(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2017-0125398
(43) 공개일자 2017년11월14일

(51) 국제특허분류(Int. Cl.)
G06F 9/30 (2017.01) *G06F 17/30* (2006.01)
G06F 9/45 (2006.01) *G06F 9/455* (2006.01)
H04L 29/08 (2006.01)
(52) CPC특허분류
G06F 9/3017 (2013.01)
G06F 17/30902 (2013.01)
(21) 출원번호 10-2017-7028178
(22) 출원일자(국제) 2016년03월29일
심사청구일자 2017년10월30일
(85) 번역문제출일자 2017년09월29일
(86) 국제출원번호 PCT/US2016/024791
(87) 국제공개번호 WO 2016/195790
국제공개일자 2016년12월08일
(30) 우선권주장
14/726,376 2015년05월29일 미국(US)

(71) 출원인
구글 엘엘씨
미국 캘리포니아 마운틴 뷰 엠피시어터 파크웨이 1600 (우:94043)
(72) 발명자
구오, 양
미국 캘리포니아 마운틴 뷰 엠피시어터 파크웨이 1600 (우:94043) 구글 인코포레이티드 (내)
보겔하임, 다니엘
미국 캘리포니아 마운틴 뷰 엠피시어터 파크웨이 1600 (우:94043) 구글 인코포레이티드 (내)
아이징어, 조첸 마티아스
미국 캘리포니아 마운틴 뷰 엠피시어터 파크웨이 1600 (우:94043) 구글 인코포레이티드 (내)
(74) 대리인
특허법인 남앤드남

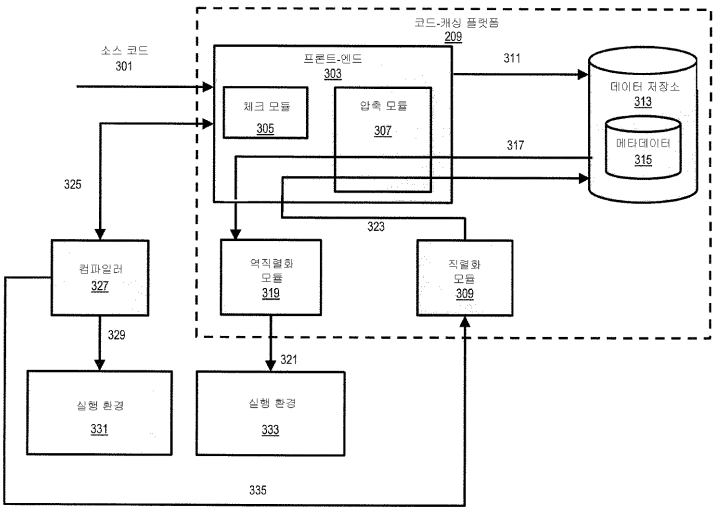
전체 청구항 수 : 총 21 항

(54) 발명의 명칭 **코드 캐싱 시스템**

(57) 요 약

코드 캐싱을 위한 시스템들 및 방법들이 제공된다. 실행을 대기하는 1차 소스 코드의 제 1 표시가 수신된다. 1차 소스 코드에 대응하는 캐싱된 데이터에 대해 리소스 캐시가 체크된다. 리소스 캐시에서의 캐시 미스 시, 1차 소스 코드로부터 컴파일된 제 1 실행가능 코드가 획득된다. 1차 소스 코드에서 참조되는 2차 소스 코드가 선택된다. 선택된 2차 소스 코드로부터 컴파일된 제 2 실행가능 코드가 획득된다. 제 1 실행가능 코드 및 제 2 실행가능 코드는 직렬화된 코드로 직렬화된다. 직렬화된 코드는 캐싱된 데이터로서 리소스 캐시에 저장된다.

대 표 도



(52) CPC특허분류
G06F 8/4434 (2013.01)
G06F 8/4441 (2013.01)
G06F 9/45529 (2013.01)
H04L 67/2842 (2013.01)

명 세 서

청구범위

청구항 1

코드 캐싱(code caching) 시스템으로서,

하나 또는 그 초과의 프로세서들; 및

명령들이 저장된 머신-판독가능 매체를 포함하며,

상기 명령들은, 상기 하나 또는 그 초과의 프로세서들에 의해 실행되는 경우, 상기 하나 또는 그 초과의 프로세서들로 하여금 동작들을 수행하게 하고,

상기 동작들은,

실행을 대기하는 1차 소스 코드의 제 1 표시를 수신하는 것;

상기 제 1 표시를 수신하는 것에 대한 응답으로, 상기 1차 소스 코드에 대응하는 캐싱된 데이터에 대해 리소스 캐시를 체크(check)하는 것; 및

상기 리소스 캐시에서의 캐시 미스(cache miss) 시:

상기 1차 소스 코드로부터 컴파일(compile)된 제 1 실행가능 코드를 획득하는 것,

상기 1차 소스 코드에서 참조되는 2차 소스 코드를 선택하는 것,

선택된 2차 소스 코드로부터 컴파일된 제 2 실행가능 코드를 획득하는 것,

상기 제 1 실행가능 코드 및 상기 제 2 실행가능 코드를 직렬화(serialize)된 코드로 직렬화하는 것. 및

상기 직렬화된 코드를 캐싱된 데이터로서 상기 리소스 캐시에 저장하는 것

을 포함하는, 코드 캐싱 시스템.

청구항 2

제 1 항에 있어서,

상기 명령들은 추가로, 상기 하나 또는 그 초과의 프로세서들로 하여금, 제 1 실행 컨텍스트에서의 상기 제 1 실행가능 코드의 실행으로부터의 실행 결과들을 획득하는 것을 포함하는 동작들을 수행하게 하며,

상기 2차 소스 코드는 상기 실행 결과들에 기초하여 선택되는, 코드 캐싱 시스템.

청구항 3

제 1 항에 있어서,

상기 2차 소스 코드는, 상기 2차 소스 코드의 사이즈, 상기 제 2 실행가능 코드의 사이즈, 상기 2차 소스 코드가 상기 1차 소스 코드에서 참조된다는 예측, 상기 2차 소스 코드가 상기 1차 소스 코드에서 참조되는 횟수, 상기 1차 소스 코드에서 참조되는 상기 2차 소스 코드의 빈도, 또는 상기 2차 소스 코드의 컴파일 시간 중 하나 또는 그 초과에 기초하여 선택되는, 코드 캐싱 시스템.

청구항 4

제 1 항에 있어서,

상기 명령들은 추가로, 상기 하나 또는 그 초과의 프로세서들로 하여금,

제 2 실행 컨텍스트에서의 실행을 대기하는 상기 1차 소스 코드의 제 2 표시를 수신하는 것 – 상기 제 2 표시는 상기 제 1 표시에 후속하여 수신됨 –;

상기 제 2 표시를 수신하는 것에 대한 응답으로, 상기 1차 소스 코드 및 상기 선택된 2차 소스 코드에 대응하는

캐싱된 데이터에 대해 상기 리소스 캐시를 체크하는 것; 및

상기 리소스 캐시에서의 캐시 히트(cache hit) 시:

상기 직렬화된 코드를 포함하는 캐싱된 데이터를 상기 리소스 캐시로부터 리트리브(retrieve)하는 것,

리트리브된 직렬화된 코드를 제 3 실행가능 코드로 역직렬화(deserialize)하는 것, 및

상기 제 2 실행 컨텍스트에서의 실행을 위해 상기 제 3 실행가능 코드를 제공하는 것

을 포함하는 동작들을 수행하게 하는, 코드 캐싱 시스템.

청구항 5

제 4 항에 있어서,

제 1 실행 컨텍스트 및 상기 제 2 실행 컨텍스트는 코드 실행을 위한 2개의 상이한 가상 머신 환경들인, 코드 캐싱 시스템.

청구항 6

제 5 항에 있어서,

상기 1차 소스 코드는 웹 페이지 내의 스크립트(script)이고, 상기 제 1 실행 컨텍스트 및 상기 제 2 실행 컨텍스트는, 상기 스크립트가 실행되는 웹 브라우저 세션들인, 코드 캐싱 시스템.

청구항 7

제 1 항에 있어서,

상기 1차 소스 코드는 웹 페이지의 최상위-레벨(top-level) 스크립트인, 코드 캐싱 시스템.

청구항 8

제 1 항에 있어서,

상기 제 1 실행가능 코드는, 상기 제 1 실행가능 코드에 임베딩된 메모리 어드레스들을 포함하는 레퍼런스(reference)들의 세트를 포함하며,

상기 제 1 실행가능 코드를 직렬화하는 것은 상기 임베딩된 메모리 어드레스들을 추상(abstract) 어드레스들로 대체하는 것을 포함하는, 코드 캐싱 시스템.

청구항 9

코드 캐싱 시스템으로서,

하나 또는 그 초과의 프로세서들; 및

명령들이 저장된 머신-판독가능 매체를 포함하며,

상기 명령들은, 상기 하나 또는 그 초과의 프로세서들에 의해 실행되는 경우, 상기 하나 또는 그 초과의 프로세서들로 하여금 동작들을 수행하게 하고,

상기 동작들은,

실행을 대기하는 1차 소스 코드의 제 1 표시를 수신하는 것;

상기 제 1 표시를 수신하는 것에 대한 응답으로, 상기 1차 소스 코드에 대응하는 캐싱된 데이터에 대해 리소스 캐시를 체크하는 것; 및

상기 리소스 캐시에서의 캐시 미스 시:

상기 1차 소스 코드로부터 컴파일된 제 1 실행가능 코드를 획득하는 것,

상기 1차 소스 코드의 사이즈, 미리결정된 시간 기간 내에 상기 1차 소스 코드가 실행되는 횟수 또는 빈도, 또는 상기 1차 소스 코드의 컴파일 시간 중 하나 또는 그 초과에 기초하여, 상기 제 1 실행가능 코드를 직렬화된

코드로 직렬화하는 것, 및

상기 직렬화된 코드를 캐싱된 데이터로서 상기 리소스 캐시에 저장하는 것

을 포함하는, 코드 캐싱 시스템.

청구항 10

제 9 항에 있어서,

상기 명령들은 추가로, 상기 하나 또는 그 초과의 프로세서들로 하여금,

상기 1차 소스 코드에서 참조되는 2차 소스 코드를 선택하는 것 - 상기 2차 소스 코드를 선택하는 것은, 적어도, 상기 2차 소스 코드의 사이즈, 제 2 실행가능 코드의 사이즈, 상기 2차 소스 코드가 상기 1차 소스 코드에서 참조되는 횟수, 상기 1차 소스 코드에서 참조되는 상기 2차 소스 코드의 빈도, 또는 상기 2차 소스 코드의 컴파일 시간, 또는 이들의 결합에 기초함 -;

상기 2차 소스 코드로부터 컴파일된 제 2 실행가능 코드를 획득하는 것; 및

상기 제 2 실행가능 코드를 상기 직렬화된 코드로 직렬화하는 것

을 포함하는 동작들을 수행하게 하는, 코드 캐싱 시스템.

