서버 사이드 렌더링과 스트리밍 서버 사이드 렌더링 간 성능 분석을 기반으로 한 Hydration 병목 완화 설계 가이드라인 제시

조민희, 유명식
숭실대학교

neroid34@gmail.com, myoo@ssu.ac.kr

Proposing Design Guidelines for Mitigating Hydration Bottlenecks based on Performance Analysis between Server-Side Rendering and Streaming Server-Side Rendering

Minhee Cho, Myungsik Yoo
Soongsil Univ.

요 약

최근 웹 애플리케이션이 널리 사용하게 되면서 렌더링 전략이 중요시되고 있다. 초기 로딩 성능과 검색 엔진 최적화라는 측면에서 Server-Side Rendering(SSR)이 Client-Side Rendering(CSR)보다 주목을 받고 있으나, hydration 과정으로 인해 상호작용이 지연되는 문제가 발생하고 있다. 이를 완화하고자 Streaming Server-Side Rendering(Streaming SSR)이 도입되었지만, hydration 병목에 대한 실질적 효과는 명확히 검증되지 않았다. 본 논문은 SSR 과 Streaming SSR 을 동일한 조건에서 비교 분석하여 hydration 병목 현상에 미치는 영향을 평가하고 결과를 바탕으로 hydration 병목 완화 설계 가이드라인을 제시하였다.

Ⅰ. 연구 배경 및 필요성

최근 웹 애플리케이션이 단순한 정적인 화면을 제공하는 것을 넘어, 대규모 데이터 처리와 복잡한 사용자 상호작용을 요구하는 형태로 발전함에 따라서 프론트엔드 렌더링 방식은 단순한 구현 방식의 선택에서 웹 성능과 검색 엔진 최적화(SEO), 그리고 사용자 체감 반응성에 직접적인 영향을 미치는 핵심 설계 요소로 자리잡았다. [1] [2]

Client-Side Rendering(CSR)은 서버에서 거의 비어있는 초기 HTML 을 클라이언트로 전송한 후 클라이언트가 JavaScript 실행을 통해 화면을 동적으로 구성하는 방식으로 풍부한 상호작용 제공하지만 JavaScript 번들이 다운로드 및 실행되기 전까지 초기 HTML 만 보여지는 한계가 존재한다. [3] [4] 이런 구조적 한계때문에 콘텐츠 중심의 웹 애플리케이션이나 SEO 가 중요한 서비스에서는 SSR 을 사용한다. [1] [5] SSR 은 CSR 과 달리 서버에서 HTML 을 완성하고 클라이언트로 전달하여, 클라이언트가 JavaScript 를 실행하고 hydration 과정 이후 상호작용이 가능하기 때문에 First Contentful Paint(FCP), Largest Contentful Paint(LCP)와 같은 초기 로딩 지표가 개선되었다. React 18 이후 도입된 Streaming Server-Side Rendering(Streaming SSR)은 SSR 에서 확장된 형태로, HTML 을 한번에 전달하지 않고 점진적으로 클라이언트에 전달하는 방식으로 빠른 초기 시각적 완성도를 보이고 있으며 사용자의 체감 성능을 향상시켰다. [6] [7]

그러나, SSR 과 Streaming SSR 은 hydration 과정이 필요하다. hydration 은 서버에서 전달된 HTML 에 클라이언트의 JavaScript 를 결합하여 상호작용이 가능하도록 하는 과정으로, 이 과정에서 대규모 JavaScript 실행이 메인 스레드를 점유하여, 사용자가 상호작용할 수 없는 hydration 병목 현상이 발생한다. [8] [9] [10]

Streaming SSR 이 초기 로딩과 사용자 체감 성능을 개선시키는 데에 효과적임을 보고하는 기존 연구는 존재하나, hydration 병목 현상을 구조적으로 완화하는지에 대해서는 연구가 부족하다. 따라서 본 논문은 SSR 과 Streaming SSR 을 동일한 조건에서 hydration 병목 현상에 미치는 영향을 분석하여, 분석 결과를 바탕으로 Hydration 병목을 완화할 수 있는 설계 가이드라인을 제시하고자 한다.

Ⅱ. 연구 방법

1. 연구 설계

비교를 위한 웹 애플리케이션은 Next.js App Router 기반으로 Node 는 20.19.6 버전, React 는 19.2.3 버전, Next.js 버전은 16.1.0 버전으로 구현하였다. 동일한 UI 를 유지하면서 서버 컴포넌트와 클라이언트 컴포넌트를 분리하였다. hydration 병목이 충분히 관찰될 수 있도록 클라이언트 렌더링 작업량과 서버 응답 지연시간을 5000 개와 1500ms 로 설정했다. 해당 조건 하에서 SSR 과 Streaming SSR 을 동일한 조건에서 Google Lighthouse 지표와 애플리케이션 내부 지표를 함께 자동화된 측정 파이프라인을 사용하여, 200 회 반복 측정하였다.

2. 연구 결과

표 1 은 FCP, LCP, Total Blocking Time(TBT)과 같은 초기 로딩 지표를 Lighthouse CI 를 이용하여 측정하여 평균값을 계산한 표이다.

표 1. Google Lighthouse 지표

지표	SSR	Streaming SSR
FCP	920.04 ms	917.58 ms
LCP	1688.49 ms	1689.17 ms
Seed Index	2678.11 ms	964.82 ms

TTI	2437.37 ms	2349.08 ms
TBT	45.30 ms	41.54 ms

표 2 는 웹 애플리케이션 내부에서 최상위 클라이언트 컴포넌트가 mount 되는 시점을 기준으로 한 hydration 완료 근사값과, hydration 및 초기 실행 과정에서 발생한 long task 의 총 지속 시간, 최대 지속 시간, 발생 횟수를 측정하여 평균값을 계산한 표이다.

