JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 31 No. 5, pp. 11-27, May 2026
https://doi.org/10.9708/jksci.2026.31.05.011

End-to-End Latency Optimization and Resource Trade-off Analysis in Large-Scale Real-Time Data Pipelines

Jiman Cha*, Choong-hee Cho**

*Student, Division of Computer Science and Engineering, Sahmyook University, Seoul, Korea
**Professor, Division of Computer Science and Engineering, Sahmyook University, Seoul, Korea

[Abstract]

Large-scale services widely use distributed pipelines for real-time log processing. However, default buffering policies, although intended to protect system resources, can create bottlenecks that degrade real-time performance. This study constructed a large-scale distributed load environment and tracked end-to-end latency from log generation to final data warehouse loading at the millisecond level. Experiments were conducted across a five-stage optimization scenario by adjusting buffer sizes, wait times, and scan intervals. The results show that a low-latency configuration can reduce explicit buffering delay, but may increase packet-level overhead and degrade throughput. In contrast, the proposed hybrid configuration does not aim for absolute optimality in a single metric; instead, it applies cross-layer tuning to mitigate ingestion-layer traffic variability while minimizing downstream transmission delays. Under the evaluated conditions, the hybrid configuration achieved the lowest P95 latency, prevented pipeline collapse in micro-batch environments, and maintained throughput exceeding approximately 300 events per second.

▸ **Key words**: Data Pipeline, End-to-End Latency, Throughput, Distributed Systems, Real-time Processing, Micro-Batching

[요 약]

대규모 서비스 환경에서는 실시간 로그 처리를 위해 분산 파이프라인이 널리 사용된다. 그러나 각 컴포넌트의 기본 버퍼링 정책은 시스템 리소스를 보호하는 동시에 실시간성을 저해하는 병목을 유발할 수 있다. 본 연구는 대규모 분산 부하 환경을 구축하고, 로그 발생부터 최종 데이터 웨어하우스 적재까지의 종단간 지연 시간을 밀리초 단위로 추적하는 시스템을 설계하였다. 이를 바탕으로 버퍼 크기, 대기 시간, 스캔 주기 등을 조절한 5단계 최적화 시나리오를 실험하였다. 실험 결과, 저지연 설정은 명시적 버퍼링 지연을 감소시킬 수 있으나 패킷 단위 오버헤드 증가와 처리량 저하를 유발하는 한계를 보였다. 반면, 제안한 하이브리드 설정은 단일 지표의 절대적 최적화보다 수집 계층의 트래픽 변동 완화와 후위 계층의 불필요한 전송 대기 최소화를 위한 계층 간 통합 튜닝 원리에 기반한다. 평가 결과, 해당 설정은 전체 시나리오 중 가장 낮은 P95 지연 시간을 기록하였으며, 마이크로 배치 환경의 파이프라인 붕괴를 방지하면서 초당 약 300건 이상의 처리량을 유지하였다.

▸ **주제어**: 데이터 파이프라인, 지연 시간, 처리량, 분산 시스템, 실시간 처리, 마이크로 배치

• First Author: Jiman Cha, Corresponding Author: Choong-hee Cho
*Jiman Cha (cjm042352@gmail.com), Division of Computer Science and Engineering, Sahmyook University
**Choong-hee Cho (cch@syu.ac.kr), Division of Computer Science and Engineering, Sahmyook University
• Received: 2026. 04. 17, Revised: 2026. 04. 28, Accepted: 2026. 05. 13.

Copyright © 2026 The Korea Society of Computer and Information
http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945
www.kci.go.kr

I. Introduction

마이크로서비스 아키텍처와 클라우드 컴퓨팅의 발전으로 애플리케이션에서 발생하는 로그 데이터의 양이 기하급수적으로 증가하고 있다[1]. 이러한 대규모 데이터를 지연 없이 수집하고 분석하는 것은 비즈니스 의사결정과 시스템 이상 탐지를 위한 핵심 기반 기술이다[2]. 현재 산업계에서는 주로 ELK(Elasticsearch, Logstash, Kibana) 스택과 분산 메시징 시스템인 Kafka, 그리고 Spark Streaming을 결합한 통합 데이터 파이프라인을 구축하여 실무에 적용하고 있다[3].

그러나 이러한 오픈소스 컴포넌트들은 기본적으로 I/O 오버헤드를 줄이고 처리량을 극대화하기 위해, 데이터를 일정량 모아서 전송하는 버퍼링(Buffering) 및 마이크로배치(Micro-batching) 정책을 기본값으로 채택하고 있다[4]. 이는 시스템 리소스 절감에는 유리하지만, 데이터가 각 파이프라인 노드에 체류하는 시간을 증가시켜 전체 End-to-End(E2E) 지연 시간을 악화시킨다. 반대로 지연 시간을 최소화하기 위해 대기 시간을 완전히 제거할 경우, 잦은 I/O 요청으로 인한 CPU 컨텍스트 스위칭(context switching)과 네트워크 부하가 폭증하여 시스템 전체가 다운되는 치명적인 상충관계가 발생한다. 이러한 상충관계를 완화하기 위한 선행 연구들은 크게 세 가지 방향으로 진행되어 왔다.

첫째, Apache Kafka를 중심으로 한 개별 컴포넌트의 성능 예측 및 처리량 한계에 대한 실증적 평가가 이루어졌으며[4]-[6], 클라우드 네이티브 및 IoT 환경 등 다양한 워크로드에서의 성능 분석이 수행되었다[7], [8].

둘째, 클러스터 구조 변경 없이 설정값 커스터마이징을 통한 Kafka 배포 최적화[9]와 상태 기반 스트림 처리에서의 처리량 향상 기법[10], 그리고 Prometheus·Grafana를 활용한 종합 모니터링 방안[11]이 제안되었다.

셋째, 실시간 처리의 신뢰성을 확보하기 위해 데이터 유입량에 동적으로 반응하는 반응형 배치 전략[12]이 연구되었다. 그러나 이러한 연구들은 대부분 단일 프레임워크(주로 Kafka)나 특정 브로커 계층 내부의 최적화에 집중되어 있어, 로그 수집기-데이터 병합기-메시지 브로커-스트림 처리 엔진으로 이어지는 이기종 컴포넌트가 직렬로 연결된 파이프라인 전체 관점에서 전위 계층의 설정 변화가 후위 컴포넌트에 미치는 연쇄적 영향을 End-to-End 관점에서 실증적으로 분석한 연구는 여전히 부족하다.

본 연구에서는 이러한 문제를 해결하기 위해 분산 트래픽 생성 환경을 구축하고 파이프라인 전 구간의 지연 시간을 세밀하게 분해하여 측정할 수 있는 테스트베드를 구현하였다. 본 연구의 차별성은 새로운 단일 튜닝 파라미터나 신규 컴포넌트를 제안하는 데 있지 않다. 본 연구는 로그 수집기, 데이터 병합기, 메시지 브로커, 스트림 처리 엔진이 직렬로 연결된 이기종 파이프라인에서 전위 계층의 버퍼링 및 스캔 정책이 후위 계층의 큐잉 지연으로 전파되는 과정을 종단 간 관점에서 분해 측정하고, 이를 바탕으로 계층별 역할에 따라 서로 다른 튜닝 원리를 적용하는 계층 간 통합 설계 원리를 제시한다는 점에 의의가 있다.

본 논문의 핵심 기여는 다음과 같다.

첫째, Nginx-Filebeat-Logstash-Kafka-Spark-Hive로 구성된 이기종 파이프라인에서 로그 발생부터 Hive 적재까지의 End-to-End 지연 시간을 구간별로 분해 측정하는 추적 체계를 구현하였다. 단순한 컴포넌트 단위 성능 측정을 넘어, 데이터 발생부터 최종 적재까지의 전 구간 지연 시간을 밀리초 단위로 분해함으로써 병목이 발생하는 계층과 그 원인을 식별할 수 있도록 하였다.

둘째, Zero-Wait, Micro-Batching, Collection-Layer I/O Control과 같은 단일 전략이 각각 지연 감소, 네트워크 패킷 증가, 처리량 스로틀링, 트래픽 스파이크, 큐잉 누적이라는 서로 다른 실패 양상을 보일 수 있음을 실험적으로 보였다. 이를 통해 지연 시간과 처리량, 네트워크 패킷 수, 시스템 안정성 간의 상충관계를 정량적으로 분석하였다.

셋째, 수집 계층에서의 트래픽 변동 완화와 후위 가공 및 전송 계층에서의 전송 대기 최소화를 결합한 계층 간 통합 기반의 하이브리드 설계 원리를 제안하였다. 제안된 전략은 계층별 역할에 따른 차별화된 튜닝을 통해 기존 단일 최적화 접근법의 한계를 극복하고, 실험 환경에서 P95 지연 시간과 처리 안정성 간의 균형을 효과적으로 향상시켰다.

본 논문의 구성은 다음과 같다. 2장에서는 분산 스트림 처리 시스템의 성능 평가 및 파라미터 튜닝에 관한 관련 연구를 정리하고 본 연구와의 차별점을 제시한다. 3장에서는 전체 파이프라인의 아키텍처와 각 컴포넌트의 역할, 그리고 End-to-End 지연 시간 측정 방법론을 설명한다. 4장에서는 실험을 위한 클라우드 기반 환경과 부하 생성 방식, 평가 지표를 기술하고, 5장에서는 5단계의 최적화 시나리오에 대한 실험 결과를 바탕으로 지연 시간과 처리량·자원 사용량 간의 상충관계를 분석한다. 끝으로 6장에서는 본 연구의 결론과 한계, 그리고 향후 연구 방향을 제시한다.

www.kci.go.kr

II. Related works

대규모 데이터 처리를 위해 Apache Kafka를 중심으로 한 스트림 처리 시스템의 성능 평가 및 예측에 관한 연구가 활발히 진행되어 왔다. Wu 외[4]는 Kafka 메시징 시스템의 성능을 예측하기 위한 분석 모델을 제안하였으며, Vyas 외[5]와 Hesse 외[6]는 실시간 데이터 스트리밍 플랫폼으로서 Kafka의 삽입 속도와 한계 처리량을 실증적으로 평가하였다. 또한 Pelle 외[7]와 Padmanaban 외[8]은 클라우드 네이티브 및 IoT 환경 등 다양한 분산 워크로드에서 Kafka 기반 파이프라인의 성능을 분석하였다. 이러한 연구들은 Kafka의 처리량, 삽입 성능, 배포 환경별 성능 특성을 이해하는 데 중요한 기여를 하였으나, 주로 Kafka라는 단일 프레임워크 또는 브로커 계층의 성능 평가에 초점을 두고 있어, 여러 이기종 컴포넌트가 직렬로 연결된 전체 파이프라인의 병목 전파 현상을 설명하는 데에는 한계가 있다.

파이프라인 환경에서 처리량과 지연 시간을 최적화하기 위한 설정 튜닝 및 모니터링 연구도 이어지고 있다. Das 외[9]는 클러스터 구조를 변경하지 않고 설정값 커스터마이징만으로 Kafka 배포 환경을 최적화할 수 있음을 보였으며, Adila 외[10]는 상태 기반 스트림 처리 환경에서 데이터 정합성과 처리량을 향상시키기 위한 최적화 기법을 제안하였다. Aung 외[11]은 Grafana와 Prometheus 등을 활용하여 Kafka 파이프라인의 성능 지표를 종합적으로 모니터링하는 방안을 제시하였다. 그러나 이러한 연구들은 대부분 특정 브로커 계층의 설정 튜닝이나 모니터링 체계 구축에 초점을 두고 있으며, 전위 수집 계층의 설정 변화가 가공, 전송, 처리, 적재 계층으로 연쇄적으로 전파되는 과정을 End-to-End 관점에서 정량적으로 분석하지는 못하였다.

