TraceMonkey 자바스크립트 엔진에서의 트레이스 오버헤드 감소 방안[1)]

유영호○ 이성원, 문수묵
서울대학교 전기·컴퓨터공학부 가상 머신 및 최적화 연구실
yoo0ho@altair.snu.ac.kr, swlee@altair.snu.ac.kr, smoon@snu.ac.kr

Reducing the Trace Overhead for TraceMonkey JavaScript Engine

Young-Ho Yoo○, Seong-Won Lee, Soo-Mook Moon
Seoul National University, Department of Electrical Engineering and Computer Science,
Virtual Machine and Optimization Lab

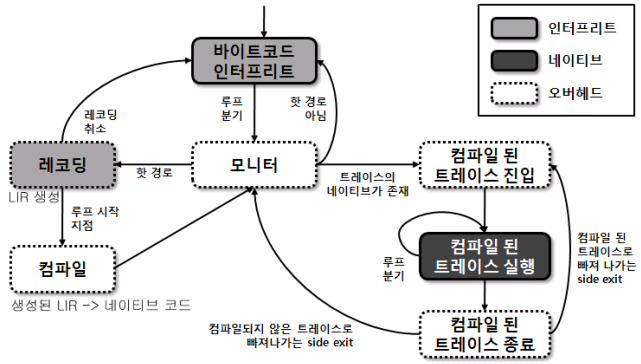
최근 IT 산업 전반에 걸쳐 모바일에 대한 중요도가 높아짐에 따라 인터넷 브라우저의 성능이 중요하게 되었다. 자바스크립트 언어의 수행은 인터넷 브라우저의 사용에 있어 상당히 비중이 높다. 이 논문에서는 자바스크립트 언어를 수행하는 엔진 중 하나인 TraceMonkey엔진이 트레이스를 하는 과정에서 생기는 오버헤드를 줄이는 최적화를 구현, 적용하고 이를 실험하여 평가한다.

1. 서론

최근 들어 스마트폰, 타블렛PC(Ipad, 갤럭시 탭) 등에 대한 사용이 늘고 관심이 증폭되면서 모바일 기기의 성능이 굉장히 중요해지고 있다. 이러한 모바일 환경에서 웹 브라우저의 성능이 상당한 비중을 차지하고 있다. 이에 필요한 자바스크립트 언어의 사용량이 급증하였다. [1] 모바일 환경에서는 PC에 비해 하드웨어가 제한적이기 때문에 웹 브라우저의 실행에서 상당한 비중을 차지하고 있는 자바스크립트 언어의 수행을 얼마나 빨리 처리 하는가가 관건이다. 최근 발표 되고 있는 대부분의 엔진들은 좋은 성능을 내기 위한 방법으로 JITC(Just In Time Compile)방식을 사용한다. [2] 이러한 JITC 방식을 사용하는 자바스크립트 엔진으로 TraceMonkey 엔진을 예로 들 수 있다. TraceMonkey 는 Mozilla 사의 Firefox 브라우저에 들어가는 오픈 소스로 자주 수행 되는 부분을 트레이스하여 네이티브 코드(머신코드)로 컴파일하여 수행하는 방식이다. [3] 이러한 트레이스(Trace)방식에는 자주 수행되는 부분을 구분하는 과정이나 문법적인 제한에 의해서 Trace가 취소되면서 오버헤드가 발생하게 된다.

2. 본론

TraceMonkey자바스크립트 엔진은 자바스크립트 코드를 바이트코드 형태로 변환한 뒤 인터프리터로 수행하는 SpiderMonkey 엔진에서 발전된 형태로서, TraceMonkey엔진의 수행 방식은 다음과 같다. 루프(loop,반복문)와 같이 자주 수행되는 핫경로(hotpath)를 만나면 이를 네이티브 코드로 컴파일 하여 수행하는 트레이스 기반(trace-based) 컴파일 방법을 사용한다. [3] 자주 반복되는 코드를 찾아 트레이스 단위로 컴파일하는 과정은 그림 1 과 같은 방법으로 진행된다. 루프는 후방 루프 분기(backward loop branch)를 포함하고 있기 때문에, 인터프리터로 바이트코드를 수행하는 도중 같은 후방 분기를 일정 횟수 이상 만나게 되면 모니터가 이를 핫경로로 인식하여 루프의 시작 지점부터 트레이스 자료구조를 생성하기 위한 레코딩 작업을 시작한다. 레코딩 작업은 그림 1에서 좌측 과정을 거친다. 컴파일 할 트레이스 내의 모든 바이트코드에 대해 LIR을 생성하고 다시 후방 분기를 만나게 되면 레코딩 작업을 완료하고 모아두었던 LIR을 네이티브 코드로 컴파일하고, 컴파일 작업이 끝나고 나면 그림 1의 우측으로 넘어가 네이티브 코드로 구성된 트레이스 자료구조가 생성되며, 루프의 남은 반복 횟수만큼 네이티브로 실행한다. 또한 생성된 네이티브 코드는 별도의 캐시에 저장되어 있다가 다음 번에 다시 해당 트레이스의 시작 부분으로 진입하게 되면 다시 컴파일 하지 않고 캐시(Cache) 에 저장된 코드를 수행하게 된다. [3] TraceMonkey의 트레이스 단위 컴파일 방식은 성능향상에 크게 도움이 되지만 트레이스 도중 루프 내부의 내용 중에 TraceMonkey 에서 정해놓은 여러 가지 제한 사항 때문에 레코딩 도중 취소가 되는 경우가 있어 이런 경우에는 레코딩을 하던 시간이 모두 오버헤드가 되어버리게 된다.