청구항 11

제 9 항에 있어서,

상기 명령들은 추가로, 상기 하나 또는 그 초과의 프로세서들로 하여금,

제 1 실행 컨텍스트에서의 상기 제 1 실행가능 코드의 실행으로부터의 실행 결과들을 획득하는 것을 포함하는 동작들을 수행하게 하며,

2차 소스 코드는 상기 실행 결과들에 기초하여 선택되는, 코드 캐싱 시스템.

청구항 12

제 10 항에 있어서,

상기 명령들은 추가로, 상기 하나 또는 그 초과의 프로세서들로 하여금,

제 2 실행 컨텍스트에서의 실행을 대기하는 상기 1차 소스 코드의 제 2 표시를 수신하는 것 - 상기 제 2 표시는 상기 제 1 표시에 후속하여 수신됨 -;

상기 제 2 표시를 수신하는 것에 대한 응답으로, 상기 1차 소스 코드 및 선택된 2차 소스 코드에 대응하는 캐싱된 데이터에 대해 상기 리소스 캐시를 체크하는 것; 및

상기 리소스 캐시에서의 캐시 히트 시:

상기 직렬화된 코드를 포함하는 캐싱된 데이터를 상기 리소스 캐시로부터 리트리브하는 것,

리트리브된 직렬화된 코드를 제 3 실행가능 코드로 역직렬화하는 것, 및

상기 제 2 실행 컨텍스트에서의 실행을 위해 상기 제 3 실행가능 코드를 제공하는 것

을 포함하는 동작들을 수행하게 하는, 코드 캐싱 시스템.

청구항 13

제 12 항에 있어서,

제 1 실행 컨텍스트 및 상기 제 2 실행 컨텍스트는 코드 실행을 위한 2개의 상이한 가상 머신 환경들인, 코드 캐싱 시스템.

청구항 14

제 9 항에 있어서,

상기 제 1 실행가능 코드는, 상기 제 1 실행가능 코드에 임베딩된 메모리 어드레스들을 포함하는 레퍼런스들의 세트를 포함하며,

상기 제 1 실행가능 코드를 직렬화하는 것은 상기 임베딩된 메모리 어드레스들을 추상 어드레스들로 대체하는 것을 포함하는, 코드 캐싱 시스템.

청구항 15

명령들을 포함하는 컴퓨터-판독가능 매체로서,

상기 명령들은, 컴퓨터에 의해 실행되는 경우, 상기 컴퓨터로 하여금,

실행을 대기하는 1차 소스 코드의 제 1 표시를 수신하게 하고;

상기 제 1 표시를 수신하는 것에 대한 응답으로, 상기 1차 소스 코드에 대응하는 캐싱된 데이터에 대해 리소스 캐시를 체크하게 하고; 그리고

상기 리소스 캐시에서의 캐시 미스 시:

상기 1차 소스 코드로부터 컴파일된 제 1 실행가능 코드를 획득하게 하고,

제 1 실행 컨텍스트에서의 상기 제 1 실행가능 코드의 실행으로부터 실행 결과들을 획득하게 하고,

상기 실행 결과들에 기초하여, 상기 1차 소스 코드에서 참조되는 2차 소스 코드를 선택하게 하고,

선택된 2차 소스 코드로부터 컴파일된 제 2 실행가능 코드를 획득하게 하고,

상기 제 1 실행가능 코드 및 상기 제 2 실행가능 코드를 직렬화된 코드로 직렬화하게 하고, 그리고

상기 직렬화된 코드를 캐싱된 데이터로서 상기 리소스 캐시에 저장하게 하는,

컴퓨터-판독가능 매체.

청구항 16

제 15 항에 있어서,

상기 제 1 실행가능 코드는, 상기 제 1 실행가능 코드에 임베딩된 메모리 어드레스들을 포함하는 레퍼런스들의 세트를 포함하며,

상기 제 1 실행가능 코드를 직렬화하는 것은 상기 임베딩된 메모리 어드레스들을 추상 어드레스들로 대체하는 것을 포함하는, 컴퓨터-판독가능 매체.

청구항 17

제 15 항에 있어서,

상기 실행 결과들은, 상기 1차 소스 코드에서 참조되는 2차 소스 코드가 상기 1차 소스 코드의 실행에서 실행된 횟수를 표시하는, 컴퓨터-판독가능 매체.

청구항 18

제 15 항에 있어서,

상기 명령들은 추가로, 상기 컴퓨터로 하여금,

제 2 실행 컨텍스트에서의 실행을 대기하는 상기 1차 소스 코드의 제 2 표시를 수신하게 하고 – 상기 제 2 표시는 상기 제 1 표시에 후속하여 수신됨 –;

상기 제 2 표시를 수신하는 것에 대한 응답으로, 상기 1차 소스 코드 및 상기 선택된 2차 소스 코드에 대응하는 캐싱된 데이터에 대해 상기 리소스 캐시를 체크하게 하고; 그리고

상기 리소스 캐시에서의 캐시 히트 시:

상기 직렬화된 코드를 포함하는 캐싱된 데이터를 상기 리소스 캐시로부터 리트리브하게 하고,

리트리브된 직렬화된 코드를 제 3 실행가능 코드로 역직렬화하게 하고, 그리고

상기 제 2 실행 컨텍스트에서의 실행을 위해 상기 제 3 실행가능 코드를 제공하게 하는,

컴퓨터-판독가능 매체.

청구항 19

제 18 항에 있어서,

상기 제 1 실행 컨텍스트 및 상기 제 2 실행 컨텍스트는 코드 실행을 위한 2개의 상이한 가상 머신 환경들인, 컴퓨터-판독가능 매체.

청구항 20

제 18 항에 있어서,

상기 명령들은 추가로, 상기 컴퓨터로 하여금,

상기 제 1 실행가능 코드와 연관된 제 1 실행 프로파일 및 상기 제 2 실행가능 코드와 연관된 제 2 실행 프로파일을 획득하게 하고;

상기 제 1 실행 프로파일 및 상기 제 2 실행 프로파일을 직렬화하게 하고; 그리고

직렬화된 제 1 실행 프로파일 및 제 2 실행 프로파일을 상기 리소스 캐시에 저장하게 하며,

상기 리트리브된 직렬화된 코드를 역직렬화하는 것은 상기 제 1 실행 프로파일 및 상기 제 2 실행 프로파일을 역직렬화하는 것을 포함하는, 시스템.

청구항 21

제 15 항에 있어서,

상기 명령들은 추가로, 상기 컴퓨터로 하여금,

상기 1차 소스 코드를 직렬화하기 위한 실행 컨텍스트의 가용성을 결정하게 하고; 그리고

상기 결정이 상기 실행 컨텍스트의 불가용성(unavailability)을 표시하는 경우, 상기 직렬화하는 것을 지연시키게 하는,

시스템.

발명의 설명

기 술 분 야

[0001] [0001] 본 기술은 실행가능 코드를 캐싱(caching)하는 것에 관한 것으로, 특히, 컴파일(compiling)을 위해 소스 코드(source code)를 선택하고, 실행가능 코드를 직렬화(serializing) 및 캐싱하고, 그리고 실행에 앞서, 캐싱된 실행가능 코드를 역직렬화(deserializing)하는 것에 관한 것이다.

배 경 기 술

[0002] [0002] 가상 머신(VM; Virtual Machine)들은, JIT 컴파일(Just in Time compilation)로도 또한 지칭되는 온-디멘드(on-demand) 컴파일 시에 소프트웨어 코드의 실행을 위해 사용되며, 여기서, 코드의 컴파일은 코드의 실행 직전에 수행되거나, 코드의 일부는 온-디멘드로(예컨대, 소극적으로(lazily)) 컴파일된다. 코드가 리소스(예컨대, 웹 페이지 내의 리소스)에 임베딩(embed)된 경우, 클라이언트 디바이스 상에서 실행되는 애플리케이션(예컨대, 브라우저) 내에서 리소스가 렌더링(render)될 수 있는 속도는 임베딩된 코드의 실행 속도에 의존한다. 클라이언트 디바이스의 사용자는 동일한 작업 세션(예컨대 웹 브라우저 세션) 내에서 또는 다수의 독립적인 작업 세션들에 걸쳐 동일한 리소스에 반복적으로 액세스할 수 있다. 그러나, 사용자가 리소스에 액세스할 때마다 코드가 컴파일되어야 한다면 매 컴파일마다 상당히 긴 시간을 필요로 할 수 있다.

발명의 내용

[0003] [0003] 다양한 양상들에서, 개시된 발명의 대상은 시스템으로 구현될 수 있다. 시스템은 하나 또는 그 초과의 프로세서들을 포함한다. 시스템은 또한 메모리를 포함한다. 메모리는, 실행되는 경우 하나 또는 그 초과의 프로세서들로 하여금 방법을 구현하게 하는 명령들을 포함한다. 명령들은, 실행을 대기하는 1차 소스 코드의 제 1 표시를 수신하기 위한 코드를 포함한다. 명령들은, 제 1 표시를 수신하는 것에 대한 응답으로, 1차 소스 코드에 대응하는 캐싱된 데이터에 대해 리소스 캐시를 체크하기 위한 코드를 포함한다. 명령들은, 리소스 캐시에서의 캐시 미스(cache miss) 시, 1차 소스 코드로부터 컴파일된 제 1 실행가능 코드를 획득하기 위한 코드를 포함한다. 명령들은, 1차 소스 코드에서 참조되는 2차 소스 코드를 선택하기 위한 코드를 포함한다. 명령들은, 선택된 2차 소스 코드로부터 컴파일된 제 2 실행가능 코드를 획득하기 위한 코드를 포함한다. 명령들은, 제 1 실행가능 코드 및 제 2 실행가능 코드를 직렬화된 코드로 직렬화하기 위한 코드를 포함한다. 명령들은, 직렬화된 코드를 캐싱된 데이터로서 리소스 캐시에 저장하기 위한 코드를 포함한다.

[0004] [0004] 이들 및 다른 구현들은 다음의 특성들 중 하나 또는 그 초과를 포함할 수 있다. 명령들은, 제 1 실행 컨텍스트(context)에서의 제 1 실행가능 코드의 실행으로부터의 실행 결과들을 획득하기 위한 코드를 포함할 수 있으며, 여기서, 2차 소스 코드는 실행 결과들에 기초하여 선택된다. 2차 소스 코드는, 2차 소스 코드의 사이즈, 제 2 실행가능 코드의 사이즈, 2차 소스 코드가 1차 소스 코드에서 참조된다는 예측, 2차 소스 코드가 1차 소스 코드에서 참조되는 횟수, 1차 소스 코드에서 참조되는 2차 소스 코드의 빈도, 또는 2차 소스 코드의 컴파일 시간 중 하나 또는 그 초과에 기초하여 선택될 수 있다. 명령들은, 제 2 실행 컨텍스트에서의 실행을 대기하는 1차 소스 코드의 제 2 표시를 수신하기 위한 코드를 포함할 수 있으며, 여기서, 제 2 표시는 제 1 표시에 후속하여 수신된다. 명령들은, 제 2 표시를 수신하는 것에 대한 응답으로, 1차 소스 코드 및 선택된 2차 소스 코드에 대응하는 캐싱된 데이터에 대해 리소스 캐시를 체크하기 위한 코드를 포함할 수 있다. 명령들은, 리소스 캐시에서의 캐시 히트(cache hit) 시, 직렬화된 코드를 포함하는 캐싱된 데이터를 리소스 캐시로부터 리트리브(retrieve)하기 위한 코드를 포함할 수 있다. 명령들은, 리트리브된 직렬화된 코드를 제 3 실행가능 코드로 역직렬화하기 위한 코드를 포함할 수 있다. 명령들은, 제 2 실행 컨텍스트에서의 실행을 위해 제 3 실행가능 코드를 제공하기 위한 코드를 포함할 수 있다.