실시간 스트림 처리 환경에서는 지연 시간과 처리량 간의 상충관계를 완화하기 위해 배치 크기 조절, 백프레셔(backpressure), 동적 자원 조정 기반의 연구도 수행되어 왔다. Wu 외[12]는 데이터 유입량과 시스템 상태에 따라 배치 크기를 동적으로 조정하는 reactive batching 전략을 제안하여 실시간 스트림 처리의 신뢰성을 향상시키고자 하였다. 또한 Carbone 외[13]는 Apache Flink의 스트림 및 배치 통합 처리 구조와 실행 모델을 제시하면서, 스트림 처리 프레임워크에서 처리 속도 차이와 연속 데이터 흐름을 관리하기 위한 구조적 기반을 설명하였다. Siachamis 외[14]는 동적 워크로드 환경에서 스트림 처리 시스템의 autoscaling 기법들을 비교·평가하였다. 이러한 접근은 런타임 상태에 따라 처리 속도 또는 자원 할당을 적응적으로 조정할 수 있다는 장점이 있으나, 프레임워크 내부의 제어 메커니즘, 별도의 제어 루프, 또는 추가 자원 확장을 전제로 한다. 반면 본 연구는 클러스터 구조 변경이나 런타임 제어기 도입 없이, 이미 운영 중인 이기종 파이프라인의 정적 설정값 조정을 통해 End-to-End 지연 시간과 리소스 상충관계를 분석한다는 점에서 차별성을 갖는다.

종합하면, 기존 연구는 크게 개별 컴포넌트 성능 분석, 정적 파라미터 튜닝, 모니터링 기반 병목 관찰, 동적 배치 및 backpressure 제어 연구로 구분할 수 있다. 반면 본 연구는 클러스터 구조 변경이나 런타임 제어기 도입 없이, 이미 운영 중인 파이프라인 컴포넌트의 정적 설정값 조정을 통해 End-to-End 지연 시간과 리소스 상충관계를 실증적으로 분석한다는 점에서 차별성을 갖는다. 특히 Nginx, Filebeat, Logstash, Kafka, Spark, Hive로 이어지는 전체 구간의 End-to-End 지연 시간을 밀리초 단위로 직접 추적하고, 수집 계층의 파일 스캔 주기와 I/O 부하가 후위 브로커 및 처리 계층의 큐잉 지연에 미치는 연쇄적 병목 현상을 분석한다. 이를 통해 본 연구는 단일 컴포넌트 최적화가 아니라, 이기종 컴포넌트가 직렬로 연결된 실시간 데이터 파이프라인에서 계층별 파라미터 조정이 전체 성능에 미치는 영향을 설명하는 실증적 가이드라인을 제시한다.

III. Data Pipeline Overview and Component Analysis

오늘날 마이크로서비스 아키텍처와 클라우드 네이티브 환경의 확산으로 단일 요청에도 수많은 분산 서비스 간의 통신 로그와 시스템 지표가 기하급수적으로 발생하고 있다. 과거의 배치 기반 처리 방식으로는 트래픽 폭증이나 시스템 이상 징후를 골든 타임 내에 탐지하기 어렵다. 따라서 애플리케이션에서 발생한 로그 데이터를 즉각적으로 수집, 가공하여 분석 시스템으로 전달하는 실시간 분산 데이터 파이프라인의 구축은 안정적인 서비스 운영을 위한 필수 불가결한 요소가 되었다. 이러한 파이프라인은 크게 데이터 발생 계층(웹 서버), 수집 계층(로그 수집기), 가공 계층(데이터 병합기), 전송 계층(메시지 브로커), 처리 및 적재 계층(스트림 처리 엔진, 데이터 웨어하우스)으로 구분되며, 각 계층은 고유한 버퍼링 정책과 배치 전략을 가지고 있어 전체 파이프라인의 End-to-End 지연 시간에

www.kci.go.kr

서로 다른 방식으로 영향을 미친다. 따라서 실시간 데이터 파이프라인을 설계할 때에는 단순히 어떤 컴포넌트를 선택할지를 넘어, 파이프라인의 실시간성과 안정성을 결정짓는 다음과 같은 요소를 종합적으로 고려해야 한다.

첫째, 각 계층의 버퍼 크기와 배치 주기가 전체 지연 시간에 미치는 누적 효과를 고려하여, 계층별 대기 시간이 E2E 경로 전체에서 중첩되지 않도록 조율해야 한다.

둘째, 수집·전송·처리 계층 간의 처리 속도 불균형으로 인해 발생할 수 있는 백프레셔(Back-pressure)와 큐 적체를 방지할 수 있는 트래픽 셰이핑(Traffic Shaping) 메커니즘이 필요하다.

셋째, CPU 컨텍스트 스위칭, 디스크 I/O, 네트워크 패킷 수 등 시스템 자원의 고갈 위험을 회피할 수 있도록 파라미터의 극단값 설정을 지양해야 한다.

본 논문에서 사용하는 주요 용어는 다음과 같이 정의한다. 파이프라인 붕괴(Pipeline Collapse)는 특정 계층의 순간 도착률이 후위 계층의 처리율을 지속적으로 초과하여 큐 적체가 누적되고 종단 간 상위 백분위 지연 시간이 급격히 증가하는 상태를 의미한다. 트래픽 셰이핑(Traffic Shaping)은 scan_frequency, 배치 크기, flush 조건 등을 조정하여 후위 계층으로 전달되는 이벤트의 버스트 크기와 도착 간격을 제어하는 기법이다. 무대기(Zero-Wait) 전송은 물리적 지연 시간이 0이라는 의미가 아니라, 배치 누적 대기 또는 전송 대기를 최소화하여 이벤트를 가능한 한 즉시 후위 계층으로 전달하는 구성 방식을 의미한다.

다음 절에서는 본 연구에서 구축한 전체 파이프라인의 아키텍처와 각 컴포넌트의 배치를 구체적으로 제시한다.

1. Overview of the entire architecture

본 연구에서 분석하는 대규모 데이터 파이프라인은 오픈소스 생태계에서 가장 널리 사용되는 표준 컴포넌트들로 구성된다. 이는 해당 컴포넌트들이 이미 실무 현장에서 광범위하게 검증되어 있어 본 연구의 결과를 산업 환경에 직접 적용할 수 있고, 선행 연구[4]-[8]와 동일한 기준에서 성능 지표를 비교 분석할 수 있으며, 소스 코드와 설정이 모두 공개되어 있어 실험의 재현성과 투명성을 확보할 수 있기 때문이다. Fig. 1은 파이프라인의 전체 아키텍처를 나타낸다.

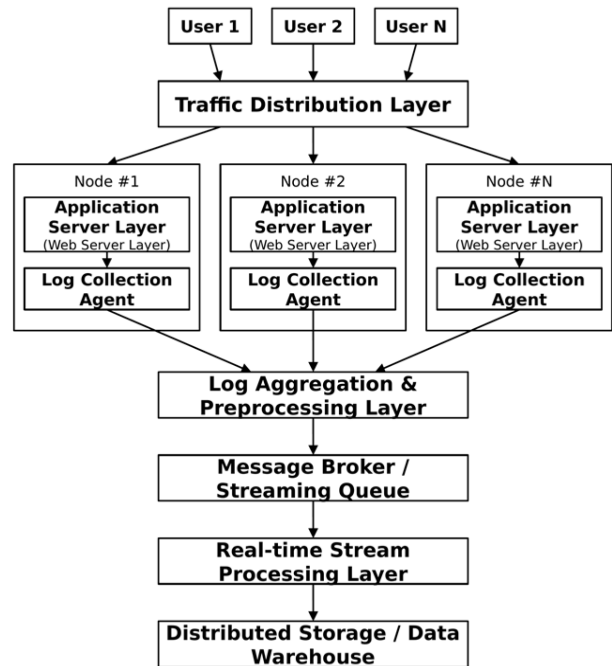

Fig. 1. Overall Architecture of the Data Pipeline

본 연구에서는 각 역할을 수행하는 구현체로 오픈소스 생태계의 대표적인 표준 컴포넌트를 채택하였다. 각 컴포넌트의 역할과 선택 근거는 다음과 같다.

먼저, Traffic Distribution Layer와 Application Server Layer는 각각 Nginx와 Flask로 구현하였다. Nginx는 클라이언트 요청을 수신하여 뒷단 애플리케이션 서버로 분산하는 리버스 프록시(Reverse Proxy) 겸 웹 서버이며 Flask는 Python 기반의 경량 웹 애플리케이션 프레임워크이다. 이 조합은 실제 웹 서비스 환경에서 가장 보편적으로 사용되는 리버스 프록시와 애플리케이션 서버 계층이다. Nginx의 접속 로그가 파일 기반으로 기록되어 후속 수집 계층과의 연동 방식이 표준적이며 밀리초 단위의 타임스탬프를 로그에 직접 기입할 수 있어 파이프라인 진입점의 시각을 정확하게 확정할 수 있다. 성능 관점에서 이 계층은 디스크 쓰기 성능과 동시 연결 처리 한계에 의해 로그 발생 속도가 결정되며, 본 연구에서는 파이프라인의 시작점이자 E2E 지연 시간 측정의 기준 시각을 제공하는 지점으로 정의한다.

각 애플리케이션 서버 하단의 Log Collection Agent는 Filebeat로 구현하였다. Filebeat는 각 서버에 설치되는 경량 에이전트로 디스크에 기록된 로그 파일을 실시간으로 감지하여 후위 계층으로 전달한다. 단일 바이너리 기반으로 배포가 간편하고 리소스 사용량이 낮으며, 파일 기반 로그 수집 방식과 Elastic Stack과의 높은 연동성으로 인해 널리 활용된다. 성능은 파일 스캔 주기와 배치 전송 크

www.kci.go.kr

Table 1. Overview of Key Configuration Parameters for Each Pipeline Component

Component	Parameter	Description	Default Value
Flask	logging format	Structured log format with timestamps for pipeline tracing.	Multi-line JSON
Filebeat	multiline	Combines multi-line JSON logs into a single event.	Enabled
	scan_frequency	Interval for scanning log files for new entries.	10 sec
	close_inactive	Time before closing an inactive file handle.	5 min
	queue.mem.flush.min_events	Minimum queued events before flushing downstream.	2048 events
	queue.mem.flush.timeout	Maximum wait time before forcing a flush.	1 sec
	output.logstash.bulk_max_size	Maximum events sent in a single batch to Logstash.	2048 events
	output.logstash.pipelining	Number of concurrent batches in flight to Logstash.	2 batches
Logstash	pipeline.batch.size	Number of events processed per worker batch.	125 events
	pipeline.batch.delay	Maximum wait time before processing a partial batch.	50 ms
Kafka	linger_ms	Producer wait time for batching messages before send.	0 ms
	acks	Acknowledgment level controlling reliability and throughput.	1 (leader acknowledgment)

기에 의해 수집 지연과 네트워크 I/O 부하가 결정되며, 본 연구에서는 이를 수집 계층의 I/O를 제어하는 핵심 지점으로 정의하고 주요 분석 대상으로 활용한다.

