임베디드 Node.js의 반응 시간 향상을 위한 이벤트 콜백 병렬화

홍경환○, 신동군
성균관대학교 정보통신공학부
RedCarrottt@gmail.com, dongkun@skku.edu

Event Callback Parallelization for Improving Response Time in Embedded Node.js

Gyeonghwan Hong○, Dongkun Shin
School of Information and Communication Engineering, Sungkyunkwan University

요 약

기존의 서버 환경에 맞추어 설계된 Node.js가 단일 쓰레드 이벤트 방식을 사용하기 때문에 스마트 TV등 임베디드 서버에서는 멀티 코어 프로세서를 충분히 활용하지 못하며, 반응 시간도 저하된다. 본 연구에서 제시하는 '이벤트 콜백 병렬화 기법'은 수행 시간이 긴 CPU-bound 이벤트를 자식 프로세스에 위임하여 '이벤트 콜백 기아 현상'을 해결하고 반응 시간 향상을 꾀하는 기법이다. 이를 실현할 때 발생하는 문맥 고립 문제를 해결한 문맥 전달 기법도 제시하고 있다. 실험을 통해 프로세서 코어 개수 만큼의 자식 프로세스들이 동작할 때 가장 큰 효율로 이벤트 반응 시간이 향상됨을 보였다.

1. 서 론

멀티 코어 임베디드 프로세서의 등장으로 스마트 폰과 같은 고성능 모바일 장치를 실현할 수 있게 되었고, 스마트 TV에도 영향을 미쳤다. 최근 스마트 TV는 모바일 단말기로 입력받아 컨텐츠를 제공하는 임베디드 서버로서 주목받고 있다.

그러나 지금 스마트 TV 는 기존의 스마트 폰과는 달리 앱 생태계가 형성되는 과정에 있으며, 심지어 프레임워크 소프트웨어도 확립된 것이 많지 않다. 이런 이유 때문에 많은 스마트 TV들은 앱 생태계와 프레임워크를 빠른 시간에 구축할 수 있으면서 컨텐츠를 제공할 수 있는 서버 소프트웨어인 Node.js를 핵심 컴포넌트로 사용하고 있다.

아파치(Apache)나 IIS (Internet Information Services) 같은 기존의 서버 소프트웨어는 대형 서버 환경에서 지나친 멀티 쓰레딩 때문에 빈번하게 일어나는 문맥 교환(context switching)으로 성능이 크게 저하되는 문제가 있었다. 이 때문에 Node.js는 단일 쓰레드 이벤트 드리븐(event-driven) 구조를 도입하고 I/O 쓰레드 풀(I/O thread pool)을 사용하여 기존 서버의 문제를 해결하였다[1].

그러나 스마트 TV에서는 대형 서버 환경과는 달리, 동시 접속자 수가 적고 멀티미디어 관련 CPU-bound 작업을 수행하는 경우가 많다. 또한 멀티 코어 프로세서를 사용하기 때문에, 충분히 CPU-bound 작업의 병렬성을 활용할 수 있다. 그럼에도 Node.js는 CPU-bound 작업을 한 쓰레드에서 사용하는 '단일 쓰레드 이벤트 드리븐 방식'을 채택하고 있기 때문에, 임베디드 환경에서는 Node.js의 반응 시간이 저하된다.

싱글 쓰레드 방식을 사용하는 자바스크립트(JavaScript)의 CPU-bound 작업을 병렬화하려는 연구는 많이 있었다. 자바스크립트를 인터프리터 단계에서 컴파일하는 ParaScript는 반복문이 많은 워크로드에서 병렬화를 꾀했다[2]. 또한 Node.js에서 멀티 프로세스와 멀티 쓰레드 성능을 비교하는 연구도 있었다[3]. 그러나 초기 설계부터 싱글 쓰레드를 가정하고 만들어진 자바스크립트 언어의 특성 상 프로세스나 쓰레드 간 문맥 고립 문제(context isolation)가 일어나기 때문에, 실제 서버 응용 프로그램에 그대로 적용하기에는 어려움이 많다[4].

