임베디드 환경 Node.js의 병렬화와 메모리 중복 제거 효과 분석

김동훈[O1] 홍경환[1] 신동군[1]
[1]성균관대학교 정보통신공학부
kimbavel@gmail.com, redcarottt@gmail.com, dongkun@skku.edu

Parallelization and memory deduplication of Node.js in embedded environment

DongHun Kim[O1] GyeongHwan Hong[1] DongKun Shin[1]
[1]School of Information and Communication Engineering, SungKyunKwan University

요 약

전통적으로 서버 소프트웨어는 대형 엔터프라이즈 환경에서만 사용되었지만, 최근에는 서버 역할을 하는 임베디드 장치가 확산되면서 임베디드 서버 소프트웨어가 대두되고 있다. 임베디드 서버는 개인용 서버이기 때문에 확장성이 중요시되며, 제한된 하드웨어를 사용한다. 현재 많이 사용되고 있는 서버 소프트웨어 중 이런 점을 만족하는 것으로 Node.js가 있다. Node.js의 특징으로는 자바스크립트(JavaScript)를 사용하여 코드의 재사용성과 확장성이 좋다는 점과, 비동기식 I/O를 지원하여 I/O에 의한 이벤트 지연을 줄였다는 것이 있다. 이 논문에서는 1) Node.js의 문제점인 단일 프로세스에서 CPU-bound 작업을 처리하는 것에 대한 대안으로 멀티 프로세스(multi-process)와 멀티 쓰레드(multi-thread)를 비교하고, 2) 불가피하게 멀티 프로세스를 사용할 경우 KSM을 적용하여 메모리 사용의 효율을 높이는 방법을 제시한다.

1. 서 론

전통적으로 서버 소프트웨어는 대형 엔터프라이즈 환경에서만 사용되었지만, 최근에는 구글과 애플이 각각 크롬캐스트(Chromecast)와 애플 TV(Apple TV)라는 스마트 TV를 출시하는 등, 서버 역할을 하는 임베디드 장치가 확산되면서 임베디드 서버 소프트웨어가 대두되고 있다.

서비스를 제공한다는 면에서 임베디드 서버는 기존의 서버와 같은 위치에 있다. 그러나 기존의 서버와는 달리 개인용 서버이기 때문에 앱 생태계를 통한 확장성이 중요시되며, 저전력, 저가의 제한된 하드웨어를 사용한다는 점이 크게 다르다. 임베디드 서버 소프트웨어는 이런 요구사항을 만족시켜야 한다.

현재 많이 사용되고 있는 서버 소프트웨어로는, 아파치(Apache)와 Node.js가 있다. 아파치는 서버에서 동작하는 프로그램을 제작하려면 상당한 수고를 들여야 하는 한편, Node.js는 코드 재사용성 및 확장성이 뛰어난 자바스크립트(JavaScript)로 프로그램을 제작할 수 있다. 따라서 앱 생태계 확장성을 위해 Node.js를 사용할 수밖에 없다.

Node.js는 이벤트 드리븐(event-driven) 서버 프로그램이다. Node.js는 I/O-bound 작업과 CPU-bound 작업을 병렬화함으로써 단일 프로세스 구조에서의 이벤트 지연 현상을 해결하기 위해, I/O 쓰레드 풀(thread pool)과 클러스터(cluster) 모듈을 제공하고 있다.[1] 그러나 클러스터 모듈은 두 가지 문제점을 지니고 있다. 프로세스 생성 시 새로운 프로세스의 코드와 데이터를 초기화하는 CPU 오버헤드와, 프로세스 사이에 메모리가 중복되는 메모리 오버헤드가 발생한다. 이 방법은 CPU 및 메모리 자원이 극도로 제한된 임베디드 환경에서 감당하기 어렵다. 따라서 임베디드 환경에 맞게 Node.js의 멀티 프로세스 구조를 개선하여, 오버헤드를 줄여나가야 한다.