이후 Log Aggregation & Preprocessing Layer는 Logstash로 구성하였다. Logstash는 다수의 입력 소스로부터 수집된 로그를 병합하고, 필터링·파싱·포맷 변환을 수행하여 후위 브로커로 전달하는 데이터 가공 계층이다. 구조화되지 않은 로그를 표준 스키마로 변환할 수 있는 유연한 처리 기능을 제공하며, Elastic Stack 기반 파이프라인에서 널리 사용된다. 성능은 내부 배치 크기와 처리 대기 시간에 의해 처리량과 지연 시간이 결정되며, 유입 트래픽을 즉시 처리하지 못할 경우 내부 큐에 백로그(Backlog)가 발생한다. 본 연구에서는 이를 수집 계층의 부하가 전파되는 첫 번째 병목 지점으로 정의한다.

그 아래의 Message Broker / Streaming Queue는 Kafka로 구현하였다. Kafka는 대규모 이벤트 스트림을 파티션 단위로 분산 저장·전달하는 분산 메시징 플랫폼으로, 프로듀서와 컨슈머 간 결합도를 낮추고 비동기 버퍼 역할을 수행한다. 높은 처리량과 확장성을 제공하여 실시간 데이터 파이프라인의 표준 브로커로 널리 활용된다. 성능은 메시지 배치 전송 대기 시간과 수신 확인 수준에 의해 전송 지연과 데이터 신뢰성이 결정된다. 본 연구에서는 이를 파이프라인의 비동기 버퍼이자 전달 지연을 결정하는 핵심 전송 지점으로 정의한다.

마지막으로, 하단의 Real-time Stream Processing Layer와 Distributed Storage / Data Warehouse는 각각 Spark Streaming과 Hive로 구성하였다. Spark Streaming과 Hive는 각각 실시간 스트림 처리와 대규모 데이터 저장을 담당하는 계층이다. Spark Streaming은 마이크로 배치 단위로 데이터를 처리하여 실시간 분석을 수행하며, Hive는 처리된 데이터를 영속적으로 저장하는 데이터 웨어하우스 역할을 한다. 이 조합은 스트림 처리와 대용량 저장을 결합한 실무 표준 구조로 널리 활용된다. 성능은 배치 처리 주기에 의해 소비 지연이 결정되며, 본 연구에서는 이를 파이프라인의 최종 소비 계층이자 End-to-End 지연 시간 측정의 종단 지점으로 정의한다.

2. Primary Configuration Parameters Affecting Performance

각 컴포넌트는 처리량과 지연 시간의 상충관계를 조절할 수 있는 핵심 파라미터를 제공한다. Table 1은 본 연구의 최적화 실험에서 다루는 각 컴포넌트별 주요 설정값과 그 특징을 나타낸다. Table 1의 Default Value 컬럼은 각 컴포넌트의 공식 배포 시 부여되는 초기 기본값을 나타낸다. V장의 각 실험 시나리오에서는 이 기본값을 기준점으로 삼아, Table 1에 열거된 파라미터 중 특정 항목만을 선택적으로 변경한다. 변경되지 않은 파라미터는 기본값이 그대로 유지되며, 각 실험에서 변경된 구체적인 항목과 설정값은 해당 절에서 명시한다.

IV. Experiments

본 장에서는 실험을 위해 구축한 클라우드 네이티브 기반의 분산 데이터 파이프라인 아키텍처와 하드웨어 및 소프트웨어 명세를 설명한다.

1. Traffic Generation Setup

1.1 Traffic Generator Design

본 실험에서는 실제 사용자의 웹 접속 패턴을 모사하기 위해 자체 개발한 트래픽 제너레이터(Traffic Generator)를 사용하였다. 트래픽 제너레이터는 다수의 가상 사용자

www.kci.go.kr

가 유한 상태 머신(FSM) 기반의 확률적 전이 모델에 따라 회원가입, 로그인, 상품조회, 검색, 장바구니 추가, 결제, 리뷰 작성 등 15개 API 엔드포인트를 무작위로 조합하여 호출하는 시나리오 기반 부하 생성 도구이다. 각 가상 사용자는 성별 및 연령대에 따른 카테고리 선호 가중치와 세션 컨텍스트(장바구니 보유 여부, 페이지 깊이 등)에 기반한 행동 패턴을 보이며, 이를 통해 단순 반복 호출이 아닌 실제 사용자에 가까운 요청 시퀀스를 생성한다. 단일 머신에서 부하를 생성할 경우 발생하는 클라이언트 측 병목(포트 고갈, CPU 한계 등)을 원천적으로 차단하기 위해, 3대의 독립적인 가상 머신에서 트래픽 제너레이터를 병렬로 구동하였다.

각 HTTP 요청은 Nginx에 의해 구조화된 JSON 포맷의 접속 로그로 디스크에 기록된다. 이러한 로그는 본 연구에서 측정 대상으로 삼는 파이프라인 입력 데이터의 원본이 되며, 이를 통해 네트워크 부하 및 처리량을 정량적으로 추적할 수 있도록 구성하였다.

1.2 Web Server Configuration

트래픽 제너레이터로부터 유입되는 요청을 처리하기 위해, Flask 기반의 웹 애플리케이션과 Nginx 리버스 프록시를 결합한 웹 서버 환경을 구축하였다. Flask 애플리케이션은 실제 온라인 쇼핑몰 서비스를 모사한 구조로 설계되었으며, MySQL 데이터베이스와 연동하여 사용자, 상품, 장바구니, 주문, 리뷰 등의 데이터를 관리한다. 이를 통해 단순한 정적 응답이 아닌, 데이터베이스 조회 및 쓰기 연산을 수반하는 실제 서비스 수준의 로그 패턴을 생성할 수 있도록 하였다. 또한 해당 애플리케이션은 회원 관리, 상품 및 카테고리 조회, 검색, 장바구니 처리, 결제, 리뷰 작성 등 실제 사용자 행위를 반영하는 다수의 API 엔드포인트로 구성되어 있다. 예를 들어, 사용자가 특정 상품을 조회하면 상품 정보와 리뷰 데이터를 조회하는 요청이 발생하고, 장바구니에 상품을 추가하거나 결제를 수행하는 경우에는 관련 데이터의 갱신 및 주문 정보의 저장이 함께 이루어진다.

트래픽 제너레이터는 이 엔드포인트들에 대해 상품 조회, 검색, 로그인, 장바구니 추가, 결제 등의 사용자 행위 시나리오를 무작위로 조합하여 요청을 발생시킨다. 예를 들어, 하나의 세션은 로그인 후 특정 카테고리의 상품 목록을 조회하고, 검색을 수행한 뒤, 선택한 상품을 장바구니에 담아 최종적으로 결제하는 순서로 진행될 수 있다. 각 요청은 Nginx를 거쳐 Flask로 전달되며, Nginx는 요청마다 $msec 변수를 활용하여 밀리초 단위의 타임스탬프와 함께 접속 로그를 디스크에 기록한다. 여기서 $msec는 현재 시각을 Unix epoch 기준의 초 단위 실수형으로 반환하는 Nginx의 내장 변수로, 소수점 이하 밀리초 해상도를 제공한다. 이렇게 생성된 접속 로그는 Filebeat에 의해 수집되어 후위 파이프라인으로 전달되는 원본 데이터로 활용된다.

1.3 Log Generation Process

전체 로그 생성 흐름은 다음과 같이 이루어진다. 먼저 3대의 트래픽 제너레이터는 각각 독립적으로 다수의 가상 사용자 세션을 병렬로 구동하며 각 가상 사용자는 1.1절에서 기술한 FSM 기반 확률적 전이 모델에 따라 행동한다. 즉, 각 세션은 로그인 → 카테고리/상품 조회 → 검색 → 장바구니 담기 → 결제 → 리뷰 작성 → 로그아웃으로 이어지는 일련의 상태를 거치며 현재 상태에서 다음 상태로 전이될 때마다 해당 상태에 대응하는 Flask 애플리케이션의 API 엔드포인트를 호출한다. 이때 가상 사용자의 성별·연령대에 따른 카테고리 선호 가중치와 세션 컨텍스트(로그인 여부, 장바구니 보유 여부, 현재 페이지 깊이 등)가 전이 확률에 반영되어 단순 반복 호출이 아닌 실제 사용자에 가까운 비정형적 요청 시퀀스가 지속적으로 생성된다.

이렇게 생성된 요청은 애플리케이션 로드 밸런서를 거쳐 하단의 Nginx 웹 서버로 고르게 분산되며 Nginx는 요청을 Flask 애플리케이션으로 전달한다. Flask는 엔드포인트별 비즈니스 로직(MySQL 데이터베이스 조회·쓰기 포함)을 수행한 뒤 응답을 반환하고 이 과정에서 Nginx가 생성한 JSON 포맷의 접속 로그가 디스크에 기록된다. 이후 Filebeat가 해당 로그 파일을 실시간으로 감시하여 새로운 라인을 감지하는 즉시 후위 파이프라인으로 전송을 시작한다. 이러한 구조를 통해 실제 운영 환경에서의 로그 발생 패턴을 충실히 재현하면서도 트래픽의 양과 패턴을 정밀하게 제어할 수 있는 실험 환경을 확보하였다.

2. Experimental Environment and System Specifications

실험의 재현성을 확보하기 위해 Table 2와 같이 독립적인 클라우드 가상 머신 인스턴스들을 구성하였다. 실험의 통계적 신뢰성을 확보하기 위해 각 시나리오는 동일한 부하 조건에서 5회 반복 수행하였으며, 실험 간 시스템 상태와 잔여 버퍼의 영향을 최소화하기 위해 5분의 쿨다운(Cooldown) 시간을 부여하였다. 동일한 반복 실험 내 이벤트 지연 시간은 시간적 자기상관을 가질 수 있으므로, 본 연구에서는 개별 이벤트가 아닌 반복 실험 단위의 요약

www.kci.go.kr

통계량을 비교 기준으로 사용하였다. 각 반복 실험에서 평균 종단 간 지연 시간과 P95 지연 시간을 산출하고, 5회 반복 결과의 평균과 표준편차를 제시하였다. 시나리오 간 통계적 차이는 반복 실험 단위의 P95 지연 시간을 대상으로 Welch's t-test를 적용하여 검토하였으며, 반복 횟수의 한계를 고려하여 p-value, 평균 차이, 표준편차, 정상 상태 구간의 결과를 종합적으로 해석하였다.

Table 2. Experimental Environment and Software Specifications

Category	Component	System Specification / Version
Hardware	Traffic Generator (×3)	4vCPU, 16GiB, 30GB SSD
	Pipeline Node	2vCPU, 8GiB, 10GB SSD
Software	Operating System	Ubuntu 22.04 LTS
	Load Balancer	KakaoCloud Managed Application Load Balancer
	Web Server	Nginx 1.18.0, Flask 2.0.1
	Log Shipper	Filebeat 7.17.29
	Data Aggregator	Logstash 7.17.29
	Message Broker	Apache Kafka 3.7.1
	Stream Processor	Apache Spark 3.5.2
	Data Warehouse	Apache Hive 3.1.3

3. Overall Experimental Architecture

Fig. 2는 본 연구에서 구축한 전체 실험 아키텍처를 나타낸다. 본 실험에서는 파이프라인을 구성하는 모든 컴포넌트를 편의상 단일 파이프라인 노드에 모아서 구성하였다. 이는 네트워크 홉(Hop), 노드 간 대역폭 차이, 분산 스케줄링 지연 등 외부 변수를 통제하고, 각 컴포넌트의 버퍼링 및 배치 설정이 종단 간 지연 시간에 미치는 순수한 영향을 분리하여 관찰하기 위함이다. 트래픽 제너레이터 3대만 별도의 독립적인 가상머신(Virtual Machine, VM)에 배치하여 클라이언트 측 병목을 방지하였다. 따라서 본 연구의 결과는 다양한 클라우드 및 멀티노드 환경 전반의 일반적 성능 보장이라기보다, 단일 파이프라인 노드 환경에서의 E2E 병목 분해와 튜닝 효과에 대한 실증 결과로 해석되어야 한다.