표 2. Hydration 및 메인 스레드 실행 지표

지표	SSR	Streaming SSR
Hydration 완료 시점 근사값	0.20 ms	0.20 ms
Long Task 총 시간	2398.05 ms	2398.05 ms
Long Task 최대 시간	2294.03 ms	2294.03 ms
Long Task 횟수	2.00 회	2.00 회

Ⅲ. 결과 분석

Speed Index 와 hydration 완료 시점과의 상관관계를 확인하기 위해 피어슨 상관관계 계수와 스피어만 상관관계 계수를 이용하여 Speed Index 와 hydration 완료 시점간의 상관관계를 분석하였다. 표 3 은 Streaming SSR 과 SSR 에서 Speed Index 와 hydration 완료 시점간의 상관관계를 분석한 결과표로 Streaming SSR 에서 Speed Index 의 큰 변동이 hydration 이 완료되는 시점과 거의 독립적으로 발생했다는 것을 확인할 수 있다.

표 3. Speed Index 와 hydration 완료 시점간의 상관계수

렌더링방식	Pearson	Spearman
SSR	0.461	-0.025
Streaming SSR	0.205	-0.087

hydration 병목이 렌더링 방식의 차이에서 발생하는 문제가 아니라, 클라이언트 측 실행 구조에서 기인한 것을 받침한다. hydration 병목 완화를 위해서는 단순한 렌더링 전략 변경이 아닌, 실행 범위 최소화와 선택적 hydration 을 중심으로 한 구조적 설계가 필요하다. 또한, Streaming SSR 은 hydration 병목을 직접적으로 해결하는 기술이 아니라, 초기 로딩 경험을 개선하기 위한 보조적 렌더링 전략으로 활용되어야 한다.

종합적으로 Streaming SSR 은 초기 로딩과 사용자 체감 성능을 유의미하게 개선하나 hydration 병목을 완화시킨다. 위의 결론을 바탕으로 하여 다음과 같은 hydration 병목 완화를 위한 설계 가이드라인을 도출했다.

1, Streaming SSR 은 hydration 병목을 해결하기 위한 수단으로 간주하지 말아야 한다.

2. hydration 병목 완화는 초기 인지 성능 개선과 분리된 설계 문제로 접근해야 한다.

3. Streaming SSR 을 hydration 병목 현상을 완화시키는 기법으로 사용하기 위해서는 다른 구조적 기법과 결합되어야 한다.

Ⅳ. 결론

동일한 조건에서 SSR 과 Streamin SSR 을 비교한 결과, Streaming SSR 은 Speed Index 를 중심으로 초기 사용자 체감 성능을 유의미하게 개선하였으나, hydration 과정에서 발생하는 long task 및 메인 스레드 병목은 SSR 과 동일한 수준으로 유지되었다. 본 논문에서 hydration 병목 완화 문제에 대한 설계 가이드라인을 제안하였다. 제안한 가이드라인은 향후 대규모 웹 애플리케이션에서 hydration 성능 문제를 분석하고 설계 전략을 수립하는데 기준으로 활용될 수 있을 것으로 기대된다. 본 연구에서는 전체 페이지가 동일한 방식으로 hydration 되는 구조로 실험을 진행하였다. 향후 연구에서는 부분 hydration 및 선택적 hydration 기법과 같은 구조적 hydration 최적화 기법과 Streaming SSR 이 결합될 경우 실제로 hydration 병목이 완화되는지 분석이 필요하다.

참 고 문 헌

[1] Iskandar, Taufan Fadhilah, et al. "Comparison between client-side and server-side rendering in the web development." IOP Conference Series: Materials Science and Engineering. Vol. 801. No. 1. IOP Publishing, 2020.

[2] Jain, Vivek. (2021). Server-Side Rendering vs. Client-Side Rendering: A Comprehensive Analysis. 7. 1-5. 10.5281/zenodo.14752604. (https://www.researchgate.net/publication/390066628_Server-Side_Rendering_vs_Client-Side_Rendering_A_Comprehensive_Analysis)

[3] A. Alkhaled, O. Cosmin, “Rendering Matters: SSR vs CSR in Modern Web Development,” Dissertation, 2025.

[4] Pati, Swostik, and Yasir Zaki. "Evaluating the Efficacy of Next. js: A Comparative Analysis with React. js on Performance, SEO, and Global Network Equity." Companion Proceedings of the ACM on Web Conference 2025. 2025.

[5] Simao, T. (2025). Evaluating Server-Side and Client-Side Rendering for Content-Heavy Web Applications.

[6] Abramov, D., et al. (2022). *React 18 and concurrent rendering*. Meta Platforms Technical Report. (https://react.dev/blog/2022/03/29/react-v18)

[7] Hulthén, J. (2024). Streaming Server-Side Rendering: An Empirical Study on Page Performance, Server Load, and User Experience: Comparing streaming Streaming Server-Side Rendering to Standard Server-Side Rendering.

[8] Chen, Kaitao. "Improving Front-end Performance through Modular Rendering and Adaptive Hydration (MRAH) in React Applications." *arXiv preprint arXiv:2504.03884* (2025).

[9] React Working Group. (2021). *Discussion #37: Streaming server rendering with Suspense*. GitHub. (https://github.com/reactwg/react-18/discussions/37)

[10] Carvalho, Fernando Miguel. (2024). Progressive Server-Side Rendering with Suspendable Web Templates. 10.1007/978-981-96-0576-7_33. (https://www.researchgate.net/publication/387265879_Progressive_Server-Side_Rendering_with_Suspendable_Web_Templates)