[0005] [0005] 이들 및 다른 구현들은 다음의 특성들 중 하나 또는 그 초과를 포함할 수 있다. 제 1 실행 컨텍스트 및 제 2 실행 컨텍스트는 코드 실행을 위한 2개의 상이한 가상 머신 환경들일 수 있다. 1차 소스 코드는 웹 페이지 내의 스크립트일 수 있고, 제 1 실행 컨텍스트 및 제 2 실행 컨텍스트는, 스크립트가 실행되는 웹 브라우저 세션들일 수 있다. 1차 소스 코드는 웹 페이지의 최상위-레벨 스크립트일 수 있다. 제 1 실행가능 코드는, 제 1 실행가능 코드에 임베딩된, 제 1 실행 컨텍스트에 특정한 고정 메모리 어드레스를 포함하는 레퍼런스(reference)들의 세트를 포함할 수 있다. 제 1 실행가능 코드를 직렬화하기 위한 코드를 포함하는 명령들은, 임베딩된 메모리 어드레스들을 제 1 실행 컨텍스트에 비-특정(non-specific)인 추상(abstract) 어드레스들로 대체하기 위한 코드를 포함할 수 있다.

[0006] [0006] 하나의 혁신적인 양상에서, 개시된 발명의 대상은 시스템에서 구현될 수 있다. 시스템은 하나 또는 그 초과의 프로세서들을 포함한다. 시스템은 또한 메모리를 포함한다. 메모리는, 실행되는 경우 하나 또는 그 초과의 프로세서들로 하여금 방법을 구현하게 하는 명령들을 포함한다. 명령들은, 실행을 대기하는 1차 소스 코드의 제 1 표시를 수신하기 위한 코드를 포함한다. 명령들은, 제 1 표시를 수신하는 것에 대한 응답으로, 1차 소스 코드에 대응하는 캐싱된 데이터에 대해 리소스 캐시를 체크하기 위한 코드를 포함한다. 명령들은, 리소스 캐시에서의 캐시 미스 시, 1차 소스 코드로부터 컴파일된 제 1 실행가능 코드를 획득하기 위한 코드를 포함한다. 명령들은, 1차 소스 코드의 사이즈, 미리결정된 시간 기간 내에 1차 소스 코드가 실행되는 횟수, 또는 1차 소스 코드의 컴파일 시간 중 하나 또는 그 초과에 기초하여, 제 1 실행가능 코드를 직렬화된 코드로 직렬화하기 위한 코드를 포함한다. 명령들은, 직렬화된 코드를 캐싱된 데이터로서 리소스 캐시에 저장하기 위한 코드를 포함한다.

[0007] [0007] 이들 및 다른 구현들은 다음의 특성들 중 하나 또는 그 초과를 포함할 수 있다. 명령들은, 적어도, 2차 소스 코드의 사이즈, 1차 소스 코드에서 2차 소스 코드가 참조되는 횟수, 2차 소스 코드의 컴파일 시간, 또는 이들의 결합에 기초하여, 1차 소스 코드에서 참조되는 2차 소스 코드를 선택하기 위한 코드를 포함할 수 있다. 명령들은, 2차 소스 코드로부터 컴파일된 제 2 실행가능 코드를 획득하기 위한 코드를 포함할 수 있다. 명령들은, 제 2 실행가능 코드를 직렬화된 코드로 직렬화하기 위한 코드를 포함할 수 있다. 명령들은, 제 1 실행 컨텍스트에서의 제 1 실행가능 코드의 실행으로부터의 실행 결과들을 획득하기 위한 코드를 포함할 수 있으며, 여기서, 2차 소스 코드는 실행 결과들에 기초하여 선택된다. 명령들은, 제 2 실행 컨텍스트에서의 실행을 대기하는 1차 소스 코드의 제 2 표시를 수신하기 위한 코드를 포함할 수 있으며, 여기서, 제 2 표시는 제 1 표시에 후

속하여 수신된다. 명령들은, 제 2 표시를 수신하는 것에 대한 응답으로, 1차 소스 코드 및 선택된 2차 소스 코드에 대응하는 캐싱된 데이터에 대해 리소스 캐시를 체크하기 위한 코드를 포함할 수 있다. 명령들은, 리소스 캐시에서의 캐시 히트 시, 직렬화된 코드를 포함하는 캐싱된 데이터를 리소스 캐시로부터 리트리브하기 위한 코드를 포함할 수 있다. 명령들은, 리트리브된 직렬화된 코드를 제 3 실행가능 코드로 역직렬화하기 위한 코드를 포함할 수 있다. 명령들은, 제 2 실행 컨텍스트에서의 실행을 위해 제 3 실행가능 코드를 제공하기 위한 코드를 포함할 수 있다.

[0008] [0008] 이들 및 다른 구현들은 다음의 특성들 중 하나 또는 그 초과를 포함할 수 있다. 제 1 실행 컨텍스트 및 제 2 실행 컨텍스트는 코드 실행을 위한 2개의 상이한 가상 머신 환경들일 수 있다. 제 1 실행가능 코드는, 제 1 실행가능 코드에 임베딩된 메모리 어드레스들을 포함하는 레퍼런스들의 세트를 포함할 수 있고, 제 1 실행가능 코드를 직렬화하기 위한 코드를 포함하는 명령들은, 임베딩된 메모리 어드레스들을 추상 어드레스들로 대체하기 위한 코드를 포함할 수 있다.

[0009] [0009] 하나의 혁신적인 양상에서, 개시된 발명의 대상은, 컴퓨터에 의해 실행될 수 있는 명령들을 포함하는 컴퓨터-판독가능 매체에서 구현될 수 있다. 명령들은, 컴퓨터로 하여금, 실행을 대기하는 1차 소스 코드의 제 1 표시를 수신하게 하기 위한 코드를 포함한다. 명령들은, 컴퓨터로 하여금, 제 1 표시를 수신하는 것에 대한 응답으로, 1차 소스 코드에 대응하는 캐싱된 데이터에 대해 리소스 캐시를 체크하게 하기 위한 코드를 포함한다. 명령들은, 리소스 캐시에서의 캐시 미스 시, 컴퓨터로 하여금, 1차 소스 코드로부터 컴파일된 제 1 실행가능 코드를 획득하게 하기 위한 코드를 포함한다. 명령들은, 컴퓨터로 하여금, 제 1 실행 컨텍스트에서의 제 1 실행가능 코드의 실행으로부터의 실행 결과들을 획득하게 하기 위한 코드를 포함한다. 명령들은, 컴퓨터로 하여금, 실행 결과들에 기초하여, 1차 소스 코드에서 참조되는 2차 소스 코드를 선택하게 하기 위한 코드를 포함한다. 명령들은, 컴퓨터로 하여금, 선택된 2차 소스 코드로부터 컴파일된 제 2 실행가능 코드를 획득하게 하기 위한 코드를 포함한다. 명령들은, 컴퓨터로 하여금, 제 1 실행가능 코드 및 제 2 실행가능 코드를 직렬화된 코드로 직렬화하게 하기 위한 코드를 포함한다. 명령들은, 컴퓨터로 하여금, 직렬화된 코드를 캐싱된 데이터로서 리소스 캐시에 저장하게 하기 위한 코드를 포함한다.

[0010] [0010] 이들 및 다른 구현들은 다음의 특성들 중 하나 또는 그 초과를 포함할 수 있다. 제 1 실행가능 코드는, 제 1 실행가능 코드에 임베딩된 메모리 어드레스들을 포함하는 레퍼런스들의 세트를 포함할 수 있고, 컴퓨터로 하여금 제 1 실행가능 코드를 직렬화하게 하기 위한 명령들은, 컴퓨터로 하여금, 임베딩된 메모리 어드레스들을 추상 어드레스들로 대체하게 하기 위한 명령들을 포함할 수 있다. 실행 결과들은, 1차 소스 코드에서 참조되는 2차 소스 코드가 1차 소스 코드의 실행에서 실행된 횟수를 표시할 수 있다. 명령들은, 컴퓨터로 하여금, 제 2 실행 컨텍스트에서의 실행을 대기하는 1차 소스 코드의 제 2 표시를 수신하게 하기 위한 코드를 포함할 수 있으며, 여기서, 제 2 표시는 제 1 표시에 후속하여 수신된다. 명령들은, 컴퓨터로 하여금, 제 2 표시를 수신하는 것에 대한 응답으로, 1차 소스 코드 및 선택된 2차 소스 코드에 대응하는 캐싱된 데이터에 대해 리소스 캐시를 체크하게 하기 위한 코드를 포함할 수 있다. 명령들은, 리소스 캐시에서의 캐시 히트 시, 컴퓨터로 하여금, 직렬화된 코드를 포함하는 캐싱된 데이터를 리소스 캐시로부터 리트리브하게 하기 위한 코드를 포함할 수 있다. 명령들은, 컴퓨터로 하여금, 리트리브된 직렬화된 코드를 제 3 실행가능 코드로 역직렬화하게 하기 위한 코드를 포함할 수 있다. 명령들은, 컴퓨터로 하여금, 제 2 실행 컨텍스트에서의 실행을 위해 제 3 실행가능 코드를 제공하게 하기 위한 코드를 포함할 수 있다. 제 1 실행 컨텍스트 및 제 2 실행 컨텍스트는 코드 실행을 위한 2개의 상이한 가상 머신 환경들일 수 있다. 명령들은, 컴퓨터로 하여금, 제 1 실행가능 코드와 연관된 제 1 실행 프로파일 및 제 2 실행가능 코드와 연관된 제 2 실행 프로파일을 획득하게 하기 위한 코드를 포함할 수 있다. 명령들은 또한, 컴퓨터로 하여금, 제 1 실행 프로파일 및 제 2 실행 프로파일을 직렬화하게 하기 위한 코드를 포함할 수 있다. 명령들은 또한, 컴퓨터로 하여금, 직렬화된 제 1 실행 프로파일 및 제 2 실행 프로파일을 리소스 캐시에 저장하게 하기 위한 코드를 포함할 수 있으며, 여기서, 리트리브된 직렬화된 코드를 역직렬화하는 것은 제 1 실행 프로파일 및 제 2 실행 프로파일을 역직렬화하는 것을 포함한다.