본 연구는 통제된 실험 환경에서 계층별 설정값 변화의 영향을 분석하였으나, 멀티 노드 환경으로 확장할 경우 추가적인 병목 요인이 발생할 수 있다. 예를 들어, 컴포넌트가 서로 다른 노드에 배치되면 네트워크 홉 증가로 인한 전송 지연과 대역폭 경합이 발생할 수 있으며, Kafka 브로커를 분리할 경우 파티션 배치, 복제, 리더 브로커 위치에 따라 처리량과 지연 시간이 달라질 수 있다. 또한 Spark를 클러스터 모드로 운영할 경우 드라이버-워커 간 통신, 태스크 스케줄링, 셔플 비용 등이 전체 지연 시간에 영향을 줄 수 있다. 따라서 멀티 노드 환경에서는 제안한 설정 전략을 그대로 적용하기보다, 각 계층의 배치 구조와 통신 비용을 함께 고려한 추가적인 성능 검증이 필요하다.

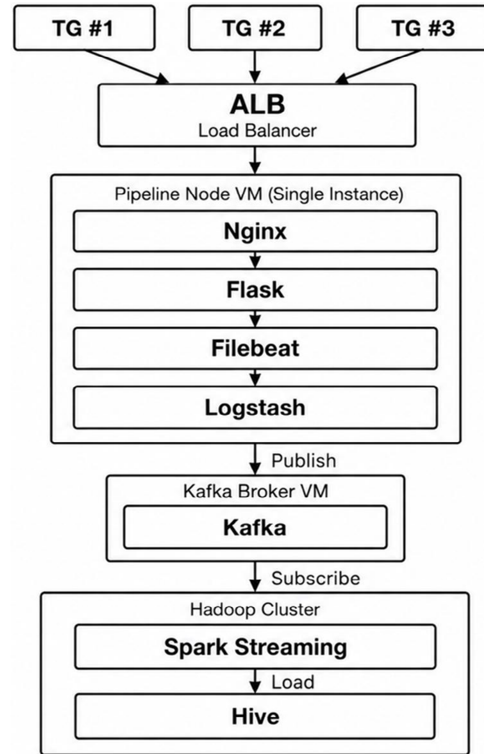

Fig. 2. Experimental Architecture

4. End-to-End Latency Tracking Mechanism

정확한 지연 시간 측정을 위해 시스템 각 계층에 타임스탬프 기록 로직을 주입하였다. 각 로그 이벤트에는 고유 식별자인 request_id를 부여하였으며, 최초 로그 발생 시점은 Nginx의 $msec 변수를 이용해 ts_nginx 필드에 기록하였다. 이후 Filebeat 수집, Logstash 가공, Kafka 전송, Spark Streaming 소비, Hive 적재 단계에서 이벤트가 각 계층을 통과하는 시각을 각각 ts_filebeat, ts_logstash, ts_kafka, ts_spark, ts_hive 필드로 추가하였다. 각 컴포넌트는 원본 JSON 이벤트를 유지한 채 해당 계층의 타임스탬프 필드만 덧붙여 후위 계층으로 전달하였고, 최종 E2E 지연 시간은 Hive 적재 레코드에서 ts_hive - ts_nginx로 산출하였다. 분산 환경에서의 정확한 타임스탬프 비교를 위해 모든 VM 인스턴스는 NTP(Network Time Protocol)를 통해 시스템 클록을 동기화하였다. 사전 측정 결과 노드 간 클록 오차는 최대 1 ms 이내로 유지됨을 확인하였으며, 이는 본 연구의 측정 단위인 밀리초 수준의 정밀도 추적에 충분히 허용 가능한 범위이다. 또한 각 실험 종료 후 request_id를 기준으로

www.kci.go.kr

Nginx 원본 로그 건수와 Hive 최종 적재 건수를 비교하여 수집-적재 구간의 이벤트 유실 여부를 확인하였으며, 본 실험에서는 분석 대상 구간에서 유의미한 이벤트 드롭이 발생하지 않았음을 확인하였다. Table 3은 본 연구에서 End-to-End 지연 시간을 분해하여 측정하는 파이프라인 구간을 정의한다. 각 구간의 명칭은 이후 실험 결과의 그래프 범례 및 본문에서 동일하게 사용된다. 전체 End-to-End 지연 시간은 아래 6개 구간의 합산으로 산출된다.

Table 3. Definition of Pipeline Latency Measurement Stages

#	Stage Name	Description
1	Nginx -> Filebeat	Time from Nginx writing the access log to disk until Filebeat detects and ingests the new log entry. Includes file scan interval.
2	Filebeat Processing	Time spent inside Filebeat for buffering and assembling a batch before transmission. Governed by flush.min_events and bulk_max_size.
3	Filebeat -> Logstash	Network transfer time for delivering the assembled batch from Filebeat to Logstash.
4	Logstash Processing	Time spent inside Logstash for filtering, parsing, and format transformation. Includes pipeline.batch.delay wait time.
5	Logstash -> Kafka	Time for the Logstash output plugin to produce the processed message to the Kafka broker. Includes linger_ms wait time.
6	Kafka -> Hive	Time from message arrival in Kafka until Spark Streaming consumes and loads the record into the Hive data warehouse.

V. Experimental Scenarios and Results Analysis

본 장에서는 파이프라인 컴포넌트의 설정 변인(버퍼 크기, 배치 대기 시간, 파일 스캔 주기 등)이 전체 시스템 성능에 미치는 영향을 분석하기 위해 설계한 5단계의 실험 시나리오와 그 결과를 제시한다. 5단계의 시나리오는 실시간 데이터 파이프라인에서 적용 가능한 대표적인 정적 최적화 전략군을 동일한 환경에서 비교하기 위한 기준선 비교로 해석할 수 있다. Baseline은 기본 버퍼링 정책, Zero-Wait Transmission은 저대기 전송 전략, Micro-Batching Optimization은 정적 배치 제어 전략, Collection-Layer I/O Control은 수집 계층 트래픽 셰이핑 전략, Hybrid Optimization은 계층 간 결합 전략에 해당한다. 따라서 본 연구는 파이프라인 구조 변경 없이 적용 가능한 정적 파라미터 튜닝 전략 간의 성능 상충관계 분석에 초점을 둔다. Table 4는 각 시나리오의 정식 명칭과 Baseline 대비 핵심 변경 사항을 정의한다. 각 실험에서는 검증하고자 하는 가설, 해당 실험에서만 변경된 설정, 실험 결과, 그리고 이에 대한 해석을 순서대로 기술한다. 본 연구에서 queue.mem.flush.timeout = 0s는 flush 대기 시간 제거, output.logstash.pipelining = 0은 Logstash 출력 파이프라이닝 비활성화를 의미한다.

Table 4. Experimental Scenario Definitions

Scenario Name	Component	Parameter	Value
Baseline	-	All parameters	Default values
Zero-Wait Transmission	Filebeat	multiline	Disabled
		queue.mem.flush.min_events	1
		queue.mem.flush.timeout	0s
		output.logstash.bulk_max_size	1
		output.logstash.pipelining	0
	Logstash	pipeline.batch.size	1
		pipeline.batch.delay	1ms
Micro-Batching Optimization	Filebeat	output.logstash.bulk_max_size	50
	Logstash	pipeline.batch.size	50
		pipeline.batch.delay	5ms
	Kafka	linger_ms	5ms
Collection-Layer I/O Control	Filebeat	scan_frequency	10ms
		close_inactive	1m
Hybrid Optimization	Filebeat	scan_frequency	10ms
		close_inactive	1m
		output.logstash.bulk_max_size	50
	Logstash	pipeline.batch.size	1
		pipeline.batch.delay	1ms

본 연구의 파이프라인은 Nginx, Filebeat, Logstash, Kafka, Spark Streaming, Hive가 직렬로 연결된 tandem queue 구조로 해석할 수 있다. 각 계층 i의 입력률을 λ_i, 서비스율을 μ_i, 배치 대기 시간을 B_i, 실제 처리 시간을 S_i, 큐잉 대기 시간을 W_i라고 하면, 전체 End-to-End 지연 시간은 다음과 같이 근사할 수 있다.

$$T_{E2E} \approx \sum_i (B_i + W_i + S_i), \quad \rho_i = \frac{\lambda_i}{\mu_i}$$

위 식에서 기본 설정은 큰 배치 크기와 flush timeout으로 인해 B_i가 증가하는 구조이며, Zero-Wait 설정은 B_i를 줄이는 대신 이벤트 단위 전송 증가로 패킷 수와 컨텍스트 스위칭을 증가시켜 일부 계층의 유효 서비스율 μ_i를 낮출 수 있다. 또한 계층 이용률 ρ_i가 1에 가까워질수

www.kci.go.kr

록 큐잉 대기 시간 W_i가 급격히 증가하므로, Micro-Batching처럼 순간 유입률의 변동성이 큰 경우 후위 계층에서 큐 적체와 꼬리 지연 증가가 발생할 수 있다. 이러한 관점에서 Hybrid Optimization은 전위 수집 계층에서 순간 유입률의 변동을 완화하여 후위 계층의 ρ_i를 낮추고, 후위 계층에서는 불필요한 B_i를 줄임으로써 전체 T_{E2E}와 꼬리 지연을 완화하는 계층별 분리 전략으로 해석할 수 있다.

1. Baseline Experiment

본 실험은 이후 수행될 최적화 시나리오들과의 비교 분석을 위한 기준 실험으로서 파이프라인을 구성하는 모든 컴포넌트를 설치 시 부여되는 초기 기본값으로 유지한 상태에서 수행한다. 이를 통해 컴포넌트에 적용되는 범용적인 설정이 대규모 실시간 트래픽 환경에서 어떠한 구조적 한계를 드러내는지 확인하고 이후 시나리오에서 변경되는 파라미터가 전체 성능에 미치는 영향을 정량적으로 평가할 수 있는 비교 기준점을 확보한다.

모든 파라미터는 Table 1에 제시된 기본 설정을 그대로 유지하였다. 이 설정에서는 웹 서버 로그가 구조화된 JSON 형식으로 기록되며 수집 계층은 여러 줄로 구성된 로그 이벤트를 하나의 레코드로 결합하여 처리한다. 또한 수집 계층은 일정량 이상의 로그가 내부 큐에 축적되거나 최대 대기 시간이 도달할 때까지 전송을 지연시키며, 한 번에 비교적 큰 단위의 배치를 구성하여 후위 계층으로 전달하도록 설정되어 있다. 가공 계층 역시 부분 배치를 즉시 처리하기보다는 일정 개수의 이벤트가 모이거나 짧은 대기 시간이 경과할 때까지 처리 시점을 유보한다. 반면 메시지 브로커 계층은 추가적인 전송 지연을 거의 두지 않는 기본 전송 정책을 유지한다. 즉, 전체적으로는 수집 계층에서 충분한 양의 데이터를 모아 일괄 전달하는 버퍼링 중심의 기본 정책이 적용된 상태이다. 해당 환경에서 180초 동안 분산 트래픽을 인가하였다.