본 논문에서는 멀티 프로세스를 이용하여 이벤트 드리븐 구조의 핵심인 이벤트 콜백을 병렬화하는 방안을 제시한다. 또한 멀티 프로세스 구조의 문맥 고립 문제를 부분적으로 해결하기 위한 이벤트 콜백 간 문맥 전달 기법을 제안한다.

2. Node.js

Node.js는 자바스크립트 인터프리터 엔진 V8을 기반으로 만들어진 서버 프로그램이며, libuv 라이브러리로 이벤트 드리븐 구조를 구현하였다. 대형 서버 같이 연결이 많은 환경에서는 쓰레드 간의 빈번한 문맥 교환으로 인한 성능 저하가 많기 때문에, Node.js에서는 단일 쓰레드를 사용하여 이벤트 드리븐 처리를 구현하였다. 그림 1과 같이, Node.js는

본 연구는 지식경제부 및 한국산업기술평가관리원의 산업융합원천기술개발사업(정보통신)의 일환으로 수행하였음. [KI0018-10041244, 스마트TV 2.0 소프트웨어 플랫폼]

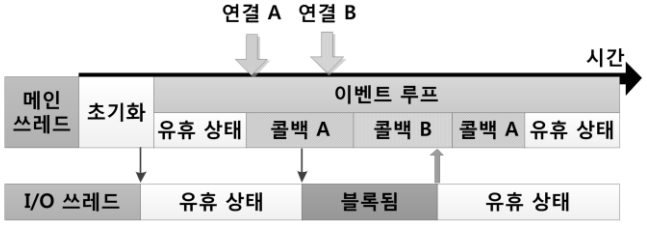

그림 1. Node.js의 이벤트 드리븐 구조 및 I/O 쓰레드

초기화 작업, 이벤트 루프(event loop)와 이벤트 콜백(event callback)이 모두 메인 쓰레드 한 곳에서 동작한다.

또한 기존 서버에서는 CPU-bound 작업보다는 I/O-bound 작업의 양이 많으며 이로 인해 소요되는 시간이 훨씬 많다는 점을 반영하여, I/O-bound 작업만 병렬화하였다. 그림 1에서 연결 이벤트 A가 들어오면 이 이벤트에 대한 콜백 A가 수행되고, 이 콜백에서 요청한 I/O 작업을 I/O 쓰레드에서 수행하며, 메인 쓰레드 대신 I/O 쓰레드가 블록되어서 콜백 A의 작업을 계속 할 수 있게 된다. I/O에 의한 블록이 끝나고 메인 쓰레드에서 진행되고 있던 콜백 작업도 끝나면 그 결과에 대한 작업을 수행하게 된다. 이를 통해 메인 쓰레드에서는 I/O로 인한 블록이 일어나지 않게 하였다[1].

3. 이벤트 콜백 병렬화

3.1 이벤트 콜백 기아 문제

그림 2의 (a)에서는 연결 이벤트 D가 연결 이벤트 A의 콜백이 끝나기 전부터 Node.js에 들어왔음에도 불구하고, 연결 이벤트 A의 콜백이 CPU에서 수행되는 시간이 길어서 연결 이벤트 D의 콜백 수행이 지연된다. 현재 Node.js에서는 병렬적으로 처리되는 I/O-bound 작업과는 달리, CPU-bound 작업은 하나의 메인 쓰레드에서 동작해야 한다. 이 때문에 특정 이벤트에 의한 콜백이 CPU 자원을 독점하고 다른 이벤트의 콜백 수행이 지연되는, '이벤트 콜백 기아(event callback starvation) 문제'가 일어나게 된다.

3.2 이벤트 콜백 병렬화

이런 이벤트 콜백 기아 문제는 이벤트 콜백 병렬화(event callback parallelization)를 통해 해결할 수 있다. 그림 2의 (b)와 같이, 먼저 메인 프로세스의 초기화 단계에서 자식 프로세스를 포크(fork)하고, 자식 프로세스에서도 새 Node.js 인스턴스를 만들어서 프로세스 풀을 구성한다. 그 후 메인 프로세스로 새 이벤트가 도착할 때마다 프로세스 풀에서 유휴 상태인 자식 프로세스에게 콜백 수행을 위임한다. 그렇지 못할 경우에는 메인 프로세스에서 이벤트를 처리하도록 한다.