[0011] [0011] 다음의 상세한 설명으로부터 본 기술의 다른 구성들이 당업자들에게 용이하게 명백해질 것이며, 여기서, 본 기술의 다양한 구성들이 예시로서 도시되고 설명된다는 것이 이해된다. 인식될 바와 같이, 본 기술은 다른 그리고 상이한 구성들이 가능하고, 이들의 몇몇 세부사항들은 다양한 다른 관점들에서 수정이 가능하며, 이들 전부가 본 기술의 범위로부터 벗어나지 않는다. 따라서, 도면들 및 상세한 설명은 제한적인 것으로서가 아닌 사실상 예시적인 것으로서 간주되어야 한다.

도면의 간단한 설명

[0012] [0012] 본 기술의 특성들은 첨부된 청구항들에 기재된다. 그러나, 설명의 목적들을 위해, 본 기술의 몇몇 구현들이 다음의 도면들에 기재된다.

[0013] [0013] 도 1은 코드 캐싱이 프로세싱될 수 있는 예시적인 네트워크 환경을 예시한다.

[0014] [0014] 도 2는 코드 캐싱이 제공될 수 있는 클라이언트 디바이스를 예시한다.

[0015] [0015] 도 3은, 스크립트의 캐싱된 코드로의 직렬화 및 캐싱된 코드의 실행가능 코드로의 역직렬화가 제공될 수 있는 코드-캐싱 플랫폼의 흐름도이다.

[0016] [0016] 도 4a-4b는, 스크립트의 캐싱된 코드로의 직렬화가 제공될 수 있는 프로세스들의 예들을 예시한다.

[0017] [0017] 도 5는, 본 기술의 몇몇 구현들이 구현되는 예시적인 전자 시스템을 개념적으로 예시한다.

발명을 실시하기 위한 구체적인 내용

[0013] 용어 해설

[0014] [0018] 추상화(Abstracting, Abstraction): 추상화는, 오브젝트(예컨대, 코드)가 새로운 실행 컨텍스트에서 재생성될 수 있도록, 레퍼런스들을 재생성하기 위한 명령들(예컨대, 재배치 데이터)을 이용하여 오브젝트의 레퍼런스들을 실행 컨텍스트에 특정한 메모리 어드레스들 및 다른 오브젝트들로 대체하는 프로세스이다.

[0015] [0019] 직렬화(Serializing, Serialization): 직렬화는, 전달될 수 있고 오브젝트(예컨대, 컴파일된 실행가능 코드)의 데이터뿐만 아니라 오브젝트에 관한 정보(이를테면, 오브젝트의 타입 또는 오브젝트에 저장된 데이터의 타입들)를 포함하는 순차적 데이터로 오브젝트의 표현을 생성하는 프로세스이다.

[0016] [0020] 역직렬화(Deserializing, Deserialization): 역직렬화는, 직렬화된 오브젝트(예컨대, 컴파일된 실행가능 코드)의 순차적 데이터로부터 오브젝트를 추출하여 생성하는 프로세스이다.

[0017] [0021] 소스 코드: 소스 코드는, 컴퓨터에 의해 수행될 동작들을 특정하는 컴퓨터 명령들의 집합이며, 인간-판독가능 컴퓨터 언어를 사용하여 기입된다.

[0018] [0022] 실행가능 코드: 실행가능 코드는, 컴퓨터에 의해 실행가능한 머신 언어 명령들의 세트이며, 소스 코드를 파싱(parse) 및 컴파일함으로써 생성된다.

[0019] [0023] 실행 컨텍스트(실행 환경): 실행 컨텍스트는, 실행가능 코드가 실행되는 컴퓨터 시스템 또는 컴퓨터 시스템의 소프트웨어 에뮬레이션(emulation)(예컨대, 가상 머신)을 정의하는 동작 파라미터들의 세트이다.

[0020] [0024] 재배치 데이터: 재배치 데이터는, 추상화된 코드 레퍼런스들을 현재 실행 컨텍스트의 메모리 위치들 또는 오브젝트들로 복원시키는데 사용되는 명령들 또는 정보이다.

[0021] [0025] 아래에 기재된 상세한 설명은 본 기술의 다양한 구성들의 설명으로서 의도되며, 본 기술이 실시될 수 있는 유일한 구성들만을 나타내도록 의도되지 않는다. 첨부된 도면들은 본원에 포함되며, 상세한 설명의 일부를 구성한다. 상세한 설명은 본 기술의 철저한 이해를 제공하려는 목적으로 특정 세부사항들을 포함한다. 그러나, 본 기술이 본원에 기재된 특정 세부사항들로 제한되지 않으며, 이러한 특정 세부사항들 없이도 실시될 수 있다는 것이 분명하고 명백할 것이다. 몇몇 예시들에서, 본 기술의 개념들을 불명료하게 하는 것을 회피하기 위해 구조들 및 컴포넌트들은 블록도 형태로 도시되어 있다.

[0022] [0026] 리소스들의 로드 시간은, 컴퓨팅 디바이스를 통해 리소스에 액세스하려 시도할 때의 사용자 경험에서 중요한 부분이다. 리소스는, 컴퓨팅 디바이스 내의 로컬 메모리로부터 로딩되는 애플리케이션, 파일, 또는 문서, 또는 예컨대 네트워크로부터 로딩되는 웹페이지 내의 파일 또는 문서와 같은 네트워크 리소스일 수 있다. 그러나, 예를 들어, 리소스 내에 임베딩된 JavaScript 코드와 같은 동적 스크립트들은 실행 전에 컴파일될 필요가 있으며, 이는 리소스를 로딩하는 프로세스를 둔화시킬 수 있다. 따라서, 컴파일된 코드를 리소스 캐시에 캐싱하고, 향후의, 후속 세션들 동안의 리소스에 대한 액세스들에 대해, 캐싱된 코드를 재사용함으로써, 컴퓨팅 디바이스들 상에서 리소스들을 렌더링하는 효율을 개선하는 것이 바람직하다.

[0023] [0027] 소스 코드는, 컴파일러(compiler) 프로그램에 의해, 컴퓨터에 의해 실행가능한 저-레벨 머신 코드로 변환될 수 있다. 그 다음, 머신 코드는 실행을 위해 저장될 수 있다. 대안적으로, 소스 코드 프로그램의 아웃컴(outcome)들을 직접 온 더 플라이(on the fly)로 분석 및 수행하기 위해 인터프리터(interpreter)가 사용될 수 있다. 실행가능 코드를 생성하기 위해 애플리케이션 또는 문서(이를테면, 예컨대 웹 페이지)에 임베딩된 소스

코드(예컨대, JavaScript 코드)를 파싱 및 컴파일하는 것은, 리소스를 로딩하는데 있어 최상 경로(critical path) 상에 있다. 실행가능 코드는, 컴퓨터 프로세서에 의해 실행되는 경우, 프로세서로 하여금 명령들에 따라 태스크들을 수행하게 한다. 머신 언어는, 컴퓨터가 하드웨어에서 수행하는 네이티브(native) 명령들의 세트이다. 로딩 프로세스의 속도를 높이기 위해, 컴파일된 코드는 예컨대 바이너리(binary) 포맷으로 직렬화 및 캐싱될 수 있다. 일단 실행가능 코드가 캐싱되면, 스크립트의 후속 실행들에 대해, 소스 코드는 더 이상 다시 파싱되고 다시 컴파일될 필요가 없으며, 실행가능 코드는 캐시로부터 역직렬화되어 실행될 수 있다.

[0024] [0028] 컴파일된 코드의 직렬화 동안, 코드의 상태는, 일 실행 환경으로부터 다른 실행 환경들로 전송될 수 있는 형태로 변환될 수 있다. 직렬화 프로세스는, 데이터 구조들 또는 오브젝트 상태들을, 직렬화된 오브젝트에 임베딩된 정보에 기초하여 동일한 컨텍스트(예컨대, 실행 환경) 또는 다른 컨텍스트에서 재구성될 수 있는 포맷으로 변환한다. 직렬화는, 컴파일된 코드로부터, 코드의 다양한 컴포넌트들 및 코드가 참조하는 오브젝트들(이를테면, 예컨대 상수들을 나타내는 데이터 구조들, 스트링 리터럴(string literal)들, 오브젝트 리터럴들, 온-디멘드 컴파일에 필요한 재배치 데이터를 포함하는 함수 오브젝트들, 공유된 코드 조각(piece)들, 실행 환경(예컨대, VM)에 대한 레퍼런스들 등)에 대한 실행가능 코드를 제공할 수 있다.

[0025] [0029] 도 1은 코드 캐싱이 프로세싱될 수 있는 예시적인 네트워크 환경(100)을 예시한다. 네트워크 환경(100)은 클라이언트 디바이스들(101a 및 101n) 및 서버들(109a 및 109m)을 포함한다. 2개의 클라이언트 디바이스들(101a 및 101n) 및 2개의 서버들(109a 및 109m)이 도 1에 도시되지만, 본 기술은 이렇게 제한되지 않으며, 2개 초과의 클라이언트 디바이스들 및 서버들 또는 단지 하나의 클라이언트 디바이스 및/또는 서버에 적용될 수 있다. 클라이언트 디바이스들(101a 및 101n) 및 서버들(109a 및 109m)은 네트워크(105)를 통해 서로 통신할 수 있다. 서버(109a 또는 109m)는 클라이언트 디바이스들과 유사한 컴퓨팅 디바이스들 및 컴퓨터-판독가능 저장 디바이스들(도시되지 않음)을 포함할 수 있다.

[0026] [0030] 클라이언트 디바이스들(101a 및 101n) 각각은 다양한 형태들의 프로세싱 디바이스들을 나타낼 수 있다. 예시적인 프로세싱 디바이스들은, 데스크톱 컴퓨터, 랩톱 컴퓨터, 핸드헬드(handheld) 컴퓨터, PDA(personal digital assistant), 셀룰러 텔레폰, 네트워크 어플라이언스(appliance), 카메라, 스마트 폰, EGPRS(enhanced general packet radio service) 모바일 폰, 미디어 플레이어, 내비게이션 디바이스, 이메일 디바이스, 게임 콘솔, 또는 임의의 이들 데이터 프로세싱 디바이스들 또는 다른 데이터 프로세싱 디바이스들의 결합을 포함할 수 있다. 클라이언트 디바이스들(101a 및 101n) 및 서버들(109a 및 109m)은, 다른 클라이언트 디바이스들(101a 및 101n) 또는 서버들(109a 및 109m) 중 임의의 것 상에서 실행되거나 그에 저장된 애플리케이션 소프트웨어를 수신하거나, 이러한 애플리케이션 소프트웨어에 대한 액세스를 제공받을 수 있다.