Fig. 3은 Baseline 환경에서의 구간별 End-to-End 지연 시간 분해 추이를 나타내며 범례의 각 구간은 Table 3에 정의된 파이프라인 단계에 대응한다. Fig. 3의 x축은 실험 시작 이후 경과 시간을, y축은 각 파이프라인 구간에서 측정된 지연 시간을 나타낸다. 그래프의 각 선은 Table 3에서 정의한 각 구간의 지연 시간 변화를 의미한다. 특히 파란색 구간의 비중이 큰 것은 Filebeat의 기본 대용량 버퍼링 설정으로 인해 이벤트가 전송 전 내부 큐에서 대기하기 때문이다. Fig. 3에서 트래픽 유입 직후 약 30~40초 구간에서 전체 지연 시간이 급격히 상승하는 원인은 두 가지 요인의 복합 작용으로 설명된다.

첫째, Filebeat의 버퍼 채움 지연이 주된 원인이다. flush.min_events가 2048로 설정되어 있어 최초 유입된 로그 이벤트들은 이 임계값에 도달하기까지 내부 버퍼에 체류하게 된다. 이 기간 동안 선착 이벤트는 후착 이벤트가 버퍼를 채울 때까지 대기해야 하므로 시간이 경과할수록 이벤트당 버퍼 체류 시간이 선형적으로 누적되어 전체 지연이 증가한다.

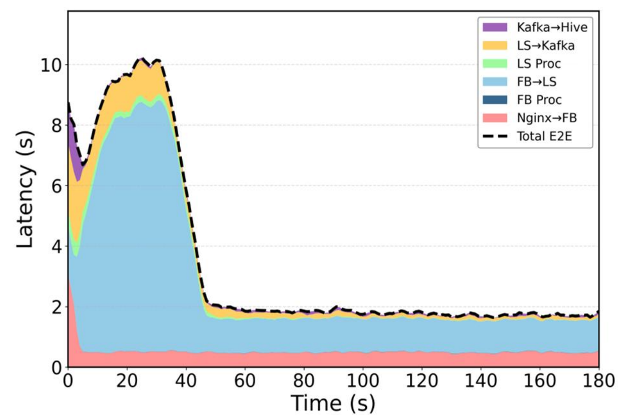

Fig. 3. Stage-wise End-to-End Latency Breakdown under the Baseline Configuration

둘째, JVM 기반 후위 컴포넌트의 웜업 효과가 보조적으로 작용한다. Logstash, Kafka, Spark Streaming은 모두 JVM 위에서 동작하며, 최초 호출 경로는 인터프리터로 실행되다가 점진적으로 JIT 컴파일에 의해 최적화된다. 또한 OS 페이지 캐시 프라이밍과 컴포넌트 간 연결 풀 초기화 비용 또한 시스템이 cold 상태에서 출발하는 매 실험의 초기 구간에서 일회적으로 발생한다. 약 40초 이후 지연 시간이 급격히 감소하여 안정화되는 이유는, 지속적인 트래픽 유입으로 Filebeat 내부 버퍼가 빠르게 채워지면서 flush.min_events(2048) 조건이 짧은 주기로 연속 충족되고, 동시에 후위 컴포넌트의 JIT 최적화와 캐시 프라이밍이 완료되기 때문이다. 파이프라인이 정상 상태에 진입하면, 각 이벤트의 평균 버퍼 체류 시간이 일정 수준으로 수렴한다. 안정화 구간 (약 60~180초)에서의 전체 End-to-End 지연 시간의 구간별 비중은 다음과 같다. Filebeat Processing 구간이 전체 지연의 약 60%로 가장 큰 비중을 차지하였으며 Filebeat → Logstash 구간이 약 18%, Logstash → Kafka 구간이 약 12%, Kafka → Hive 구간이 약 7%를 차지하였다. Nginx → Filebeat 및 Logstash Processing 구간은 합산 약 3%로 상대적으로

www.kci.go.kr

미미하였다.

Table 5와 Table 6은 Baseline 환경에서의 시스템 전체 및 서비스별 리소스 사용률을 나타낸다.

Table 5. System-wide Resource Utilization under the Baseline Configuration

Resource Metric	Filebeat	Logstash	Nginx/ Flask	Etc	Total
CPU utilization (%)	2.3 (3.6%)	19.3 (30.2%)	5.2 (8.1%)	37.2 (58.1%)	64.0
Disk write rate (kB/s)	3.2 (1.1%)	2.3 (0.8%)	152.0 (50.2%)	147.7 (48.8%)	302.8
Network traffic (pck/s)	-	-	-	-	8,303

Table 5와 Table 7의 Nginx/Flask 항목은 웹 계층 전체의 집계 사용량을 의미하고 Table 6과 Table 8의 Nginx 항목은 개별 Nginx 프로세스 기준의 사용량을 의미한다.

Table 6. Per-service Resource Utilization under the Baseline Configuration

Resource Metric	Filebeat	Logstash	Nginx
CPU utilization (%)	2.37	19.34	2.68
Disk write rate (kB/s)	0.99	2.17	78.44
Disk read rate (kB/s)	5.88	0.16	6.78
Cache hit rate (faults/s)	75.68	58.19	20.81

시스템 전체의 평균 CPU 사용량은 64% 수준이었으나, Table 6에서 Filebeat의 CPU 점유율은 2.37%에 불과하였다. 반면 Fig. 3에서는 전체 지연 시간의 약 60%가 Filebeat Processing 구간에 집중되었다. 이는 Baseline의 주요 병목이 CPU 처리 한계가 아니라, flush.min_events 2048 설정으로 인해 충분한 이벤트가 모일 때까지 버퍼에서 대기하는 논리적 지연에서 비롯되었음을 의미한다. 따라서 Baseline은 시스템 안정성은 유지하지만, 실시간 처리 측면에서는 구조적 한계를 갖는 것으로 확인되었다.

2. Low-Latency Transmission Experiment

본 실험은 1절의 Baseline 실험에서 확인된 누적 버퍼링 병목을 제거하기 위한 극단적 저지연 구성의 하한선 탐색 실험으로서 파이프라인을 구성하는 주요 컴포넌트의 대기 시간과 배치 크기를 최소화한 상태에서 수행한다. 이를 통해 로그 조립 지연과 배치 대기 시간을 동시에 최소화했을 때 관찰되는 종단 간 지연 시간의 실험적 하한선을 측정하고 이와 동시에 잦은 I/O 요청이 시스템 자원(CPU 컨텍스트 스위칭, 네트워크 패킷 처리량 등)에 미치는 부하를 관찰함으로써 단순한 대기 제거만으로는 해결되지 않는 구조적 한계를 규명한다.

Baseline의 Multi-line JSON 포맷은 Filebeat의 multiline 처리 과정에서 로그 조립 대기를 유발할 수 있다. 본 실험에서는 이를 줄이기 위해 Flask 로그를 Single-line JSON 포맷으로 변경하고, Filebeat와 Logstash의 배치 크기 및 대기 시간을 최소화하였다. 나머지 파라미터는 Table 1의 Baseline 설정을 유지하였다. 이에 따라 Zero-Wait Transmission 결과에는 로그 조립 지연 제거와 배치 대기 축소 효과가 함께 반영된다.

Table 7은 Zero-Wait Transmission 환경에서의 시스템 전체 및 서비스별 리소스 사용률을 나타낸다. 본 실험은 웹 서버 설정과 입력 요청률을 동일하게 통제한 상태에서 Filebeat, Logstash, Kafka 계층의 설정 변화가 리소스 사용률에 미치는 영향을 관찰하는 데 초점을 두었다. 이에 따라 Nginx 관련 지표는 Baseline과 유사한 수준으로 유지되었다.

Table 7. System-wide Resource Utilization under the Zero-Wait Transmission Configuration

Resource Metric	Filebeat	Logstash	Nginx/ Flask	Etc	Total
CPU utilization (%)	1.5 (2.2%)	25.4 (37.2%)	5.2 (7.6%)	36.1 (52.9%)	68.2
Disk write rate (kB/s)	3.1 (1.2%)	1.0 (0.4%)	114.2 (44.9%)	136.0 (53.5%)	254.3
Network traffic (pck/s)	-	-	-	-	9,267

단일 이벤트 단위의 잦은 컨텍스트 스위칭으로 인해 시스템 전체의 네트워크 패킷 발생량이 초당 9,267 pck/s로 급증하였다.

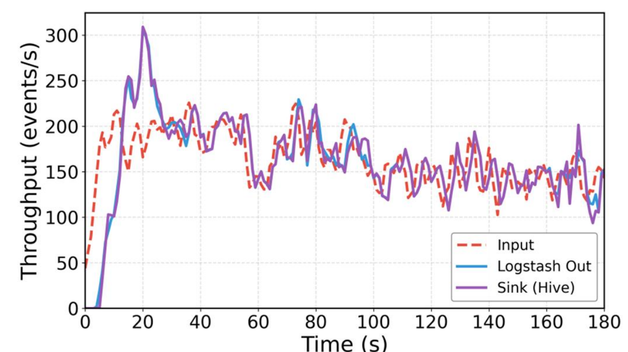

Fig. 4. Throughput Throttling under the Zero-Wait Transmission Configuration

www.kci.go.kr

Fig. 4는 Zero-Wait Transmission 환경에서의 파이프라인 단계별 처리량 추이를 나타낸다. 그래프의 Input은 Filebeat가 로그 파일에서 읽어 들여 후위 계층으로 최초 전달한 이벤트 수(events/s), Logstash Out은 Logstash가 필터링·가공을 완료하여 Kafka로 출력한 이벤트 수, Sink (Hive)는 Spark Streaming을 거쳐 최종적으로 Hive에 적재 완료된 이벤트 수를 각각 나타낸다. 전체 시스템의 처리량이 초당 약 300건(최대 309건) 수준의 박스권에 갇혀 강제 제한되었다.

저지연 전송 설정은 버퍼링으로 인한 논리적 대기 시간을 제거하는 데에는 성공하였으나, 이벤트 한 건마다 독립적인 네트워크 패킷을 발생시킴으로써 OS 레벨의 네트워크 병목을 유발하였다. 그 결과 트래픽 제너레이터가 초당 수백 건 이상의 부하를 인가하였음에도 불구하고, 파이프라인의 처리량이 초당 약 300건 수준에서 강제 제한되는 연쇄적 성능 저하가 관찰되었다. 이는 단순히 버퍼링을 제거하는 것만으로는 성능 최적화를 달성할 수 없으며, 오히려 잦은 I/O 요청이 시스템 전체의 처리 능력을 심각하게 저해할 수 있음을 실증한다. 따라서 지연 시간과 시스템 안정성 간의 상충관계를 고려한 균형 잡힌 접근이 필요함을 확인하였다.

3. Micro-Batching and Traffic Control Experiments

본 절에서는 Zero-Wait Transmission의 부작용을 극복하기 위해 배치 크기를 조절하는 Micro-Batching Optimization 실험과 수집 계층의 스캔 주기를 제어하는 Collection-Layer I/O Control 실험을 순차적으로 기술한다.

3.1 Micro-Batching Optimization

본 실험은 Zero-Wait Transmission에서 관찰된 단건 전송의 네트워크 과부하를 완화하기 위해, 단건 전송과 기본 대용량 버퍼링 사이의 절충안으로 소규모 배치 전송을 적용한 실험이다. 구체적으로 Filebeat의 output.logstash.bulk_max_size를 50으로 설정하고, Logstash의 pipeline.batch.delay를 5ms로 조정하여 네트워크 전송 오버헤드를 줄이면서도 과도한 배치 대기를 방지하고자 하였다. 이를 통해 소규모 배치 설정이 처리량과 지연 시간, 그리고 후위 컴포넌트로 전달되는 트래픽 패턴에 미치는 영향을 분석하였다.