기존의 서버 프로그램에서도 멀티 쓰레드를 사용하여 이벤트를 병렬적으로 처리하고자 하는 시도는 있었다. 그러나 이들은 지나치게 많은 쓰레드를 생성하여 문맥 교환이 빈번해졌고, 결국 시스템 성능 저하를 일으켰다. 이 때문에 본 연구에서 제안하는 기법에서는 자식 프로세스를 프로세스의 코어 개수만큼 생성하여 문맥 교환 비용을 최소화하였다.

3.3. 문맥 고립 문제 및 문맥 전달

Node.js는 단일 쓰레드를 사용하는 것을 전제로 한 언어인 자바스크립트를 사용하기 때문에, 서로 다른 쓰레드가 변수나

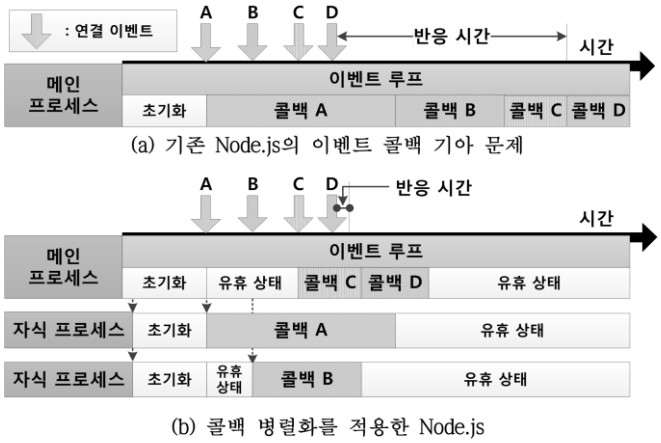

그림 2. 이벤트 콜백 기아 현상 및 이벤트 콜백 병렬화

함수를 공유할 수 없도록 강제하고 있다[4]. 이 때문에 Node.js에서 멀티 프로세스나 멀티 쓰레드를 사용하려면 자식을 생성할 때마다 처음부터 V8과 Node.js를 초기화하여 문맥을 새로 만들어야 하며, 각 프로세스/쓰레드의 문맥은 고립되어 서로 변수나 함수를 접근할 수 없다. 이를 문맥 고립(context isolation) 문제라고 한다.

부모 프로세스가 자식 프로세스에 실행되어야 하는 콜백 함수, 매개변수, 전역 변수를 전달해야 하는 콜백 병렬화 기법 구현 시 문맥 고립은 큰 문제점이다. 자바스크립트의 언어적 특성 상 이를 완전히 해결하기는 어려우나, Node.js에서 프로세스 간 텍스트 전달은 제공하고 있다는 점을 이용하여 부분적으로 해결할 수 있다. 본 기법에서는 이벤트가 들어올 때마다 매개 변수를 JSON (JavaScript Object Notation)으로 변환하고, 콜백 함수를 텍스트로서 자식 프로세스에 전달하도록 하였다. 이런 Node.js 프로세스 간 변수 및 함수 전달 과정을 '문맥 전달(context transmission)'이라고 하며, 이를 통해 문맥 고립 문제를 부분적으로 해결하였다.

그러나 자바스크립트 객체가 함수와 변수를 함께 포함하는 경우에는 JSON으로 표현할 수 없어서 모든 자바스크립트 객체를 프로세스 간에 전달하는 것은 불가능하다. 이 때문에 문맥 고립 문제를 완벽하게 해결했다고 볼 수는 없다. 또한 Node.js 실행 초기에 자식 프로세스를 생성할 때마다 문맥 초기화 시간과 프로세스 간 메시지 전달 비용이 추가된다.