[0027] [0031] 서버들(109a 및 109m)은, 프로세서, 메모리, 및 클라이언트 디바이스들(101a 및 101n)에 컨텐츠를 제공하기 위한 통신 능력을 갖는 임의의 시스템 또는 디바이스일 수 있다. 몇몇 예시적인 양상들에서, 서버(109a 또는 109m)는 단일 컴퓨팅 디바이스, 예컨대 컴퓨터 서버일 수 있다. 다른 구현들에서, 서버(109a 또는 109m)는, 서버 컴퓨터의 동작들(예컨대, 클라우드 컴퓨팅)을 수행하도록 함께 동작하는 하나 초과의 컴퓨팅 디바이스를 나타낼 수 있다. 추가로, 서버(109a 또는 109m)는, 웹 서버, 애플리케이션 서버, 프록시 서버, 네트워크 서버, 또는 서버 팜(server farm)을 포함(그러나 이에 제한되지 않음)하는 다양한 형태들의 서버들을 나타낼 수 있다.

[0028] [0032] 몇몇 양상들에서, 클라이언트 디바이스들(101a 및 101n)은, 통신 인터페이스(도시되지 않음)를 통해 무선으로 통신할 수 있고, 통신 인터페이스는, 필요한 경우 디지털 신호 프로세싱 회로를 포함할 수 있다. 통신 인터페이스는, 다양한 모드들 또는 프로토콜들, 예컨대, 다른 것들 중에서도, GSM(Global System for Mobile communication) 음성 호(call)들, SMS(Short Message Service), EMS(Enhanced Messaging Service) 또는 MMS(Multimedia Messaging Service) 메시징, CDMA(Code Division Multiple Access), TDMA(Time Division Multiple Access), PDC(Personal Digital Cellular), WCDMA(Wideband Code Division Multiple Access), CDMA2000, 또는 GPRS(General Packet Radio System) 하의 통신들을 제공할 수 있다. 예를 들어, 통신은 라디오-주파수 트랜시버(도시되지 않음)를 통해 발생할 수 있다. 부가하여, 단거리 통신은, 예를 들어 Bluetooth, WiFi, 또는 다른 그러한 트랜시버를 사용하여 발생할 수 있다.

[0029] [0033] 몇몇 양상들에서, 네트워크 환경(100)은 네트워크들(예컨대, 네트워크(105))에 걸쳐 있는 분산형 클라이언트/서버 시스템일 수 있다. 네트워크(105)는 대형 컴퓨터 네트워크, 예컨대 임의의 개수의 모바일 클라이언트들, 고정 클라이언트들, 및 서버들을 연결하는 LAN(local area network), WAN(wide area network), 인터넷, 셀룰러 네트워크 또는 이들의 결합일 수 있다. 추가로, 네트워크(105)는, 버스 네트워크, 스타(star)

네트워크, 링(ring) 네트워크, 메시(mesh) 네트워크, 스타-버스 네트워크, 트리(tree) 또는 계층적 네트워크 등을 포함하는 네트워크 기술들 중 임의의 것을 포함(이들로 제한되지 않음)할 수 있다. 몇몇 양상들에서, 각각의 클라이언트(예컨대, 클라이언트 디바이스들(101a 및 101n))와 서버(109a 또는 109m) 간의 통신은, VPN(virtual private network), SSH(Secure Shell) 터널, 또는 다른 보안 네트워크 연결을 통해 발생할 수 있다. 몇몇 양상들에서, 네트워크(105)는, 기업 네트워크(예컨대, 인트라넷) 및 무선 액세스 포인트들을 더 포함할 수 있다.

[0030] [0034] 예시적인 양상들에서, 클라이언트 디바이스들(101a 또는 101n) 중 임의의 클라이언트 디바이스는, 예를 들어, HTTP(Hypertext Transfer Protocol) 프로토콜과 같은 네트워크 프로토콜을 사용하여 네트워크(105)를 통해 서로 그리고 서버들(109a 또는 109m)과 통신할 수 있다. 클라이언트 디바이스(101a 또는 101n)는, 클라이언트 디바이스(101a 또는 101n)의 사용자 인터페이스 내의 애플리케이션(107a 또는 107n)(예컨대, 웹 브라우저)에 대한 사용자 입력을 수신할 수 있다. 사용자 입력은 어느 컨텐츠를 로딩할 것인지를 특정할 수 있다. 예를 들어, 사용자 입력은, 서버(109a 또는 109m)로부터 로딩될 웹 컨텐츠를 특정하는, 웹사이트와 연관된 URL(Uniform Resource Locator) 어드레스일 수 있다. 사용자는 또한, 사용자가 클라이언트 디바이스(101a 또는 101n)(현재 도 1에 도시됨)의 사용자 인터페이스(UI)를 통해 보기를 희망하는 웹 페이지의 부분을 특정하기 위해 입력을 입력할 수 있다. 이러한 예시적인 양상들에서, 클라이언트 디바이스들(101a 또는 101n) 각각은 코드-캐싱 플랫폼(103a 또는 103n)을 포함할 수 있는데, 코드-캐싱 플랫폼(103a 또는 103n)은, 사용자로부터의 요청에 기초하여, 캐싱된 실행가능 코드에 대한 캐시를 체크하고, 클라이언트 디바이스(101a 또는 101n) 상에서의 실행을 위해 캐싱된 실행가능 코드를 제공할 수 있다. 코드-캐싱 플랫폼(103a 또는 103n)은 또한, 캐싱을 위해, 실행가능 코드의 부분들을 선택할 수 있다.

[0031] [0035] 도 2는 코드 캐싱이 제공될 수 있는 클라이언트 디바이스를 예시한다. 클라이언트 디바이스(200)는 도 1의 클라이언트 디바이스들(101a 및 101n)과 유사하다. 도시된 바와 같이, 클라이언트 디바이스(200)는 프로세서(201), 네트워크 인터페이스(203), 및 메모리(205)를 포함한다. 프로세서(201)는, 컴퓨터-판독가능 매체, 예컨대 메모리(205)에 저장된 컴퓨터 명령들을 실행하도록 구성된다. 프로세서(201)는 CPU(central processing unit)일 수 있다. 네트워크 인터페이스(203)는, 클라이언트 디바이스(201)가 네트워크(105)(예컨대, 인터넷, 인트라넷, 로컬 영역 네트워크 또는 셀룰러 네트워크)에서 데이터를 송신 및 수신하는 것을 허용하도록 구성된다. 네트워크 인터페이스(203)는 NIC(network interface card)들을 포함할 수 있다.

[0032] [0036] 메모리(205)는 데이터 및 명령들을 저장한다. 도시된 바와 같이, 메모리(205)는, 도 1의 애플리케이션들(107a 및 107n)과 유사한 애플리케이션(207), 및 도 1의 코드-캐싱 플랫폼(103a 및 103n)과 유사한 코드-캐싱 플랫폼(209)을 포함한다. 애플리케이션(207) 및 코드-캐싱 플랫폼(209)은 가상 머신(211)과 같은 실행 환경 내에서 기능할 수 있다.

[0033] [0037] 애플리케이션(207)은, 프로세서(201)에 의해 클라이언트 디바이스(200) 상에서 실행되는 소프트웨어 애플리케이션일 수 있다. 예를 들어, 애플리케이션(207)은, 클라이언트 디바이스(200)의 사용자에게, 네트워크(105)를 통해 서버들(109a 또는 109m)에 의해 제공되는 서비스들에 대한 액세스를 제공하는 웹 브라우저일 수 있다. 서버들(109a 및 109m)은 웹사이트들의 형태로 클라이언트 디바이스(200)에 서비스들을 제공할 수 있고, 서버(109a 또는 109m)에 의해 제공되는 웹 페이지는, 클라이언트 디바이스(200)에서 로딩될 때 컴파일 및 실행되는 다수의 소스 코드 또는 스크립트들(예컨대, JavaScript)을 포함할 수 있다.

[0034] [0038] 코드-캐싱 플랫폼(209)은 코드-캐싱 서비스들을 애플리케이션(207)에 제공하도록 구성될 수 있으며, 여기서, 애플리케이션(207)은 서버들(109a 및 109m)에 의해 클라이언트 디바이스(200)에 제공되는 서비스들을 렌더링한다. 몇몇 양상들에서, 코드-캐싱 플랫폼(209)은, 클라이언트 디바이스(200) 상에서의 실행을 대기하는 애플리케이션(207)에 임베딩된 소스 코드의 표시를 수신한다. 소스 코드의 표시는, 애플리케이션(207)에 임베딩된 소스 코드의 메모리(205)에서의 위치를 지칭하는 링크일 수 있다. 표시는 또한, 소스 코드와 연관된 식별자 또는 소스 코드의 전체 컨텐츠일 수 있다. 표시를 수신하는 것에 대한 응답으로, 코드-캐싱 플랫폼(209)은, 소스 코드에 대응하는 캐싱된 실행가능 코드에 대해, 메모리(205) 내의 리소스 캐시(213)를 체크할 수 있다. 리소스 캐시(213)는 영구적/비-휘발성 메모리에 저장될 수 있다.

[0035] [0039] 예를 들어, 소스 코드, 또는 소스 코드의 부분들(예컨대, 서브-코드, 함수 등)은 이전에 컴파일되어 리소스 캐시(213)에 캐싱되었을 수 있다. 2차 소스 코드로 또한 지칭되는 서브-코드는, 소스 코드 내에서 참조된(예컨대, 호출된) 함수일 수 있다. 2차 소스 코드는, 예를 들어, 사용자 입력, 다른 2차 소스 코드로부터의 출력 등과 같은 다양한 조건들에 기초하여 소스 코드 내에서 참조될 수 있다. 부가하여, 예컨대 타입 정보와 같

은 소스 코드에 관련된 데이터가 캐싱되었을 수 있다. 캐싱된 데이터는, 실행가능 코드를 더 효율적으로 생성하기 위해 컴파일러에 의해 사용될 수 있다. 예를 들어, 코드-캐싱 플랫폼(209)은, 실행가능 코드로부터 컴파일된 소스 코드를 클라이언트 디바이스(200) 내의 컴파일러(도 2에 도시되지 않음)에 전송함으로써, 소스 코드로부터 컴파일된 실행가능 코드를 획득할 수 있고, 클라이언트 디바이스(200) 상에서의 실행을 위해 실행가능 코드를 제공할 수 있다. 소스 코드 또는 소스 코드에서 참조되는 2차 소스 코드에 대응하는 캐싱된 실행가능 코드가 리소스 캐시(213)에서 발견되지 않으면, 코드-캐싱 플랫폼(209)은, 컴파일될 소스 코드에서 참조되는 2차 소스 코드를 선택할 수 있다. 코드-캐싱 플랫폼(209)은, 소스 코드에 대응하는 실행가능 코드 및/또는 2차 소스 코드에 대응하는 실행가능 코드를 캐싱할 수 있다. 그러나, 소스 코드에 대응하는 캐싱된 실행가능 코드가 리소스 캐시(213)에서 발견되면, 캐싱된 실행가능 코드는 리소스 캐시로부터 리트리브될 수 있고, 소스 코드의 컴파일을 필요로 하지 않고도 실행될 수 있다.