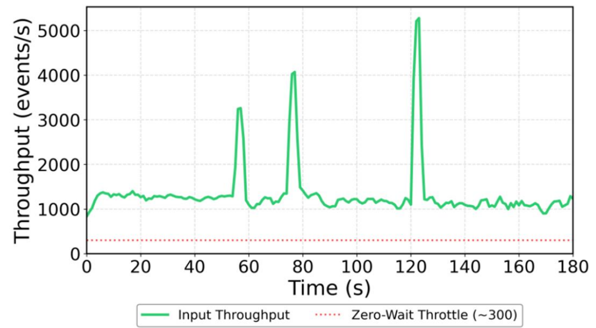

Fig. 5. Explosive Input Throughput Spikes under the Micro-Batching Configuration

Fig. 5는 Micro-Batching Optimization 환경에서의 입력 처리량 추이를 나타낸다. 입력 처리량은 Filebeat가 로그 파일에서 읽어 들여 후위 계층으로 최초 전달한 이벤트의 단위 시간당 건수(events/s)를 의미한다. 그래프에서 점선으로 표시된 기준선은 Zero-Wait Transmission 실험에서 관찰된 약 300 events/s 수준의 처리량 상한을 의미한다. Zero-Wait Transmission 실험에서 억눌렸던 전송량이 일시적으로 폭발하여 초당 4,000~5,000건 이상의 극단적인 스파이크(Spike)가 다수 발생하였다.

Fig. 6은 Micro-Batching Optimization 환경에서 관찰된 파이프라인 붕괴 현상의 직접 근거를 제시한다. Fig. 6(a)는 Baseline, Zero-Wait Transmission, Micro-Batching Optimization 환경의 전체 End-to-End 지연 시간 비교 결과로, x축은 실험 시작 이후 경과 시간을, y축은 각 시나리오의 종단 간 지연 시간을 나타낸다. Baseline 및 Zero-Wait Transmission은 초기 웜업 이후 약 2초 수준에서 안정화된 반면, Micro-Batching Optimization 환경에서는 60~115초 범위의 높은 지연 시간이 지속되었으며, 약 125초 시점에는 누적 큐잉으로 인한 추가적인 스파이크가 관찰되었다.

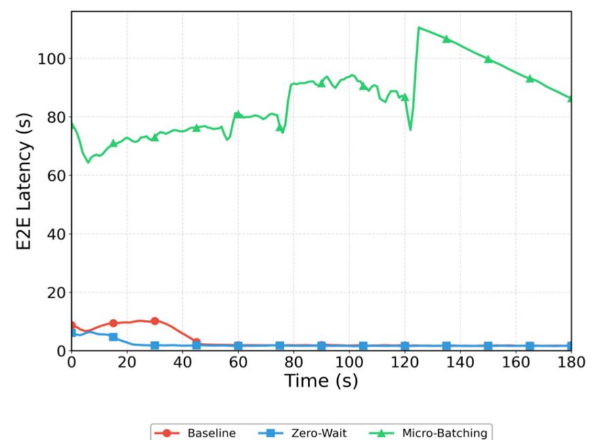

Fig. 6(a). End-to-End Latency Comparison across Optimization Stages

www.kci.go.kr

Fig. 6(b)는 Micro-Batching 환경에서 지연 누적이 가장 크게 나타난 두 구간을 보여준다. Nginx → Filebeat 구간은 평균 56.17초, 최대 80.11초, Filebeat → Logstash 구간은 평균 29.34초, 최대 38.83초의 지연을 보였다. 반면 나머지 구간의 평균 지연은 모두 1초 미만으로 유지되었다. 이는 Fig. 6(a)의 60초 이상 지속된 End-to-End 지연 증가가 전체 계층의 처리 지연이 아니라, 수집 및 전송 초기 구간의 큐잉 지연 누적과 관련됨을 보여준다.

마이크로 배치 설정은 네트워크 효율을 개선하였으나, 배치 단위 전송으로 인해 후위 계층에 입력 스파이크와 큐잉 지연이 발생하였다. 이는 배치 크기 조정만으로는 파이프라인 안정성을 보장하기 어렵고, 유입 속도 제어가 함께 필요함을 보여준다.

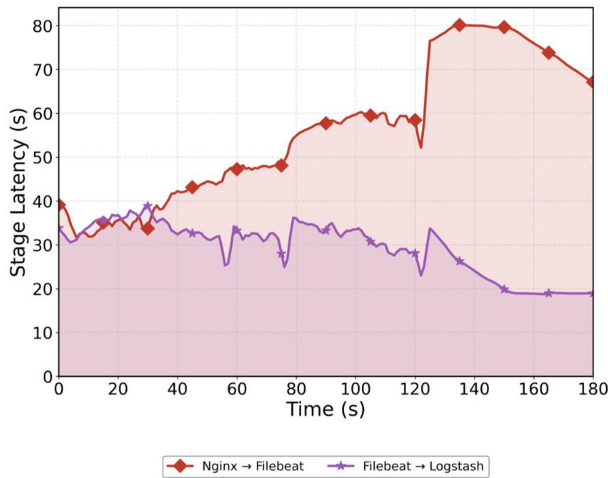

Fig. 6(b). Upstream Queueing Delay under Micro-Batching

3.2 Collection-Layer I/O Control

본 실험은 3.1절의 Micro-Batching Optimization 실험에서 관찰된 입력 트래픽 스파이크와 이로 인한 파이프라인 붕괴 현상을 해소하기 위한 트래픽 셰이핑 검증 실험으로서, 앞선 마이크로 배치 설정을 그대로 유지한 상태에서 Filebeat의 파일 스캔 주기만을 극단적으로 단축하여 수행한다. 이를 통해 수집 계층이 데이터를 거대한 덩어리가 아닌 작고 균일한 흐름으로 분할하여 지속적으로 밀어냄으로써, 후위 컴포넌트로 전달되는 트래픽 패턴이 평탄화되는지를 검증하고, 수집 계층의 I/O 제어가 파이프라인 전체의 안정성에 미치는 연쇄적 효과를 실증한다.

Filebeat의 기본 파일 스캔 주기는 통상 10초 단위로 설정되어 있어 초기 지연의 주범이 된다. 이를 10ms로 강제 단축함으로써 수집 계층의 파일 읽기 대기 시간을 제거하고, 뒷단으로 향하는 트래픽 유입 속도를 균일하게 조절하는 것을 목표로 한다. 또한 close_inactive를 기본값인 5분에서 1분으로 단축하여, 비활성 상태가 된 파일 핸들을 빠르게 해제함으로써 잦은 스캔으로 인해 누적될 수 있는 핸들 점유 부담을 완화하였다.

Table 8은 Collection-Layer I/O Control 환경에서의 Filebeat 리소스 사용률을 Baseline과 비교하여 나타낸다.

Table 8. Per-service Resource Utilization under the Collection-Layer I/O Control Configuration

Resource Metric	Filebeat	Logstash	Nginx
CPU utilization (%)	3.31	18.31	2.68
Disk write rate (kB/s)	1.84	3.04	62.08
Disk read rate (kB/s)	6.54	3.09	7.42
Cache hit rate (faults/s)	217.61	31.98	21.07

Baseline 대비 Filebeat의 Cache Hit이 75.68에서 217.61 faults/s로 약 2.9배 폭증하였고, CPU 사용률도 2.37%에서 3.31%로 상승하였다. 반면 Filebeat의 Disk Write는 0.99에서 1.84 kB/s로 증가한 수준에 머물렀다. 이는 scan_frequency 단축이 디스크 쓰기가 아닌 디스크 스캔·읽기(Cache Hit와 Disk Read) 부하를 집중적으로 증가시켰음을 의미한다.

Fig. 7은 Collection-Layer I/O Control를 적용한 뒤의 전체 지연 시간 추이를 나타낸다. Micro-Batching Optimization 실험에서 관찰된 80~100초 수준의 거대한 지연 시간이 1초 미만으로 극적으로 안정화되었다.

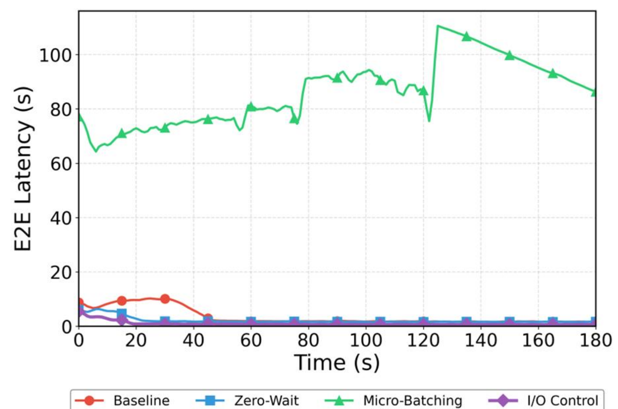

Fig. 7. Dramatic Latency Stabilization after Applying Collection-Layer I/O Control

스캔 주기를 10ms로 단축한 결과, Filebeat가 배치 크기를 100% 채우기 전에도 데이터를 지속적으로 밀어내게 되었다. 이로 인해 후위 컴포넌트로의 데이터 유입이 거대한 덩어리가 아닌 작고 균일한 흐름으로 전환되어,

www.kci.go.kr

Micro-Batching Optimization 실험에서 발생한 스파이크와 파이프라인 붕괴가 완전히 해소되었다. 이 결과는 수집 계층의 파일 스캔 주기가 단순히 로그 감지 속도뿐 아니라 전체 파이프라인의 트래픽 세이핑 역할을 수행하며 후위 컴포넌트의 안정성에 결정적인 영향을 미친다는 사실을 실증한다.

4. Hybrid Optimization Experiment

본 실험은 앞서 수행한 1절부터 3.2절까지의 네 가지 실험에서 도출된 교훈을 종합하여 설계한 최종 검증 실험으로서 전위 계층(수집 계층)에는 3.2절에서 효과가 입증된 정교한 I/O 제어 기반 트래픽 세이핑 전략을, 후위 계층(가공·전송 계층)에는 2절에서 확인된 Zero-Wait Transmission을 각각 적용하는 계층별 Hybrid Optimization 전략이 파이프라인의 지연 시간, 처리량, 시스템 안정성이라는 세 가지 핵심 성능 지표를 동시에 만족할 수 있는지를 최종 검증한다. 이를 통해 본 연구가 제안하는 계층별 하이브리드 전략이 단일 전략의 부작용을 상호 보완하면서 실무에 즉시 적용 가능한 실용적 최적화 모델임을 실증하고자 한다.

전위 계층인 Filebeat는 Collection-Layer I/O Control 실험의 설정을 유지하여 scan_frequency: 10ms, bulk_max_size: 50으로 디스크 I/O를 정교하게 제어하고 대규모 덩어리 전송을 방지한다. 반면, 메모리 기반으로 동작하여 비교적 I/O 여유가 있는 후위 계층인 Logstash(batch.size: 1, batch.delay: 1ms)와 Kafka(linger_ms: 0)는 Zero-Wait Transmission 실험의 설정으로 복귀시켰다. 이는 앞단에서 유입 속도를 선제적으로 통제한 상태에서 뒷단의 전송 대기만 제거하면, Zero-Wait Transmission 실험에서 발생한 네트워크 과부하를 회피하면서도 Zero-Wait Transmission의 이점을 확보할 수 있다는 가설에 기반한 설계이다.