본 연구에서는 Node.js의 멀티 프로세스 구조를 개선하기 위해 두 가지 대안을 제시하였다. 첫째는 프로세스 간 메모리 중복을 제거하기 위해 멀티 프로세스에 KSM을 적용하는 방법이고, 둘째는 메모리 중복을 제거하면서 프로세스 초기화 CPU 오버헤드까지 개선한 멀티 쓰레드를 사용한 방법이다.

이 논문의 2장에서는 Node.js의 매커니즘을, 3장에서는 Node.js의 현재 작업 병렬화 방식의 문제점과 그 대안을 다룬다. 4장에서는 이 연구에서 제시한 대안의 효과를 분석한다. 마지막으로 5장에서는 결론과 향후 연구에 대해 논의한다.

2. Node.js

Node.js는 V8 자바스크립트 엔진을 기반으로 동작하는 이벤트 드리븐 서버 소프트웨어다.[2] V8 자바스크립트 엔진은 구글에서 개발한 자바스크립트 엔진이다. V8은 히든 클래스(hidden class), 인라인 캐시(inline cache) 등의 최적화 기법을 사용하여 자바스크립트 코드 인터프리팅 속도를 높였다.

사용자의 자바스크립트 코드는 실행 시간에 컴파일되어 그림 1과 같이 V8 컨텍스트(context) 내에서 동작하

이 논문은 2013년도 정부(교육부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임 (2013R1A1A2A10013598)