[0036] [0040] 실행가능 코드는 레퍼런스들의 세트를 포함할 수 있다. 실행가능 코드에 포함된 레퍼런스들은, 예를 들어, 실행가능 코드가 실행되는 실행 환경(예컨대, 가상 머신(211))에 대응하는 실행가능 코드에 임베딩된 메모리 어드레스들일 수 있다. 임베딩된 어드레스들은 일 실행 컨텍스트로부터 다른 실행 컨텍스트로 이식가능(portable)하지 않을 수 있다. 그러나, 임베딩된 어드레스들을 추상화함으로써 생성되는 재배치 데이터는 실행 컨텍스트들에 걸쳐 이식가능할 수 있다. 재배치 데이터는, 예를 들어, 이식(porting) 이후에 비-이식가능 어드레스들을 복원함으로써, 실행가능 코드의 임베딩된 어드레스들의 추상화를 구현하는데 사용될 수 있다. 소스 코드 및 선택된 2차 소스 코드에 대한 실행가능 코드를 획득할 시, 코드-캐싱 플랫폼(209)은 실행가능 코드들을 직렬화한다. 직렬화는, 실행 컨텍스트(예컨대, 가상 머신(211))로부터의 실행가능 코드의 레퍼런스들의 세트를 추상화하는 것을 포함한다. 실행 컨텍스트는, 실행가능 코드를 실행하기 위한 프로세싱 전력 및 메모리에 대한 액세스를 컴퓨터 시스템 내의 동작 환경에 제공한다. 코드-캐싱 플랫폼(209)은, 실행가능 코드와 연관된 재배치 데이터를 추상화들을 반영하도록 생성함으로써, 생성된 재배치 데이터에 기초하여, 추상화된 레퍼런스들의 추가적인 변환이 수행될 수 있다.

[0037] [0041] 코드-캐싱 플랫폼(209)은 소스 코드 및 소스 코드에서 참조되는 선택된 2차 소스 코드에 대한 직렬화된 코드를 캐싱된 데이터로서 메모리(205) 내의 리소스 캐시(213)에 저장할 수 있다. 리소스 캐시를 관리하기 위해 종래의 캐시 관리 전략들이 사용될 수 있다. 부가하여, 리소스 캐시에서의 캐싱된 데이터 위치를 식별하기 위해 캐시 태그들이 사용될 수 있다. 코드-캐싱 플랫폼(209)은 또한, 가상 머신(211)에서의 실행을 위해 실행가능 코드를 제공할 수 있다. 몇몇 양상들에서, 직렬화된 코드를 리소스 캐시(213)에 저장한 후의 시점에, 코드-캐싱 플랫폼(209)은, 상이한 가상 머신 환경에서, 실행을 대기하는 소스 코드의 제 2 표시를 수신할 수 있다. 예를 들어, 앞서 논의된 바와 같은 실행가능 코드의 실행 이후, 클라이언트 디바이스(200)는 재시작 또는 재부팅되었을 수 있다. 논의된 바와 같이, 표시를 수신하는 것에 대한 응답으로, 코드-캐싱 플랫폼(209)은, 소스 코드 또는 소스 코드에서 참조되는 선택된 2차 소스 코드에 대응하는 캐싱된 실행가능 코드에 대해, 메모리(205) 내의 리소스 캐시(213)를 체크할 수 있다. 2차 소스 코드가 이전에 컴파일되어 리소스 캐시(213)에 캐싱되었으므로, 2차 소스 코드에 대응하는 캐싱된 실행가능 코드가 리소스 캐시(213)에서 발견될 수 있다. 리소스 캐시(213)에서의 캐시 히트 시, 코드-캐싱 플랫폼(209)은 메모리(205) 내의 리소스 캐시(213)로부터 캐싱된 데이터를 리트리브한다.

[0038] [0042] 캐싱된 코드는, 이전 컴파일 동안 2차 소스 코드로부터 추상화된 레퍼런스들에 대응하는 직렬화된 코드를 포함한다. 코드-캐싱 플랫폼(209)은, 리트리브된 직렬화된 코드들을 새로운 실행가능 2차 소스 코드들로 역직렬화할 수 있다. 역직렬화 프로세스는, 초기 코드에 관한 직렬화된 데이터, 초기 코드에 의해 참조되는 오브젝트들의 타입들, 및 오브젝트들에 저장된 데이터의 타입들을 수반하는 정보(예컨대, 재배치 데이터)를 사용하여, 저장된 데이터의 시퀀스로부터 코드를 추출한다. 역직렬화를 통해 생성되는 실행가능 코드는, 초기 실행가능 코드와 유사한 구조, 특성들, 및 거동(behavior)을 갖는다. 따라서, 실행가능 코드 및 실행가능 코드에 의해 참조되는 오브젝트들의 컴퓨터에 의한 실행은, 초기 코드의 실행에 의해 수행되는 것과 동일한 태스크들을 수행한다. 코드에 의해 참조되는 오브젝트들 및 코드 둘 모두는 직렬화 및 역직렬화를 겪을 수 있다. 역직렬화는, 재시작 시에 클라이언트 디바이스(200) 상에 설정된 새로운 가상 머신 환경에서 새로운 실행가능 2차 소스 코드가 실행가능하도록, 리트리브된 직렬화된 코드를 사용하여 새로운 실행가능 2차 소스 코드의 각각의 2차 소스 코드로부터의 레퍼런스들의 세트를 복원하는 것을 포함할 수 있다. 그 다음, 코드-캐싱 플랫폼(209)은, 클라이언트 디바이스(200) 상의 새로운 가상 머신 환경에서의 실행을 위해 새로운 실행가능 2차 소스 코드를 제공할 수 있다. 새로운 실행가능 2차 소스 코드는, 캐싱되지 않았고 이 실행을 위해 컴파일되는, 소스 코드에서 참조되는 다른 2차 소스 코드와 함께 실행될 수 있다. 코드-캐싱 플랫폼(209)은, 소스 코드의 상이한 실행들에서, 캐싱을 위한 새로운 2차 소스 코드를 선택할 수 있다. 코드-캐싱 플랫폼(209)은 또한, 소스 코드의 2차 소

스 코드의 실행 이력에 기초하여, 앞서 선택되어 캐싱된 2차 소스 코드를 다른 2차 소스 코드로, 예컨대 더 빈번하게 실행되는 2차 소스 코드로 대체할 수 있다.

[0039] [0043] 앞서 논의된 바와 같이, 코드-캐싱 플랫폼(209)에 의한 코드 캐싱은, 네트워크(105)를 통한 서버(109a 또는 109b)와의 통신 없이도 클라이언트 디바이스(101a 또는 101b) 내에서 수행될 수 있다. 예를 들어, 일 예시적인 양상에서, 코드-캐싱 플랫폼(209)에 의해 직렬화 및 역직렬화되는 소스 코드는, 클라이언트 디바이스(101a 또는 101n)의 메모리 상에 로컬로 저장된 코드일 수 있다.

[0040] [0044] 도 3은, 소스 코드 및/또는 소스 코드에서 참조되는 선택된 서브-코드들의 컴파일 및 캐싱된 코드로의 직렬화, 및 캐싱된 코드의 실행가능 코드로의 역직렬화가 제공될 수 있는 코드-캐싱 플랫폼의 흐름도이다. 도 3에 도시된 바와 같이, 코드-캐싱 플랫폼(209)은, 프론트-엔드(front-end)(303), 직렬화 모듈(309), 및 역직렬화 모듈(319)을 포함할 수 있다. 프론트-엔드(303)는 체크 모듈(305) 및 압축 모듈(307)을 포함할 수 있다. 프론트-엔드(303)는 소스 코드(301)를 수신한다. 소스 코드를 수신할 시, 체크 모듈(305)은, 소스 코드에 대응하는 캐싱된 직렬화된 코드에 대해 데이터 저장소(313)를 체크한다(화살표(311)로 도시됨). 데이터 저장소(313)에서 직렬화된 코드가 발견되지 않으면(캐시 미스), 프론트-엔드(303)는 소스 코드를 컴파일러(327)에 전송한다(화살표(325)로 도시됨). 컴파일러(327)는 소스 코드를 컴파일하고, 실행가능 코드 및 레퍼런스들을 저장한다. 컴파일러(327)는, 실행 환경(331)과 연관된 메모리 위치(도 3에 도시되지 않음)에 실행가능 코드 및 레퍼런스들을 저장할 수 있다. 실행가능 코드는, 브라우저 세션 내에서, 실행 환경(331)(예컨대, 도 2의 가상 머신(211)과 유사한 가상 머신) 내에서의 실행을 위해 실행 환경(331)에 전송될 수 있다(화살표 329로 도시됨).

[0041] [0045] 컴파일러(327)는 실행가능 코드 및 레퍼런스들을 직렬화를 위해 직렬화 모듈(309)에 전송할 수 있다(335로 도시됨). 프론트-엔드(303)는, 예를 들어, 지난 실행들에서의 부분들의 실행의 빈도에 기초하거나 소스 코드 내의 그 부분들을 참조하는 함수 호출들의 빈도에 기초하여, 직렬화를 위해, 소스 코드에서 참조되는 부분들(예컨대, 2차 소스 코드들)을 선택할 수 있다. 코드-캐싱 플랫폼(209)은, 다양한 기준들(예컨대, 경험칙(heuristic)들)에 기초하여, 직렬화 및 캐싱을 위한, 소스 코드 및/또는 소스 코드에서 참조되는 2차 소스 코드를 선택할 수 있다. 몇몇 예시적인 기준들은, 소스 코드 또는 2차 소스 코드의 사이즈; 소스 코드 또는 2차 소스 코드들의 나이(age)(예컨대, 소스 코드 또는 2차 소스 코드의 마지막 수정 날짜 또는 만료 날짜에 기초함); 소스 코드 또는 2차 소스 코드들의 사용 빈도(예컨대, n번째 인카운터(encounter) 이후 코드를 캐싱함); 리소스 캐시(213)에서 대체되는 캐싱된 코드의 빈도; 2차 소스 코드가 1차 소스 코드에서 참조되는 횟수; 또는 2차 소스 코드의 컴파일 시간일 수 있다. 코드-캐싱 플랫폼(209)은, 직렬화 및 캐싱을 위한, 소스 코드에서 참조되는 2차 소스 코드를, 그 2차 소스 코드가 소스 코드에서 참조된다고 예측하는 것에 기초하여 선택할 수 있다. 예를 들어, 코드-캐싱 플랫폼(209)은, 소스 코드가 컴파일될 때, 2차 소스 코드가 소스 코드에서 참조된다고 예측할 수 있다. 소스 코드로부터의 컴파일 결과들은, 실행 동안 2차 소스 코드가 소스 코드에 의해 참조될 것이라고 표시할 수 있다.