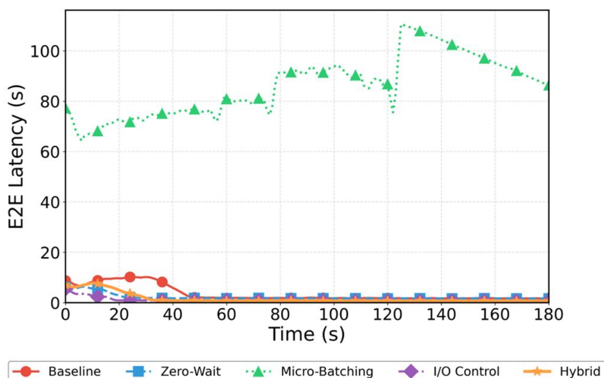

Fig. 8. Final End-to-End Latency Comparison across All Optimization Scenarios

Fig. 8은 전체 5가지 실험 시나리오의 End-to-End 지연 시간 최종 비교 결과를 나타낸다. Hybrid Optimization 설정이 초기 웜업 구간 이후 안정화된 상태에서 평가된 시나리오 중 낮은 P95 지연 시간과 안정적인 지연 수준으로 나타났다.

Fig. 9는 최적화 시나리오 간 파이프라인 단계별 처리량 종합 비교 결과를 나타낸다. Hybrid Optimization이 스파이크나 스로틀링 없이 파이프라인 전 구간에서 비교적 안정적인 처리량을 유지하는 것으로 나타났다.

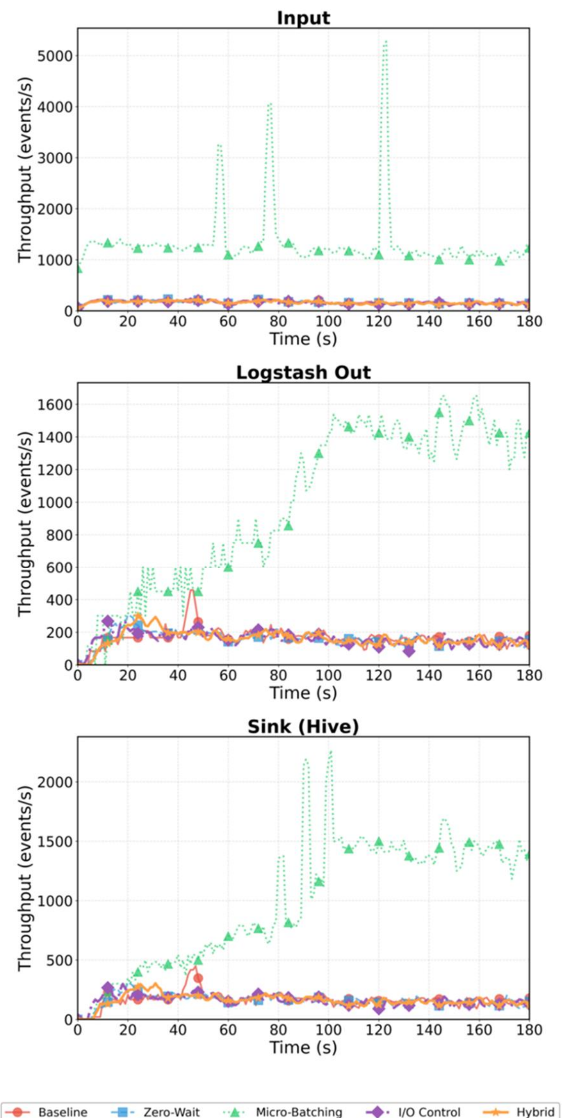

Fig. 9. Stage-wise Throughput Comparison across Optimization Scenarios

Table 9는 전체 5가지 실험 시나리오의 핵심 성능 지표를 종합한 결과이다. TP는 처리량을, I/O Control은 Collection-Layer I/O Control을 의미한다. 평균 및 P95 지연 시간은 초기 버퍼 채움과 웜업 구간을 포함한 전체

www.kci.go.kr

180초의 End-to-End 데이터를 기준으로 산출하였다. 또한 정상 상태에서의 지속 처리 성능을 비교하기 위해 초기 40초를 제외한 t ≥ 40s 구간의 평균 및 P95 지연 시간을 Steady Mean E2E와 Steady P95 E2E로 함께 제시하였다. p-value는 Baseline 대비 반복 실험 단위의 P95 지연 시간에 대해 Welch's t-test를 적용한 결과이다.

Table 9. Comprehensive Comparison of Key Performance Metrics across Scenarios

Metric	Baseline	Zero-Wait	Micro-Batch	I/O Control	Hybrid
Mean E2E (s)	15.19 ± 8.02	15.13 ± 7.79	65.88 ± 33.95	14.17 ± 8.25	14.42 ± 7.57
P95 (s)	31.28	30.18	113.97	30.77	28.45
Peak TP (req/s)	449	309	2,267	298	303
Packets (pck/s)	8,303	9,266	7,524	8,112	9,161
Steady Mean E2E (s)	18.04	18.23	75.80	17.40	17.52
Steady P95 E2E (s)	32.57	31.41	114.54	31.51	29.81
p-value vs. Baseline	-	0.1072	<0.0001	0.1679	0.0402
System Stability	Stable (high latency)	Unstable (throttling)	Unstable (spikes)	Stable	Stable / balanced

Table 9에서 하이브리드 실험의 최대 처리량(303 req/s)이 기본 설정(449 req/s)보다 낮게 나타난 것은 파이프라인의 성능 저하가 아니라 트래픽 셰이핑의 의도된 결과이다. Baseline은 데이터를 거대한 버퍼에 가두었다가 한 번에 밀어내어 순간적인 최대 처리량만 높게 측정될 뿐, 극심한 꼬리 지연(P95 31.28초)을 동반한다. 반면 하이브리드 설정은 유입 스파이크를 억제함으로써 최대 처리량은 낮아졌으나, 파이프라인 전 구간에서 지연이 급격히 증가하지 않는 안정적인 처리 흐름을 보였다.

Table 9의 full-run 결과는 전체 180초 기준이며, t ≥ 40s 결과는 안정화 이후 구간을 기준으로 산출하였다. 웜업 구간을 제외해도 시나리오 간 경향은 유지되었으며, Micro-Batching Optimization은 정상 상태에서도 높은 지연을 보여 백로그 누적 영향이 지속됨을 확인하였다. Welch's t-test 결과, Micro-Batching Optimization과 Hybrid Optimization은 각각 Baseline 대비 유의한 지연 증가와 감소를 보였으나, Zero-Wait Transmission과 Collection-Layer I/O Control은 유의한 차이를 보이지 않았다.

종합하면, Hybrid Optimization 실험은 평가된 시나리오 중 가장 낮은 P95 지연 시간(28.45초)을 보였으며, Baseline 대비 P95 지연 시간을 31.28초에서 28.45초로 약 9% 감소시켰다. 또한, Micro-Batching Optimization 실험에서 관찰된 파이프라인 붕괴 현상을 완전히 방지하고, 초당 300건 이상의 안정적인 처리량을 유지하였다. 이러한 결과는 본 연구에서 제안한 수집 계층의 I/O 제어와 후위 계층의 Zero-Wait Transmission을 결합한 계층별 하이브리드 전략이 P95 꼬리 지연 시간과 처리 안정성 간의 균형을 개선하는 데 효과적임을 보여준다.

5. Sensitivity Analysis

Hybrid Optimization의 주요 파라미터 영향을 확인하기 위해 scan_frequency, bulk_max_size, linger_ms를 대상으로 OVAT 분석을 수행하였다. 각 변형은 하나의 파라미터만 변경하고 나머지 설정은 Hybrid와 동일하게 유지한 상태에서 5회 반복하였다. 분석 결과는 Table 10에 제시하였다. Table 10에서 OVAT-1, OVAT-2, OVAT-3은 각각 scan_frequency를 10ms에서 1s로, bulk_max_size를 50에서 2048로, linger_ms를 0에서 5ms로 변경한 경우이다. 각 값은 반복 결과의 평균과 표준편차이며, Welch's t-test는 반복 단위 P95 값을 기준으로 Hybrid와 비교하였다.

Table 10. OVAT Sensitivity Analysis of Key Hybrid Parameters

Variant	Change Parameter	Hybrid → OVAT	Mean E2E(s)	P95 Latency(s)	Welch's p vs Hybrid
Hybrid	-	-	14.79± 0.38	29.06 ± 2.37	-
OVAT-1	scan_frequency	10ms → 1s	5.86 ± 0.05	25.92 ± 0.74	0.0389
OVAT-2	bulk_max_size	50 → 2048	6.08 ± 0.09	25.14 ± 0.61	0.0187
OVAT-3	linger_ms	0 → 5ms	6.42 ± 0.13	27.09 ± 1.65	0.1698

OVAT-1과 OVAT-2는 Hybrid보다 낮은 평균 지연 시간과 P95 지연 시간을 보였다. 특히 OVAT-2는 bulk_max_size를 2048로 설정했음에도 Micro-Batching Optimization과 달리 지연 누적이 발생하지 않았다. 이는 Micro-Batching의 성능 저하가 배치 크기만의 영향이 아니라, 수집 계층의 파일 스캔 주기와 후위 계층 배치 설정의 결합 효과였음을 시사한다. 반면 OVAT-3은 P95 지연 시간이 다소 감소하였으나 통계적으로 유의하지 않아, 본

www.kci.go.kr

실험 환경에서는 Kafka의 짧은 대기 시간 조정보다 수집 계층의 유입 제어가 더 큰 영향을 미친 것으로 나타났다. 따라서 OVAT 결과는 Hybrid가 모든 단일 지표에서 절대적으로 최적이라는 의미보다는, 수집 계층의 I/O 제어와 후위 계층의 저대기 전송을 결합하는 설계 방향의 타당성을 보완적으로 보여준다. 또한 Hybrid의 작은 bulk_max_size 설정은 안정 부하에서의 최소 P95만을 위한 것이 아니라, 유입 변동 시 순간 부하 전파를 줄이기 위한 보수적 설정으로 해석할 수 있다.

6. Burst Workload Robustness Check

시간에 따라 유입량이 변하는 상황에서 각 설정의 지연 특성을 비교하기 위해 시간 기반 버스트 부하 실험을 추가로 수행하였다. 실험은 전체 180초 동안 진행되었으며, 0-60초는 정상 부하, 60-90초는 정상 부하의 2배 수준의 버스트 부하, 90-180초는 회복 구간으로 구성하였다. 평가 대상은 Baseline, Micro-Batching Optimization, Hybrid Optimization의 세 가지이며, 기존 정상 부하 실험과 구분하기 위해 각각 B-Baseline, B-Micro-Batching, B-Hybrid로 표기하였다. 분석 결과는 Table 11에 제시하였다.

Table 11. Burst Workload Robustness Results

Scenario	P95 during burst (s)	Max stage lag (s)	Recovery P95 (s)	Observation
B-Baseline	39.11 ± 3.99	0.05	44.66 ± 2.46	Delayed recovery
B-Micro-Batching	35.73 ± 5.83	7.99	41.27 ± 2.37	Downstream lag
B-Hybrid	35.58 ± 3.61	0.08	38.75 ± 3.11	Stable recovery

Table 11에서 B-Baseline은 버스트 구간의 P95 지연 시간이 39.11초로 증가하였고, 회복 구간에서도 44.66초로 유지되어 지연이 즉시 감소하지 않았다. 이는 버스트 구간에서 발생한 대기 누적이 이후 구간에도 영향을 미친 결과로 볼 수 있다. B-Micro-Batching은 버스트 구간의 P95가 35.73초로 Baseline보다 낮았으나, Kafka-Spark 단계의 최대 지연이 7.99초로 나타나 후위 계층의 적체가 관찰되었다.

