TraceMonkey 자바스크립트 엔진의 메소드 단위 컴파일 적용

오진석° 정원기, 오형석, 문수묵
서울대학교 전기 컴퓨터 공학부
jjingoh@altair.snu.ac.kr, dream9903@altair.snu.ac.kr, oracle@altair.snu.ac.kr, smoon@snu.ac.kr

Method-based Trace for TraceMonkey JavaScript Engine

JinSeok Oh° Won-Ki Jung, Hyeong-Seok Oh, Soo-Mook Moon
School of Electrical Engineering and Computer Sciences, Seoul National University

1. 서론

자바스크립트는 웹상에서 클라이언트 사이드의 프로그래밍에 사용되는 오브젝트 기반의 언어이다. 기존의 웹 브라우저에서 사용되던 자바스크립트엔진은 인터프리터를 통해 수행되어 낮은 성능을 내고 있었지만, 최근에 성능을 개선한 자바스크립트 엔진이 등장 하였다.

Mozilla의 TracceMonkey는 수행 중에 바이트 코드를 머신 코드로 번역하여 수행하는 Just-In Time 컴파일러를 사용하고 있다. 일반적인 프로그램에서 대부분의 시간을 사용하는 핫 스팟 경로들(주로 루프)을 기반으로 한 동적 컴파일을 사용한다. 이러한 핫 스팟 경로를 Trace라 부르는데 이는 반복적으로 수행되는 일련의 명령어들을 말하는 것으로 주로 메소드 내에 존재하는 핫한 루프에서 반복적으로 수행되기 때문에 루프를 단위로 컴파일하게 된다.[1] 이러한 TraceMonkey의 방식은 핫 루프의 경우 매우 빠른 성능을 낼 수 있었다. 하지만 최근 웹에서 사용되는 자바스크립트의 특징을 분석하는 연구가 진행되면서 실제 웹은 복잡한 계산을 하는 경우가 적고 대부분의 메소드가 루프를 포함하지 않아서 핫 루프가 수행시간에서 차지하는 비중이 적은 것으로 나타났다. [2]

본 논문에서는 실제 웹 특성을 반영하기 위하여 Trace의 단위에 핫 메소드를 포함 하고자 한다. Trace의 컴파일 오버헤드와 핫 메소드를 컴파일 해서 얻은 이득을 벤치마크를 통해 비교하여 핫 메소드를 포함 하는 것이 적합한지, 이를 개선할 수 있을지를 평가하고자 한다.

2. 본론

TraceMonkey는 자바스크립트 코드를 중간 언어의 형태인 바이트 코드로 번역된 뒤, 인터프리터로 수행 하는 것을 기본으로 한다. 인터프리터의 수행 중 루프와 같이 자주 수행되는 핫패스를 만나면 레코딩을 수행하게 된다. 핫 루프를 인식하는 방법은 루프에 존재하는 후방분기명령어를 가지는 바이트 코드를 만날 때 Monitor를 통해 분석하는 것이다. 레코딩 과정에서 핫 루프에 해당하는 바이트코드와 바이트 코드에서 사용되는 변수들의 타입 정보를 기록하고, 이를 통해 컴파일을 수행한다. 컴파일을 통해 머신 코드를 만들어 내면 이를 수행하고 루프가 끝나면 다시 인터프리터의 수행으로 돌아간다.

TraceMonkey의 루프에는 OP_LOOP라는 바이트 코드가 헤더에 항상 존재한다. 루프의 마지막에 존재하는 후방 분기명령의 타깃은 항상 바이트 코드의 헤더인 OP_LOOP 바이트코드이다. 그렇기 때문에 Trace의 시작은 항상 OP_LOOP 바이트 코드부터 시작되게 구성되어 있다. 그러므로 메소드 컴파일을 적용하기 위해서도 마찬가지로 모든 함수의 시작에 OP_LOOP 바이트코드가 생성되도록 구성했다. TraceMonkey에서 메소드의 시작을 의미하는 바이트 코드는 따로 없고 OP_LOOP 바이트 코드 또한 헤더의 의미만 가지고 수행은 이루어지지 않는 바이트 코드이기 때문에 OP_LOOP 바이트 코드가 문제없이 추가될 수 있었다.

본 연구는 서울형산업 기술개발 지원사업(NT080546)의 지원으로 수행되었음.