[0042] [0046] 코드-캐싱 플랫폼(209)은, 소스 코드가 임계치 S보다 작은 사이즈를 갖는 경우 소스 코드를 직렬화할 수 있다. 큰 코드 조각들을 직렬화하는 것은 긴 시간이 소요될 수 있고, 캐싱된 데이터를 저장하기 위해 많은 양의 메모리(예컨대, 클라이언트 디바이스(101a-101n) 상의 이용가능한 디스크 공간에 기초하여 결정됨)를 요구할 수 있다는 것이 그 이유이다.

[0043] [0047] 다른 예시적인 기준들은, 캐시 액세스 레이턴시(latency)(예컨대, 캐시 엔트리 사이즈); 소스 코드 또는 소스 코드의 부분들의 컴파일 시간(예컨대, 소스 코드의 초기 컴파일이 얼마나 오래 걸리는지를 기록하고 판단에서 그 시간을 고려함), 리소스 캐시(213)를 호스팅하는 클라이언트 디바이스(101a-101n)의 메모리 용량; 소스 코드 업데이트의 빈도(예컨대, 소스 코드가 빈번하게 업데이트되면, 소스 코드를 캐싱하는 것은 유익하지 않을 수 있음) 등일 수 있다. 코드-캐싱 플랫폼(209)은, 직렬화할 코드의 선택에 앞서 위의 팩터들 간의 트레이드-오프(trade-off)를 계산할 수 있다. 예를 들어, 위에 논의된 다양한 기준들의 중요성의 레벨을 결정하기 위해, 소스 코드 또는 소스 코드의 부분들에 대한 가중치 팩터(weight factor)들이 정의될 수 있다. 트레이드-오프 계산은, 예를 들어, 정의된 가중치 팩터들에 기초하여 수행될 수 있다. 예를 들어, 코드 사이즈와 클라이언트 디바이스(101a-101n)의 메모리 용량 간의 트레이드-오프는, 코드가 빈번하게 호출됨에도 불구하고, 클라이언트 디바이스(101a-101n) 상의 상당한 메모리 공간을 점유할 수 있는 전술한 큰 사이즈를 갖는 코드의 캐싱을 초래할 수 있다.

[0044] [0048] 다양한 예시들에서, 프론트-엔드는, 실행가능 코드 및 레퍼런스들을 직렬화할지 여부를 판단할 수 있다. 예를 들어, 직렬화될 소스 코드 또는 소스 코드의 부분들의 타입들 및 직렬화 없이 컴파일 및 실행될 타입들에

대한 몇몇 기준들이 서버들(109a 또는 109m)에 의해 정의될 수 있다. 부가하여, 특정 타입들의 소스 코드는, 보안 목적들을 위해 소스 코드의 매 실행에 앞서 검증될 필요가 있을 수 있다. 이러한 경우들에서, 프론트-엔드(303)는, 소스 코드 또는 소스 코드의 부분들을 직렬화되지 않을 소스 코드로 플래깅(flag)할 수 있다.

[0045] [0049] 다른 코드 캐싱 기준들은, 미리정의된 횟수를 초과하여 컴파일되는 코드들에 대한 코드 캐싱의 적용을 포함할 수 있다. 예를 들어, 처음 캐시 미스가 검출된 때, 소스 코드는 컴파일되지만 직렬화되거나 캐싱되지는 않을 수 있다. 그 소스 코드는, 코드의 직렬화가 수행됨이 없이 "한 번 컴파일된 것"으로 마킹(mark)될 수 있다. 유사하게, 소스 코드는, 미리정의된 임계치 C에 도달할 때까지 소스 코드가 컴파일된 횟수를 표시하는 카운터로 마킹될 수 있으며, C번째 캐시 미스 시, 코드-캐싱 플랫폼(209)은 소스 코드를 직렬화 및 캐싱할 수 있다. 대안적으로, 일반적인 마커(marker) 대신, 첫 번째 캐시 미스 상에 타임스탬프가 셋팅될 수 있으며, 첫 번째 캐시 미스와 C번째 캐시 미스 간의 시간 차이가 사용되어 직렬화가 판단될 수 있다. 그 기준들은, 예를 들어, "컴파일된 코드가 적어도 일주일에 C회 실행되면, 컴파일된 코드를 캐싱함"으로 해석될 수 있다. 소스 코드를 캐싱할지를 판단하기 위한 다른 기준들은, 이를테면, 예컨대 컴파일 시간, 생성된 코드 사이즈, HTTP 캐시 헤더 데이터 등으로 간주될 수 있다.

[0046] [0050] 코드-캐싱 플랫폼(209)은, 캐싱되고 후속 실행들에서 추가적으로 사용될 2차 소스 코드를 선택하기 위해, 소스 코드의 실행 데이터를 사용할 수 있다. 실행 데이터는, 예를 들어, 실행 동안에 실행된 소스 코드가 참조하는 다른 코드 조각들, 소스 코드의 실행 프로파일, 캐싱할 가치가 있을 만큼 충분히 자주 소스 코드가 실행되는지 여부(예컨대, 미리정의된 횟수를 초과하는지 여부), 실행 동안 관측되는 타입(예컨대, 사용자 정의 데이터 구조들) 정보 등을 포함할 수 있다. 부가하여, 실행 데이터가 또한 직렬화될 수 있으므로, 데이터는 실행 컨텍스트에서 역직렬화될 수 있다. 역직렬화 이후, 실행 데이터는 역직렬화된 코드와 함께 사용될 수 있다. 예를 들어, 실행 프로파일은, 실행가능 코드를 최적화하는데 시간을 소비할지 여부를 판단하기 위해 실행 컨텍스트에서 사용될 수 있다.

[0047] [0051] 예를 들어, JavaScript와 같은 동적으로 타이핑되는(dynamically-typed) 언어들은, 소스 코드에 정의된 오브젝트들에 대한 풍부한 타입 정보를 제공한다. 동적 타입들은, 소스 코드의 상이한 실행들 동안 소스 코드의 오브젝트가 상이한 타입들을 갖는 것을 가능하게 한다. 그러나, 소스 코드의 다양한 실행들을 관측하는 것은, 오브젝트의 타입이 얼마나 자주 변하는지에 대한 통찰들을 제공할 수 있다.

[0048] [0052] 특정 오브젝트 O의 타입 T는 오브젝트 컨텐츠에 기초하여 결정될 수 있다. 그러나, 소스 코드가 오브젝트 O와 상호작용하는 경우, 소스 코드는 타입 정보에 액세스할 수 없으며, 따라서, 오브젝트 타입 T는 소스 코드에 알려져 있지 않다. 결과적으로, 소스 코드는 오브젝트 O에 액세스하기에 앞서 타입 T를 결정할 필요가 있다. 몇몇 양상들에서, 코드-캐싱 플랫폼(209)은 오브젝트 타입 T를 결정 및 캐싱한다. 예를 들어, 코드-캐싱 플랫폼(209)은, 실행 환경에서(예컨대, 웹사이트에서) 오브젝트 O가 액세스되는 때에 오브젝트의 특정 타입을 소스 코드에서 관측할 수 있다. 코드-캐싱 플랫폼(209)은 액세스 웹사이트를 웹사이트에 의해 액세스되는 오브젝트의 타입 T와 연관시킬 수 있다. 타입 T와 함께, 코드-캐싱 플랫폼(209)은 액세스 시간에 결정된 "액세스 방식" 정보를 또한 캐싱할 수 있다. 후속하여, 오브젝트 O가 동일한 실행 환경에서 소스 코드에 의해 액세스되는 경우, 예상되는 타입 T 및 오브젝트 O에 액세스하기 위한 액세스 방법이, 캐싱된 데이터로부터 알려진다. 코드-캐싱 플랫폼(209)은 또한, 예상되는 타입 T가 현재 액세스에서의 오브젝트의 타입과 동일한지 여부를 체크할 수 있다. 타입들이 매칭하면, 필요한 액세스 방법에 대한 추가적인 결정 없이, 캐싱된 액세스 방법이 오브젝트 O에 액세스하는데 사용될 수 있다.

[0049] [0053] 부가하여, 오브젝트 타입 T는, 실행 컨텍스트에서 오브젝트 O에 액세스하기 위한 액세스 방법에 대한 정보를 제공할 수 있다. 일단 오브젝트 O에 액세스하기 위한 액세스 방법이 결정되면, 액세스 방법에 관한 정보는 타입 정보로서 그리고 액세스 방법과 연관되어, 직렬화 동안 리소스 캐시에 저장될 수 있다. 저장된 타입 정보를 사용함으로써, 역직렬화 시, 오브젝트 O에 대한 후속 액세스들은 액세스 방법을 결정할 필요 없이 더 빠르게 이루어질 수 있다.

[0050] [0054] 직렬화 모듈(309)에 의한 직렬화는, 실행가능 코드 내의 어드레스들을 추상 어드레스들로 대체함으로써 실행가능 코드 내의 레퍼런스들을 추상화할 수 있다. 직렬화 모듈(309)은, 실행가능 코드에 대응하는 직렬화된 코드 및 실행가능 코드 내의 레퍼런스들의 추상화에 대응하는 재배치 데이터를 프론트-엔드(303)에 제공한다(323으로 도시됨). 직렬화된 코드를 제공하면서, 직렬화 모듈(309)은, 실행 환경(331)으로부터의 실행가능 코드 및 레퍼런스들에 대응하는 관련 부분들 및 레퍼런스들을 발견할 수 있다. 직렬화 모듈(309)은, 그 부분들 및 레퍼런스들을 직렬화된 코드에 포함될 연속된(contiguous) 바이너리 표현(실행 환경(331)과 독립적임)으로

변환할 수 있다. 프론트-엔드(303)는, 직렬화 모듈(309)에 의해 생성되는 직렬화된 코드를 캐싱된 직렬화된 코드로서 데이터 저장소(313)에 저장할 수 있다. 직렬화 모듈(309)은, 직렬화된 코드 내의 다른 데이터 타입들과 코드를 구별하기 위한 태그를 사용하여, 추상화에 대응하는 직렬화된 코드를 인코딩할 수 있다.