반면 B-Hybrid는 버스트 구간에서 P95 35.58초, 회복 구간에서 P95 38.75초를 보였으며, 버스트 이후 회복 구간에서의 P95 증가 폭도 가장 작았다. 이는 Hybrid 구성이 일시적인 유입 증가 상황에서도 후위 계층의 부하 집중을 완화하고, 지연 회복 측면에서 상대적으로 안정적인 특성을 보였음을 시사한다.

VI. Conclusions

본 연구에서는 클라우드 네이티브 환경 기반의 대규모 로그 데이터 파이프라인을 구축하고, 5단계의 파라미터 튜닝 시나리오를 통해 End-to-End 지연 시간과 시스템 리소스 간의 상충관계를 실증적으로 분석하였다. Baseline은 자원 사용은 안정적이나 누적 버퍼링으로 인한 심각한 지연을 유발하였으며, 완전 저지연 설정은 레이턴시는 감소하였으나 패킷 폭증으로 인해 전체 시스템의 안정성 및 처리량을 저해함을 확인하였다. 마이크로 배치 설정은 앞단의 네트워크 효율을 개선하였으나 후위 컴포넌트의 처리 한계를 초과하는 스파이크를 유발하여 파이프라인 붕괴를 초래하였다. Collection-Layer I/O Control을 통해 이러한 스파이크를 해소할 수 있음을 확인하였으며, 최종적으로 전위 계층의 스캔 주기 및 마이크로 배치를 통해 디스크/네트워크 부하를 능동적으로 제어하고, 후위 계층에 저지연 처리를 적용한 Hybrid Optimization 모델이 P95 지연 시간과 처리 안정성 간의 균형을 개선할 수 있음을 확인하였다. 추가적으로 수행한 OVAT 분석에서는 수집 계층의 파일 스캔 주기와 배치 크기 조합이 전체 지연 및 안정성에 중요한 영향을 미침을 확인하였다. 이는 본 연구에서 제안한 계층별 역할 분리, 즉 수집 계층에서는 유입 변동을 완화하고 후위 계층에서는 불필요한 전송 대기를 줄이는 설계 방향을 뒷받침한다. 또한 버스트 부하 실험에서도 Hybrid 구성은 일시적인 유입 증가 이후 상대적으로 안정적인 회복 특성을 보여, 시간적으로 변동하는 부하 상황에서도 적용 가능성이 있음을 확인하였다.

본 연구에서 제안한 계층별 Hybrid Optimization 전략은 특정 파이프라인에서 검증되었으나 그 핵심 원리인 수집 계층의 I/O를 능동적으로 제어하여 후위 컴포넌트의 부하를 평탄화하는 트래픽 셰이핑은 유사한 직렬 구조를 가진 이기종 파이프라인(예: Fluentd-RabbitMQ-Flink 조합 등)에도 적용 가능성이 있을 것으로 판단된다. 다만 최적의 파라미터 임계값은 하드웨어 사양, 네트워크 대역폭, 페이로드 특성에 따라 달라질 수 있으므로, 타 환경 적용 시 본 연구의 시나리오 설계 방법론에 따른 독립적인 튜닝 실험이 선행되어야 한다.

그럼에도 불구하고 본 연구는 몇 가지 한계점을 가지며, 이를 향후 연구를 통해 극복하고자 한다. 첫째, 실험 환경이 KakaoCloud의 특정 인스턴스 타입(m2a 시리즈)에 국한되어 있어, 타 클라우드 환경(AWS, GCP 등) 또는 온프레미스 환경에서 동일한 성능이 재현됨을 보장하지는 않는다. 본 연구에서 제안한 계층별 Hybrid Optimization의

www.kci.go.kr

핵심 원리는 인프라에 비종속적이므로, 향후 AWS, GCP 등 멀티클라우드 환경에서의 교차 검증 실험을 수행하여 전략의 적용 가능성을 추가 검증할 계획이다. 둘째, 본 연구의 트래픽 부하는 확률적 FSM 기반의 합성 부하로 생성되어 실험 기간 동안 일정한 강도로 인가되었다. 이로 인해 실제 운영 환경에서 관찰되는 시간대별 트래픽 변동, 플래시 세일과 같은 극단적인 버스트 패턴, 그리고 가변 크기의 페이로드를 온전히 반영하지 못한다는 한계가 있다. 향후 연구에서는 실제 운영 트래픽 로그를 재생하는 방식의 실험을 추가하여, 동적이고 불규칙한 부하 환경에서의 파라미터 민감도 분석을 수행할 예정이다. 셋째, 각 시나리오의 측정 시간이 180초로 제한되어 있어 장기 운영 환경에서의 메모리 누수 및 안정성 이슈를 검증하기에는 한계가 있다. 이를 보완하기 위해 24시간 이상의 연속 실험과 메모리 누수 모니터링을 포함한 장기 내구성 테스트를 후속 연구로 계획하고 있다.

본 연구는 이러한 한계에도 불구하고, 분산 데이터 파이프라인의 계층별 파라미터가 전체 성능에 미치는 연쇄적 영향을 종합적으로 분석하고, 실무 환경에서 참고 가능한 Hybrid Optimization 가이드라인을 제시한 데 의의가 있다. 향후 연구 과제로는 유입되는 트래픽의 동적 변화 패턴을 머신러닝 기법으로 학습하여, 런타임 환경에서 각 컴포넌트의 버퍼 크기와 스캔 주기를 동적으로 조절하는 오토 튜닝(Auto-Tuning) 알고리즘을 설계하고 적용할 계획이다.

ACKNOWLEDGEMENT

This research was supported by the MSIT(Ministry of Science, ICT), Korea, under the National Program for Excellence in SW, supervised by the IITP(Institute for Information communications Technology Planning&Evaluation) in 2026(2021-0-01440)

REFERENCES

[1] D. Reinsel, J. Gantz, and J. Rydning, "The Digitization of the World – From Edge to Core," International Data Corporation (IDC), Framingham, MA, USA, White Paper US44413318, Nov. 2018.

[2] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A Distributed Messaging System for Log Processing," Proc. 6th Int. Workshop on Networking Meets Databases (NetDB), pp. 1–7, Athens, Greece, Jun. 2011.

[3] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized Streams: Fault-Tolerant Streaming Computation at Scale," Proc. 24th ACM Symp. on Operating Systems Principles (SOSP), pp. 423–438, Farmington, PA, USA, Nov. 2013. DOI: 10.1145/2517349.2522737

[4] H. Wu, Z. Shang, and K. Wolter, "Performance Prediction for the Apache Kafka Messaging System," Proc. 2019 IEEE 21st Int. Conf. High Perform. Comput. Commun. (HPCC/SmartCity/DSS), pp. 154–161, Zhangjiajie, China, Aug. 2019. DOI: 10.1109/HPCC/SmartCity/DSS.2019.00036

[5] S. Vyas, R. K. Tyagi, C. Jain, and S. Sahu, "Performance Evaluation of Apache Kafka – A Modern Platform for Real Time Data Streaming," Proc. 2022 2nd Int. Conf. Innovative Practices in Technology and Management (ICIPTM), pp. 465–470, Gautam Buddha Nagar, India, Feb. 2022. DOI: 10.1109/ICIPTM54933.2022.9754154

[6] G. Hesse, C. Matthies, and M. Uflacker, "How Fast Can We Insert? An Empirical Performance Evaluation of Apache Kafka," Proc. 2020 IEEE 26th Int. Conf. Parallel Distrib. Syst. (ICPADS), pp. 641–648, Hong Kong, Dec. 2020. DOI: 10.1109/ICPADS51040.2020.00089

[7] I. Pelle, B. Szőke, A. Fayad, T. Cinkler, and L. Toka, "A Comprehensive Performance Analysis of Stream Processing with Kafka in Cloud Native Deployments for IoT Use-cases," Proc. NOMS 2023 IEEE/IFIP Network Operations and Management Symposium, pp. 1–6, Miami, FL, USA, May 2023. DOI: 10.1109/NOMS56928.2023.10154377

[8] K. Padmanaban, T. R. Ganesh Babu, K. Karthika, B. Pattanaik, Dhanabhavithra K, and C. Srinivasan, "Apache Kafka on Big Data Event Streaming for Enhanced Data Flows," Proc. 2024 8th Int. Conf. I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), pp. 977–983, Kirtipur, Nepal, Oct. 2024. DOI: 10.1109/I-SMAC61858.2024.10714884

[9] S. Das, S. Goswami, and C. N. Subalalitha, "Optimizing Apache Kafka Deployments – Configuration Customization is All You Need," Proc. 2023 14th Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT), pp. 1–9, Jul. 2023. DOI: 10.1109/ICCCNT56998.2023.10307242

[10] R. Adila, A. B. Nusantara, and U. L. Yuhana, "Optimization Techniques for Data Consistency and Throughput Using Kafka Stateful Stream Processing," Proc. 2023 6th Int. Semin. Res. Inf. Technol. Intell. Syst. (ISRITI), pp. 480–485, Dec. 2023. DOI: 10.1109/ISRITI60336.2023.10467675

[11] T. Aung, H. T. Zaw, A. H. Maw, and M. T. Mon, "Comprehensive Analysis: Monitoring Apache Kafka with Grafana, JMX Exporter, and Prometheus," Proc. 2024 5th Int. Conf. Advanced Information

www.kci.go.kr

Technologies (ICAIT), pp 1-6, Nov. 2024. DOI: 10.1109/ICAIT 65209.2024.10754944

[12] H. Wu, Z. Shang, G. Peng, and K. Wolter, "A Reactive Batching Strategy of Apache Kafka for Reliable Stream Processing in Real-time," Proc. 2020 IEEE 31st Int. Symp. Softw. Reliab. Eng. (ISSRE), pp. 207-217, Coimbra, Portugal, Oct. 2020. DOI: 10. 1109/ISSRE5003.2020.00028

[13] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, “Apache Flink™: Stream and Batch Processing in a Single Engine,” Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, vol. 36, no. 4, pp. 28-38, Dec. 2015.

[14] G. Siachamis, K. Psarakis, G. Christodoulou, M. Fragkoulis, A. van Deursen, and A. Katsifodimos, “Evaluating Stream Processing Autoscalers,” in Proc. 18th ACM International Conference on Distributed and Event-Based Systems (DEBS ’24), Villeurbanne, France, Jun. 2024, pp. 110-122, doi: 10.1145/ 3629104.3666036.

Authors

Jiman Cha is currently an undergraduate student in the Division of Computer Science and Engineering at Sahmyook University, Korea, and in his fourth year of the B.S. program.

Jiman Cha has been studying as an undergraduate researcher and serving as the leader of the student club, Cloud Lab, in the Division of Computer Science and Engineering at Sahmyook University, Korea. His research interests include Cloud Computing and Cloud Engineering.

Choong-hee Cho received the B.S. degree in Computer Engineering from Sahmyook University, Korea, in 2010, and the Ph.D. degree from the Korea University of Science and Technology, Korea, in 2019.

Dr. Cho joined the faculty of the Department of Data Cloud Engineering at Sahmyook University, Seoul, Korea, in 2022. He is currently an Assistant Professor in the Department of Data Cloud Engineering, Sahmyook University. His research interests include cloud architecture and microservices architecture (MSA).

www.kci.go.kr