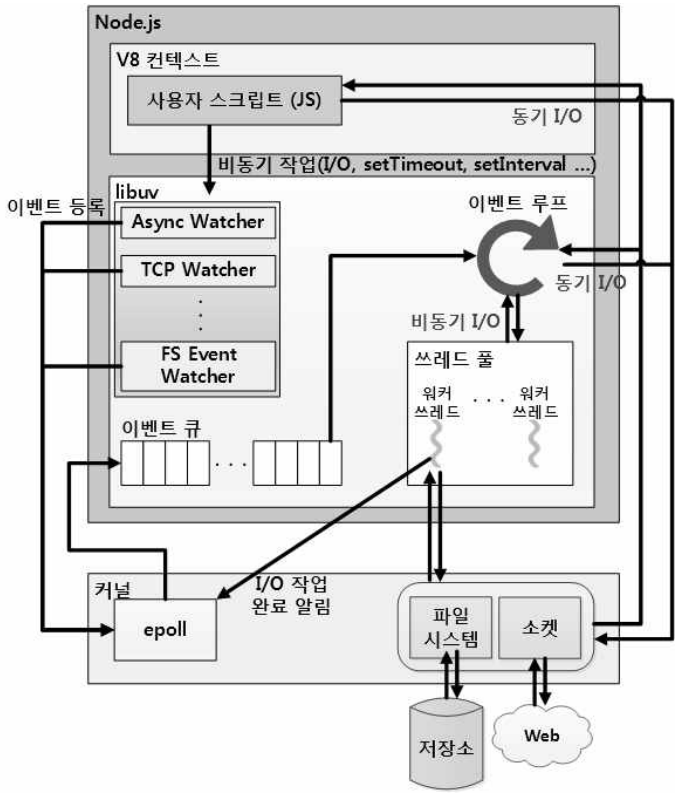

그림 1. Node.js의 구조

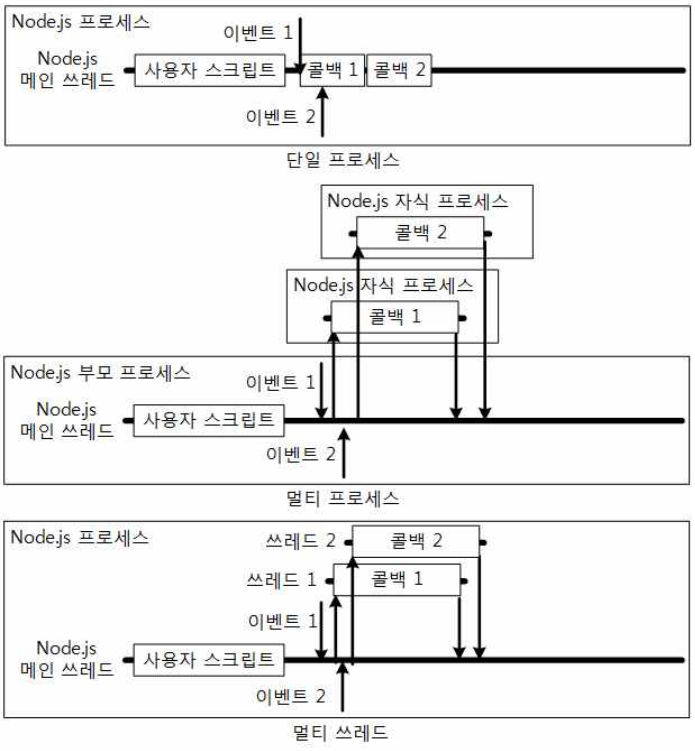

그림 2. Node.js의 단일 프로세스, 멀티 프로세스, 멀티 쓰레드 방식 비교

게 된다. 비동기 작업들은 이것에 해당하는 와쳐(watcher)를 통하여 커널(kernel)의 epoll에 이벤트를 등록하게 된다.

사용자의 코드가 모두 실행되면 libuv의 이벤트 루프(event loop)로 진입하게 된다. 이벤트 루프에서는 epoll이 넣어주는 이벤트 큐(event queue)에 들어오는 이벤트들에 해당하는 콜백 함수를 수행한다.

3. Node.js 작업 병렬화

3.1. Node.js 작업 병렬화의 문제점

Node.js와 같은 이벤트 드리븐 서버 프로그램에서 단일 프로세스(single process) 방식을 채택하면, 처리 시간이 긴 이벤트를 작업할 때 다음 이벤트를 작업할 때까지 지연이 크게 일어나게 된다. 처리 시간이 긴 작업은 I/O-bound 작업과 CPU-bound 작업으로 나뉠 수 있으며, Node.js에서는 I/O 쓰레드 풀과 클러스터 모듈을 사용하여 이 문제를 해결하도록 지원하고 있다.

I/O 쓰레드 풀은 프로세스를 블로킹(blocking)하여 처리 시간이 길어지는 I/O 동작을 별도의 쓰레드에서 병렬적으로 처리하는 Node.js의 기능이다. 클러스터 모듈은 그림 2의 멀티 프로세스 방식을 구현하는 Node.js의 기본 모듈로, 프로그래머가 직접 자식 프로세스를 포크(fork)하여 병렬적으로 CPU-bound 작업을 처리하도록 지원한다. 이들 모두 병렬화를 통해 I/O-bound 작업과 CPU-bound 작업에 의한 다른 이벤트의 지연을 최소화하고 있다.

그러나 클러스터 모듈은 CPU-bound 작업 지연의 완전한 해결책이 될 수 없다. 자식 프로세스를 생성하는 과정에서 Node.js 코드와 데이터를 초기화하여 CPU 오버헤드가 발생하고, 프로세스 사이에 중복되는 코드와 데이터가 메모리에 존재하여 메모리 오버헤드가 발생한다. 임베디드 환경에서는 열악한 CPU 처리 속도와 메모리 용량을 지니고 있으므로, 이러한 오버헤드 때문에 CPU-bound 병렬화를 사용하기 어렵다.

3.2. 해결책

이 중 메모리 중복 문제는 KSM을 이용하여 해결할 수 있다. KSM(Kernel Same-page Merging)은 실시간으로 중복되는 물리 메모리 페이지를 하나로 합쳐서 메모리 사용 효율을 높이는 기법이다.[3] Node.js에서 자바스크립트 코드와 데이터는 V8 엔진에 의해 스택과 힙 영역에 할당되게 되는데, 여러 개의 Node.js 프로세스가 생성되면 이 영역들이 중복되게 된다. 따라서 KSM으로 각 프로세스의 스택(stack)과 힙(heap) 영역을 포함한, 전체 익명 메모리(anonymous memory) 영역을 중복 제거 대상으로 삼아, 멀티 프로세스 환경에서 일어날 수 있는 메모리 중복을 최소화하고자 하였다.