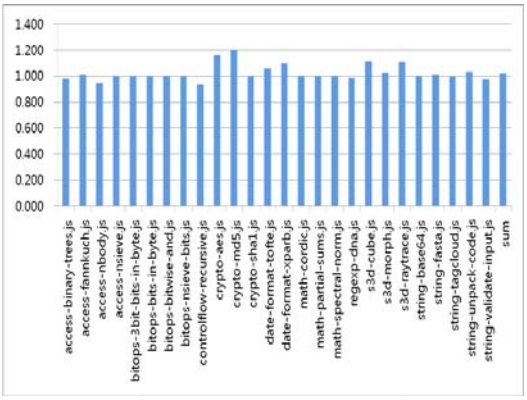
(그림 1 TraceMonkey 의 컴파일 과정)

이러한 트레이스에 따른 오버헤드를 줄이기 위해 다음과 같은 최적화를 적용하였다. 첫 번째는 EVAL 바이트코드에 따른 레코딩 취소 오버헤드를 줄이는 방법으로, TraceMonkey의 바이트코드 중에 EVAL 바이트코드는 자바스

1) 본 연구는 서울형산업 기술개발 지원사업(NT080546)의 지원으로 수행하였음

크립트의 eval함수가 불리었을 때 쓰이는 바이트코드로 eval 함수는 eval 함수에 쓰이는 문자형 수식을 계산하거나 문자형 숫자를 숫자(int,float)형 숫자로 바꿔주는 함수다. 어떤 루프가 핫경로라고 판단하여 레코딩 도중에 EVAL바이트코드를 만나면 이제까지 레코딩 했던 부분을 취소해버리고 핫경로가 아니라고 판단하여 트레이스를 포기한다. 이러한 오버헤드를 줄이기 위해 핫경로 판단을 하기 이전인 최초 인터프리터 수행 시에 미리 루프 내부에 EVAL 바이트코드가 있나 없나를 확인하여 EVAL 바이트코드가 있다면 아예 핫경로 가 아니라는 정보를 남기고 그 정보를 확인하여 레코딩 자체를 시도를 안 하게 하였다. 두 번째는 함수 호출에 의한 트레이스 오버헤드를 줄이기 위한 방법으로, 외부루프(outer loop) 과 내부루프(inner loop)가 있을 경우 외부루프의 후방분기에 의해 외부루프가 먼저 트레이스 되다가 내부루프의 후방분기를 만나면 내부루프를 트레이스 하게 되는데 외부 루프에 대해 레코딩 했던 부분이 취소되어 버린다. 이때 외부루프의 트레이스 과정에서 내부루프를 트레이스 하기 전에 함수를 호출한다고 가정하면 함수의 호출(call)과 리턴(return)과정에서 필요한 스택 프레임(stack frame) 이 생성되고 소멸하는 작업을 한다. 이 스택프레임이 생성되고 소멸 될 때 많은 일을 하게 된다. 이와 같은 과정까지를 레코딩를 했다가 취소하게 되면 그대로 오버헤드가 되어 수행시간이 그 만큼 증가하게 된다. 따라서 외부루프의 트레이스 중 함수 호출이 일정 값 이상일 경우에는 외부루프의 트레이스를 취소하지 않도록 하였다.