메소드가 컴파일 되기 위해서는 메소드가 호출될 때 이를 모니터를 통해 핫메소드인지 확인하고 핫메소드이면 레코딩을 통해 컴파일 될 수 있게 해야 한다. 메소드를 호출하는 JSOP_CALL 바이트 코드에서 메소드가 호출되기 전에 모니터를 활성화해서 이를 통해 핫메소드인지를 확인할 수 있도록 했다.

기존에 루프에서의 Trace의 경우 레코딩 종료 시점은 루프를 인식하게 한 후방분기명령을 만났을 경우 이다. 하지만 메소드의 경우 루프와 달리 반복 되는 부분이 아니므로 메소드의 종료를 나타내는 OP_RET바이트 코드를 만났을 때 종료 되도록 했다. 즉 메소드 레코딩의 경우 메소드의 시작부터 메소드의 리턴 부분까지 이루어진다.

TraceMonkey의 모든 루프가 Trace가 가능한 것은 아니다. Trace내에 재귀함수나 지원하지 않는 라이브러리 함수를 포함하고 있는 경우에는 Trace가 불가능하다. TraceMonkey에서는 이럴 경우 블랙리스트에 추가하여 Trace가 계속 시도 되는 것을 방지한다. 마찬가지로 메소드의 경우에도 메소드가 Trace가 불가능할 경우 블랙리스트에 추가하여 더 이상 Trace를 시도하는 것을 방지하도록 했다.

TraceMonkey의 Trace특징 중 하나는 루프 안에 존재하는 메소드가 인라인의 효과를 가진다는 점이다. 레코딩 과정에서 메소드 호출이 존재할 때 호출된 메소드가 함께 레코딩 되기 때문이다. 메소드의 호출도 결국 바이트 코드의 이동이므로 다른 바이트 코드처럼 레코딩되어 머신코드로 컴파일 된다. 그렇기 때문에 Trace단위의 메소드 컴파일을 적용할 경우 루프Trace를 레코딩 할 때 이미 컴파일된 메소드를 사용할 경우 이러한 인라인된 효과를 받을 수 없다. 그렇기 때문에 레코딩 중에는 메소드 컴파일이 이루어 지지 않도록 했다.

실험은 일반적인 JavaScript 성능을 테스트 하는데 자주 사용되는 Sunspider 벤치마크로 하였다. 실험결과 성능이 나아지지 않은 것을 알 수 있었다. 가장 큰 이유는 벤치마크로 사용한 Sunspider가 루프에 상당히 많은 시간을 소모하기 때문에 실제로 성능에 영향을 주는 핫패스가 모두 컴파일 되었기 때문이다. 일부 메소드가 컴파일 되기는 하지만 컴파일 된 메소드가 실제로 호출 되는 수가 적어서 오히려 컴파일 오버헤드로 인해 성능이 더 늦어진 것으로 보인다.

3. 결론

TraceMonkey는 핫루프를 Trace로 하여 이를 컴파일 하여 좋은 성능을 내고 있다. 하지만 실제 자바스크립트가 주로 사용되는 웹에서는 루프가 기존의 프로그램에 비해 많이 사용되지 않고 결과적으로 루프가 수행 시간에 덜 영향을 준다는 것이 밝혀졌다.

이를 통해 Trace의 단위로 루프뿐만 아니라 메소드를 적용하는 것을 해보았다. 하지만 벤치마크의 특성과 Trace의 시작과 끝의 오버헤드로 인해 오히려 성능이 나빠지는 것을 발견 했다.

메소드의 경우에는 Trace방식 보다는 더 간단한 방식의 Just-in-Time컴파일러를 적용하여 Trace의 오버헤드와 컴파일 오버헤드를 적용하는 것이 더 좋을 것으로 보인다.

실제 웹에서의 자바스크립트의 수행은 기존의 프로그램과 다른 동작을 보이는 것이 나타나고 있다. 이를 통해 기존의 최적화 방식이 아닌 새로운 방식의 최적화 방식을 적용하는 것이 필요할 것이다.

Reference

[1] Gal, A., Eich, B., Shaver, M., Anderson, D., Mandelin, D., Haghighat, M.R., Ka-plan, B., Hoare, G., Zbarsky, B., Orendorff, J., Ruderman, J., Smith, E., Reitmaier, R., Bebenita, M., Chang, M., Franz, M.: Trace-based just-in-time type specializa-tion for dynamic languages. In: Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation (PLDI'09), Dublin, Ireland(June 2009) 465-478

[2] P. Ratanaworabhan, B. Livshits, D. Simmons, and B. Zorn. JSMeter: Characterizing real-world behavior of JavaScript programs. Technical Report MSR-TR-2009-173, Microsoft Research, Dec. 2009.