KSM을 이용하는 멀티 프로세스 방법은 여전히 프로세스 초기화 CPU 오버헤드를 해결하지 못하는 한계가 있어, CPU 오버헤드까지 모두 해결할 수 있는 멀티 쓰레드 방법도 고려할 수 있다. 멀티 쓰레드는 새로운 쓰레드와 V8 컨텍스트(context)를 만들고, 해당 컨텍스트 상에서 CPU-bound 작업을 수행하는 방법이다. 이 방법에서는 프로세스 사이에 코드를 공유하므로 메모리 중복이 일부 제거되고, 초기화 CPU 오버헤드가 획기적으로 줄어든다. 그러나 V8의 컨텍스트 간 샌드박스(sandbox) 정책 때문에[4], 다른 컨텍스트에 존재하는 기존 모듈을 활용할 수 없다는 한계에 부딪쳐, 실용적으로 활용하기 어렵다는 단점이 있다.

4. 실험 및 분석

실험은 라즈베리 파이(Raspberry-Pi) B모델에서 수행하였다. 이 기기는 CPU로 ARM1176JZF-S 700MHz을, 512MB DRAM을 사용하였다. Node.js는 버전 0.10.16을 사용하였다.

먼저 멀티 프로세스와 멀티 쓰레드 방식의 CPU 오버헤드와 메모리 오버헤드를 알아보기 위해, 프로세스와 쓰레드의 생성 시간과 수행 시간, 메모리 사용량을 측정하여 비교·분석하였다. 또한 멀티 프로세스 구조에서 메모리 중복을 제거하기 위해 KSM을 적용한 효과를 측정하고 분석하였다.

4.1 프로세스/쓰레드 생성 시간

프로세스 생성 시간으로는 Node.js 프로세스 시작부터 이벤트 루프 진입 전까지의 수행시간을, 쓰레드 생성 시간으로는 쓰레드 생성 함수의 수행시간을 측정하였다.

이 실험에서 프로세스 생성 시간은 1076ms, 쓰레드 생성 시간은 4ms로 측정되었다. 이를 통해 멀티 프로세스 환경이 멀티 쓰레드 환경보다 269배 만큼 CPU 오버헤드를 많이 발생한다는 점을 알 수 있다.

4.2 메모리 사용량 및 수행 시간

스마트 TV에 가깝게 실험 환경을 구현하기 위하여 라즈베리 파이 TV(Raspberry-Pi TV) 프로젝트를 실험 목적에 맞게 수정하였다. 이 실험에서는 라즈베리 파이 TV 플랫폼 상에서 프로세스 또는 쓰레드가 게임, 웹 서버, 웹 소켓이라는 세 가지의 워크로드를 각각 수행하도록 하였다. 게임은 충돌 판별(collision detetion) 알고리즘을, 웹 서버는 스마트 TV의 UI를 보여주는 작업을, 웹 소켓은 스마트 TV가 사용자 클라이언트와의 통신 작업을 수행한다. 게임 워크로드는 CPU-bound 작업, 웹 서버와 웹 소켓은 I/O-bound 작업이다. 메인 프로세스/쓰레드에서는 웹 서버와 웹 소켓을, 자식 프로세스/쓰레드에서는 게임을 수행하였다.

수행시간은 웹 서버에 대한 클라이언트의 접근이 시작될 때부터 모든 프로세스/쓰레드의 충돌 연산이 한번 종료될 때까지의 시간을 측정하였다. PSS(Proportional Set Size)는 모든 Node.js 프로세스에 대해 합산하였다.

그 결과, 그림 3에서와 같이 멀티 프로세스보다 멀티 쓰레드 환경에서 메모리 사용량은 9.1~28.5%, 수행 시간은 10~12.5% 만큼 줄어들었다. 따라서 멀티 프로세스 환경보다 멀티 쓰레드 환경에서 CPU 오버헤드와 메모리 오버헤드가 더 적다.

