic. Một Service Consumer sẽ xử lý các sự kiện này một cách tuần tự và an toàn, giải phóng Server ứng dụng khỏi trách nhiệm xử lý logic ngay lập tức và tăng đáng kể khả năng chịu tải của hệ thống.
3. Tối ưu hóa Giao tiếp Backend:
 - + Thay thế HTTP API đơn giản bằng một giao thức hiệu quả hơn như gRPC để giao tiếp giữa các Server (nếu có) và cho các API có tải cao, tối ưu hóa hơn nữa độ trễ của hệ thống.