4. 실험 및 분석

4.1 실험 환경

스마트 TV와 비슷한 환경에서 Node.js에 이벤트 콜백 병렬화를 적용하였을 때의 효과와 문맥 생성 및 전달 비용을 비교하고 분석하기 위해 실험 환경을 구축하였다.

실험은 듀얼 코어 1.2GHz ARM Cortex-A9을 사용하는 엑시노스 4210을 탑재한 ODROID-PC에서 진행되었다. 이 환경에 탑재된 DRAM은 1GB이며, OS는 Linaro 12.04 (Kernel 3.4.0)를, Node.js는 0.10.2 버전을 사용하였다.

4.2 이벤트 콜백 병렬화의 효과

본 실험에서는 Raspberry PI TV[5]라는 Node.js 기반 스마트 TV 프로젝트를 수정하여, 스마트 TV와 유사한 환경을

비교군	직렬화된 콜백	병렬화된 콜백
사용자 반응 시간 (ms)	697	11

표 1. 이벤트 콜백 병렬화의 효과

구성하였다. ODROID-PC에서는 스마트 폰으로 접속하여 조작하는 리모트 컨트롤 웹 페이지와 스마트 TV의 컨텐츠를 보여줄 웹 페이지를 제공하는 웹 서버가 구동된다. 이 환경에서는 다음 두 가지 요청을 처리하도록 하였다.

- 요청 1: 1920x1080 크기의 PNG 이미지 디코딩
  Node.js에서 주기적으로 타이머 이벤트가 발생되도록 하고, 이 이벤트에 대해 콜백에서 자바스크립트 기반 PNG 이미지 디코딩 모듈인 PNG-js로 이미지 디코딩을 수행하도록 했다. 수행 시간이 긴 CPU-bound 동작의 대표로 이를 선정하였다.
- 요청 2: 동영상 재생 요청
  사용자는 리모트 컨트롤러 웹 페이지에 있는 버튼을 누르게 되면, 네트워크를 통해 ODROID-PC에 동영상 재생 요청이 들어오게 된다. 이 요청이 ODROID-PC에 들어오게 되면 Node.js 외부에 mplayer 프로세스를 실행하게 된다. 이 요청에 의해 일어나는 동작은 요청 1에 비해 수행 시간이 짧은 CPU-bound 동작이다.

이 환경에서 기존의 직렬화된 콜백 방식과 본 연구에서 제안한 이벤트 콜백 병렬화를 각각 적용하여 실험하였다. 표 1과 같이, 사용자가 입력한 요청 2에 대한 반응 시간이 직렬화된 콜백 방식에서 697ms, 병렬화된 콜백 방식에서 11ms 소요되어, 이벤트 콜백 병렬화로 인해 사용자 반응 시간이 큰 폭으로 향상되었음을 확인할 수 있었다.

4.3 최적의 자식 프로세스 개수

이벤트 콜백 병렬화에서는 자식 프로세스의 개수가 시스템 성능을 좌우한다. 이 때문에 두 워크로드를 비교하며 최적의 자식 프로세스 개수를 도출하였다. CPU-bound 동작인 피보나치 함수 연산을 자식 프로세스 위의 콜백에서 동작하는 피보나치 워크로드(그림 3)와, 피보나치 워크로드에 I/O-bound 동작인 웹 서버가 메인 프로세스에서 함께 동작하는 혼합 워크로드(그림 4)를 수행하였다. 이 실험에서는 초기화 시간, 피보나치 연산 시간 및 그 외 오버헤드로 나누어 자식 프로세스의 수행 시간을 측정하였다.

듀얼 코어 AP에서 실험했기 때문에, 그림 3과 그림 4에서 볼 수 있듯이 자식 프로세스의 개수가 1개일 때에 비해 2개일 때 두 워크로드 모두 수행 시간이 약 4~7% 증가한다. 그러나 프로세스의 개수가 3개 이상이 되면, 모든 프로세스를 병렬적으로 처리하지 못하고 순차적으로 처리한다. 이 때문에 피보나치 연산 시간과 문맥 교환 비용이 늘어나, 수행 시간이 프로세스 한 개당 약 440~550ms씩 선형적으로 증가한다.