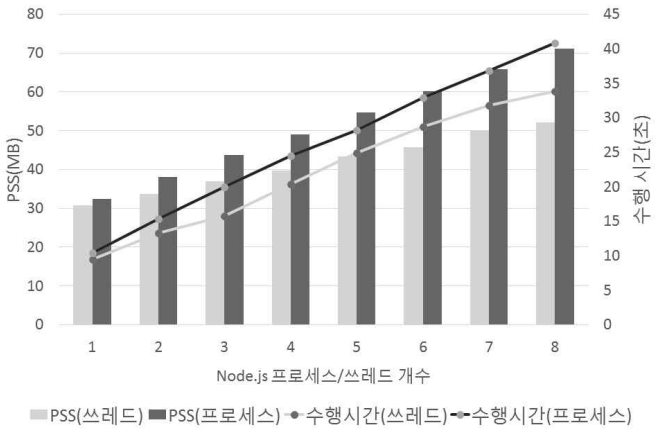

그림 3. 멀티 프로세스와 멀티 쓰레드 환경에서 메모리 사용량(PSS) 및 수행 시간 비교

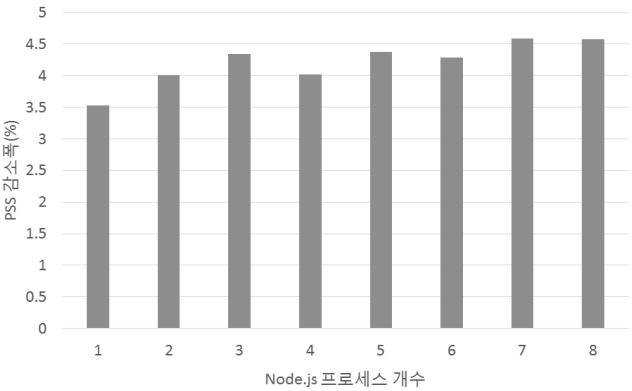

그림 4. KSM을 적용한 멀티 프로세스 방식의 메모리 사용량 감소 효과

4.3 메모리 중복 제거

4.2와 같은 실험 워크로드에서 KSM을 적용한 Node.js 환경과 적용하지 않은 환경에 대하여 실험을 진행하였다. 중복 제거 대상 메모리 영역은 각 Node.js 프로세스의 힙과 스택 등 익명 메모리 영역으로 정하였다.

그 결과, 그림 4에서와 같이 KSM에 의하여 약 3.5~4.5%의 메모리 사용량 감소폭이 나타났다. Node.js의 기존 멀티 프로세스 구조에서 KSM을 적용하면 메모리 오버헤드가 줄어드는 것을 알 수 있다.

5. 결론 및 향후 계획

우리는 이 연구를 통하여 임베디드 환경에서 Node.js의 CPU-bound 작업 병렬화의 한계를 지적하였고, 이를 해결하기 위한 대안으로 KSM을 적용한 멀티 프로세스 방식과 멀티 쓰레드 방식을 제시하였다.

실험을 통해 멀티 프로세스 방식보다 멀티 쓰레드 방식에서 CPU 오버헤드는 생성 시간이 296배, 실행 시간이 10~12.5% 만큼 향상되며, 메모리 오버헤드는 9.1~28.5% 만큼 향상됨을 입증하였다. 또한 멀티 프로세스 방식은 KSM을 적용하여 3.5~4.5%의 메모리 오버헤드 향상이 추가로 일어날 수 있음을 보였다.

멀티 쓰레드 방식이 두 가지 오버헤드에서 우위가 있음에도 불구하고, 샌드박싱으로 인해 현실적으로 사용하기 어렵다. 향후 연구에서는 Node.js 멀티 쓰레딩의 한계를 극복하여, 더 나은 병렬화를 통해 임베디드 서버 소프트웨어의 성능을 향상시킬 것이다.

참 고 문 헌

[1] TILKOV, Stefan; VINOSKI, Steve. Node. js: Using JavaScript to build high-performance network programs. Internet Computing, IEEE, 2010, 14.6: 80-83, 2010.

[2] https://developers.google.com/v8 , V8 JavaScript Engine

[3] ARCANGELI, Andrea; EIDUS, Izik; WRIGHT, Chris. Increasing memory density by using KSM. In: Proceedings of the linux symposium. p. 19-28. 2009.

[4] https://developers.google.com/v8/embed#security, Embedder's Guide