이에 따른 실험 결과는 다음과 같다. 실험 환경은 하드웨어는 x86 머신에서 실험하였으며, CPU는 Intel(R) Core(TM)2 Quad CPU Q8200@2.33GHz와 4G 메인 메모리의 머신에서 실험하였으며, TraceMonkey 1.9 버전에서 실험하였고, 벤치마크는 Sunspider를 사용하였다. 표 1 결과는 TraceMonkey원본과의 수행시간 비교로써 수행시간으로 표기했다. 결과를 보면 약 20% 성능향상이 있는 벤치마크도 있지만 종합적으로는 약 2%의 성능향상에 그친 것을 알 수 있다. 이 두 최적화가 Sunspider 벤치마크에서는 큰 효과를 보지 못한 이유에는 다음을 들 수 있다. eval 함수는 Mozilla 사를 비롯한 회사에서 자주 사용하지 말 것을 권고하고 있는데, 이는 모든 자바스크립트 엔진이 마찬가지로 겪고 있는 타입 체크의 문제로써 eval 함수를 사용하게 되면 타입을 확정 짓기가 어려워 JITC를 하는데 어려움이 따른다. [4] 그런데 이 eval 함수가 Sunspider 사용된 파일이 Sunspider 에서는 하나뿐이지만 실제 웹사이트에서는 이와 달리 eval 함수가 현저히 많이 사용되고 있기 때문에 이와 같은 최적화는 반드시 필요하다고 볼 수 있다.



(표 1 종합 최적화 결과)

사이트 이름	eval 함수 사용 횟수
네이버(Naver)	22
다음(Daum)	7
네이트(Nate)	9
구글코리아(Google)	11
야후코리아(Yahoo)	9

대한민국의 대표 검색사이트[5]를 검색해본 결과, 각 검색사이트의 메인 페이지에서 사용하고 있는 eval 함수의 개수는 다음 표 2와 같다. 실제 웹사이트에서의 성능은 측정하지는 못했지만 date-format-tofte.js 에서 루프 내에서 쓰이는 eval 로 인한 트레이스 시도가 세 번이었는데 이를 해결하여 4.5%가 좋아졌으므로 표 2 와 같이 eval 이 실제 사이트에서 자주 사용하므로 어느 정도 성능을 발휘할 것이라고 예상한다. 또, 표 1에서 많은 성능 향상을 보인 벤치마크 파일에는 외부루프 안에 함수 호출이 있는 경우 들이 있었는데 이것에 대한 오버헤드가 줄었다고 볼 수 있다. 반대로 외부루프 안에 함수 호출이 없는 경우의 벤치마크 파일에서는 오히려 이러한 함수 호출에 대한 검사코드가 오버헤드가 되어 성능이 느려짐을 알 수 있다. 결과적으로 앞에서 살펴본 바와 같이 두 최적화를 전부 적용하면 Sunspider 벤치마크 기준으로 약 2%의 성능향상(기하평균)을 이루었다고 볼 수 있다.

3. 결론 및 요약

TraceMonkey 는 트레이스를 기반으로 하는 자바스크립트 엔진이다. 루프 단위로 트레이스를 하는 TraceMonkey는 때로는 트레이스가 취소가 되는 경우가 있는데, 이는 고스란히 순수한 오버헤드가 되어 성능 저하에 원인이 된다. 이러한 오버헤드를 없애기 위해서 eval 함수가 루프 내에 있을 경우에는 아예 트레이스 자체를 시작을 안 하는 최적화와 외부루프에 서 함수호출이 있을 경우에는 내부루프에 대한 함수호출에 대한 오버헤드를 최소화 하기 위한 최적화를 적용하여 Sunspider 벤치마크에서 약 2%의 성능 향상을 이루었다. 이 두 최적화는 실제 웹사이트의 자바스크립트 언어 성향에 비추어 볼 때 더욱 효과적일 수 있다고 볼 수 있다.

추후 연구 과제로 실제 웹사이트에서의 eval 함수의 사용 현황과 연계하여 직접적인 결과로 실제 웹사이트의 수행시간을 도출해내면 더 좋은 연구 결과를 낼 수 있을 것으로 생각된다.

[참고 문헌]

[1] R. Chugh, J. Meister, R. Jhala and S. Lerner, Staged Information Flow of JavaScript, In Proceedings of PLDI, 2009.
[2] ECMA International. ECMAScript Language Specification. Standard, ECMA-262, 1999.
[3] A. Gal, B. Eich, M. Shaver, D. Anderson, D. Mandelin, M. R. Haghighat, B. Kaplan, G. Hoare, B. Zbarsky, J. Orendorff, J. Ruderman, E. Smith, R. Reitmaier, M. Bebenita, M. Chang, and M. Frantz, Trace-based Just-In-Time Type Specialization for Dynamic Languages, In Proceedings of PLDI, 2009.
[4] Paruj Ratanaworabhan, Benjamin Livshits, David Simmons, Benjamin Zorn, JSMeter: Characterizing Real-World Behavior of JavaScript Programs , Microsoft Research Technical Report , 2009
[5] ttp://trend.logger.co.kr, 대표사이트 분석리포트 2010.01.01~2010.09.01 까지의 순위