그림 4의 혼합 워크로드와 같은 경우, 웹 서버의 콜백 동작이 메인 프로세스 상에서 끼어들어 수행되기 때문에 그림 3의 피보나치 워크로드보다 수행 시간이 23% 더 길어졌으며, 그 외에는 비슷한 양상을 보이고 있다. 이를 통해 AP의 코어 개수만큼 자식 프로세스를 생성하면, 콜백의 동작 유형과 관계 없이, 콜백의 병렬성을 최대한 활용하면서도 문맥 교환

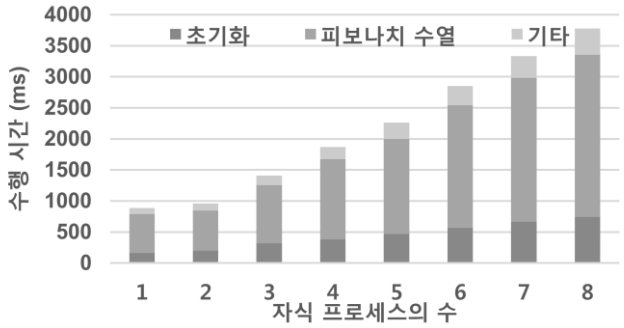

그림 3. 병렬화된 이벤트 콜백 수행 시간 (피보나치)

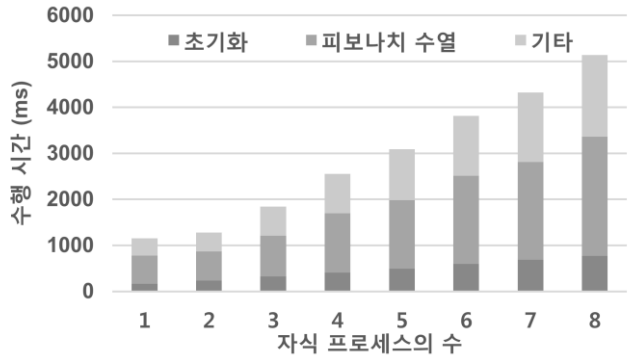

그림 4. 병렬화된 이벤트 콜백 수행 시간 (혼합)

비용이 적어서 가장 큰 효율을 보여준다는 점을 확인하였다.

5. 결 론

본 논문에서는 서버 소프트웨어 Node.js의 반응 시간이 저하되는 이벤트 콜백 기아 현상을 해결하는 이벤트 콜백 병렬화 기법을 제시하였다. 또한 이벤트 콜백 병렬화의 구현에 가장 큰 걸림돌이 되는 문맥 고립 문제를 해결하기 위한 문맥 전달 기법도 제시하였다.

이벤트 콜백 병렬화의 효과를 입증하기 위해 실제 스마트 TV와 비슷한 워크로드를 이용하여 반응 시간을 측정하였으며, 기존 방식에 비해 반응 시간이 큰 폭으로 향상되었다. 실험을 통해 콜백의 유형과 관계 없이, AP의 코어 개수 만큼 자식 프로세스를 생성했을 때 이벤트 콜백의 병렬성과 최소의 문맥 교환 비용 모두 만족한다는 점을 밝혔다.

참고 문헌

[1] Stefan Tilkov, et al. “Node. js: Using JavaScript to Build High-Performance Network Programs.” IEEE Internet Computing, 14.6, 2010.

[2] Mojtaba Mehrara, et al. “Dynamic parallelization of JavaScript applications using an ultra-lightweight speculation mechanism.” High Performance Computer Architecture, p.87-98, 2011.

[3] 김동훈, et al. “임베디드 환경 Node. js 의 병렬화와 메모리 중복 제거 효과 분석.”, 한국정보과학회 학술발표논문집, p.1290-1292, 2013.

[4] “V8 JavaScript Engine Embedder’s Guide”, https://developers.google.com/v8/embed

[5] “Raspberry Pi TV”, https://github.com/DonaldDerek/PiR.tv