[0051] [0055] 다른 예시에서, 직렬화된 코드가 프론트-엔드(303)에 의해 데이터 저장소(313)에 그리고 메타데이터(315) 내의 재배치 데이터에 저장된 후, 프론트-엔드(303)는, 상이한 실행 환경(333), 예컨대 새로운 브라우저 세션 내의 새로운 VM 내에서 소스 코드의 실행에 대한 새로운 요청을 수신할 수 있다. 이러한 예시에서, 체크 모듈(305)은, 캐싱된 직렬화된 코드를 데이터 저장소(313)에서 성공적으로 발견(캐시 히트)할 수 있다. 캐시 히트 시, 프론트-엔드(303)는 캐싱된 직렬화된 코드를 데이터 저장소(313)로부터 로딩할 수 있고(317로 도시됨), 로딩된 캐싱된 직렬화된 코드를 역직렬화 모듈(319)로 전송할 수 있다. 캐싱된 직렬화된 코드를 로딩하기에 앞서, 프론트-엔드(303)는, 캐싱된 직렬화된 코드가 소스 코드의 동일한 버전에 대응하는지 여부를 결정할 수 있다. 버전들이 매칭하지 않으면, 프론트-엔드(303)는, 캐싱된 직렬화된 코드를 역직렬화하지 않는다고 결정하고 소스 코드가 컴파일되어야 한다는 요청을 컴파일러(327)에 전송할 수 있다. 프론트-엔드(303)는 또한, 메모리 공간을 확보(free up)하기 위해, 오래된(outdated) 캐싱된 코드들을 축출(evict)할 수 있다. 프론트-엔드(303)는, 예를 들어, HTTP(Hypertext Transfer Protocol)와 같은 네트워크(105) 내의 통신 프로토콜에 의해 제공되는 표준 버전 확인 프로세스에 기초하여, 캐싱된 직렬화된 코드 및 소스 코드의 버전들을 결정할 수 있다.

[0052] [0056] 캐싱된 직렬화된 코드를 데이터 저장소(313)로부터 획득할 시, 역직렬화 모듈(319)은, 캐싱된 직렬화된 코드의 유효성을 검증하고, 직렬화된 코드 내의 재배치 데이터를 사용하여, 캐싱된 직렬화된 코드를 실행가능 코드 및 레퍼런스들로 역직렬화한다. 역직렬화 모듈(319)은, 실행을 위해 실행가능 코드 및 레퍼런스들을 실행 환경(333)에 전송한다(321로 도시됨). 직렬화된 코드가 소스 코드의 부분들에 대응하면, 소스 코드의 캐싱되지 않은 부분들은 컴파일러(327)에 의해 실행가능 코드로 컴파일될 수 있고, 실행가능 코드는, 실행 환경(333)에서 실행되기에 앞서, 역직렬화된 실행가능 코드와 결합될 수 있다.

[0053] [0057] 직렬화 모듈(309) 및 역직렬화 모듈(319)은 상이한 실행 환경들(331 및 333) 내에서 활성화될 수 있음을 유의한다. 직렬화된 코드가 실행 환경(331)(여기서, 실행가능 코드 및 레퍼런스들이 생성 및 실행됨) 내에서 생성되는 한편, 역직렬화 모듈(319)에 의해 제공되는 실행가능 코드 및 레퍼런스들은 후속 실행 환경(333)에서 생성된다.

[0054] [0058] 데이터 저장소(313) 내의 캐싱된 데이터의 사이즈를 감소시키기 위해, 압축 모듈(307)은, 예를 들어, 데이터 저장소(313)에 직렬화된 코드를 저장하기에 앞서, 직렬화된 코드에 대해 다양한 압축 알고리즘들을 적용함으로써 데이터를 압축할 수 있다. 압축 모듈(307)은 또한, 캐싱된 직렬화된 코드를 역직렬화 모듈(319)에 전송하기에 앞서, 캐싱된 직렬화된 코드를 압축해제할 수 있다.

[0055] [0059] 캐싱된 직렬화된 코드는, 상이한 실행 환경에서 실행가능 코드 및 레퍼런스들의 컨텍스트 독립적인 전체 표현으로서 사용될 수 있다. 예를 들어, 실행 환경(333)에 대한 실행가능 코드 및 레퍼런스들을 생성할 때, 역직렬화 모듈(319)은, 실행 환경(331)에 대한 실행가능 코드 및 레퍼런스들을 생성할 때의 컴파일러(327)의 거동을 모방할 수 있다. 따라서, 캐싱된 직렬화된 코드의 실행가능 코드 및 레퍼런스들로의 역직렬화 이후, 실행 환경(333) 내에서의 실행가능 코드 및 레퍼런스들의 실행은, 실행 환경(331) 내에서 컴파일러(327)에 의해 제공되는 실행가능 코드 및 레퍼런스들의 실행과 동등한 결과들을 제공할 수 있다.

[0056] [0060] 역직렬화 모듈(319)에 의한 역직렬화의 프로세스는, 레퍼런스들이 실행 환경(331)과 실행 환경(333) 간의 차이들을 반영하도록, 메타데이터(315) 내의 재배치 데이터를 사용하여 레퍼런스들을 복원할 수 있다. 예를 들어, 위에 논의된 바와 같이, 직렬화 및 역직렬화가 상이한 실행 환경들(예컨대, VM 인스턴스(instance)들)에서 실행될 수 있으므로, 실행 환경(331)에 대응하는 실행가능 코드 및 레퍼런스들의 메모리 어드레스들은 실행 환경(333)에서 유효하지 않을 수 있다. 그러나, 직렬화 및 역직렬화는, 역직렬화된 실행가능 코드 및 레퍼런스들에서의 메모리 어드레스들이 실행 환경(333) 내에서 유효한 어드레스들임을 보장한다. 예를 들어, 실행 환경에서의 함수 어드레스들에 대한 레퍼런스들은, 이 함수 어드레스들이 상이한 실행 환경들(331 및 333) 간에 상이할 수 있기 때문에, 직렬화 모듈(309)에 의해 축어적으로(verbatim) 인코딩되지 않는다.

[0057] [0061] 직렬화 모듈(309)은, 각각의 오브젝트가 직렬화되어야 하는 방식을 결정하기 위해, 소스 코드 내의 각각의 오브젝트 및 실행가능 코드 및 레퍼런스들을 조사할 수 있다. 예를 들어, 특정한 고유 오브젝트들(예컨대, 정규적(canonical) 미정의된 값)이 추상 인덱스(예컨대, 루트 어레이(root array)) 내의 실행가능 코드 및 레퍼런스들에 포함될 수 있다. 그러한 고유 오브젝트가 직렬화 모듈(309)에 의해 직렬화되는 경우, 직렬화 모듈

(309)은 고유 추상 인덱스에 따라 오브젝트를 직렬화한다. 고유 오브젝트의 역직렬화 모듈(319)에 의한 역직렬화 시, 오브젝트를 식별하기 위해 인덱스가 사용된다.

[0058] [0062] 부가하여, 소스 코드는, 엘리멘트리(elementary) 태스크들을 수행하는 실행가능 코드의 조각들로 생성되는 빌트-인(built-in) 코드를 포함할 수 있다. 빌트-인 코드는, 소스 코드 내의 다수의 오브젝트들에 의해 공유될 수 있고, 따라서, 실행 환경(331)(또는 333) 내에서 항상 이용가능해야 한다. 직렬화 모듈(309)은 각각의 빌트-인 코드를 빌트-인 식별자에 맵핑할 수 있고, 직렬화된 코드 내의 빌트-인 코드를 나타내는 것으로서 빌트-인 식별자를 직렬화한다. 역직렬화 시, 프론트-엔드(303)는 실행 환경(333)에서 동일한 식별자를 갖는 빌트-인 오브젝트를 발견할 수 있다.

[0059] [0063] 유사하게, 소스 코드는 코드-스터브(code-stub)를 포함할 수 있다. 코드-스터브는, 코드-스터브들이 온-디멘드로 생성될 수 있고 실행 환경(333)에서 반드시 이용가능한 것은 아니라는 점을 제외하면, 빌트-인 코드와 유사하다. 직렬화 모듈(309)은, 직렬화된 코드를 생성할 때, 코드-스터브를 코드-스터브 키(key)에 맵핑할 수 있다. 역직렬화 시, 프론트-엔드(303)는, 실행 환경(333)이 코드-스터브를 포함하는지 여부를 체크할 수 있다. 실행 환경(333)이 코드-스터브를 포함하지 않으면, 역직렬화 모듈(319)에 의해, 코드-스터브 키에 대응하는 코드-스터브가 생성된다.

[0060] [0064] 또한, 소스 코드는 고유 스트링 리터럴들을 포함할 수 있다. 고유 스트링 리터럴들은, 직렬화 모듈(309)에 의해 축어적으로 직렬화될 수 있다. 역직렬화 시, 프론트-엔드(303)는, 동일한 스트링 리터럴이 실행 환경(333)에 이미 존재하는지 여부를 체크한다. 고유 스트링 리터럴이 실행 환경(333)에 존재하면, 역직렬화 모듈(319)은, 스트링 리터럴을 이미 존재하는 고유 스트링으로 정규화(canonicalize)한다. 정규화 프로세스는, 하나 초과의 표현(예컨대, 캐싱된 직렬화된 코드의 하나의 표현 및 실행 환경(333)의 제 2 표현)을 갖는 스트링 리터럴을 표준 또는 정규 형태로 변환한다.

[0061] [0065] 몇몇 예시들에서, 직렬화된 오브젝트들은 소스 코드의 소스 스트링에 대한 레퍼런스들을 갖는다. 그러한 예시들에서, 직렬화 모듈(309)은 오브젝트를 특수한 표현으로 대체할 수 있다. 역직렬화 시, 역직렬화 모듈(319)은, 특수한 코드를, 실행 환경(333) 내에서의 역직렬화 시점에 별개로 제공되는 소스 스트링에 대한 레퍼런스로 대체할 수 있다.

[0062] [0066] 위에 논의된 오브젝트 어드레스들의 직렬화에 부가하여, 실행가능 코드에 임베딩된 어드레스들은, 실행 환경에서의 특정 값들의 어드레스들 또는 VM의 함수들의 어드레스들일 수 있다. 실행 환경에서의 특정 값들의 어드레스들 및 VM의 함수들의 어드레스들은 컴파일러 및 VM에 알려져 있다. 이들 어드레스들은, 예를 들어, 레퍼런스 식별자에 의해 표현될 수 있고, 레퍼런스 식별자는, 역직렬화 시에 어드레스를 복원하는데 사용될 수 있다. 또한, 소스 코드는 비-고유 오브젝트들을 포함할 수 있다. 비-고유 오브젝트들은 카피(copy)들로서 직렬화될 수 있다. 예를 들어, 비-고유 오브젝트들은 데이터 필드들 및 다른 오브젝트들에 대한 레퍼런스들로 이루어질 수 있다. 직렬화 시, 데이터 필드들은 카피될 수 있고, 다른 오브젝트들에 대한 레퍼런스들은, 고유하다면, 고유 오브젝트들과 관련하여 앞서 논의된 바와 같이 직렬화될 수 있다.

[0063] [0067] 다음의 예는, 샘플 코드의 컴파일로부터 직렬화 및 역직렬화까지의 코드-캐싱 프로세스를 설명한다. 이 예에서는 샘플 JavaScript 코드(A)가 고려된다.

```
function foo ( ) { return 1; }
foo ( );
```

(A)

[0064]

[0065] 샘플 코드 (A)에 대한 실행가능 코드는, 함수 헤더, 스택 오버플로우(stack overflow)에 대한 체크를 위한 명령들(호출 스택 오버플로우 빌트-인), 변수 선언 "foo", 내부 함수 ( ) { return 1; }에 대한 클로저(closure)의 인스턴스화(instantiation), "foo"라는 이름의 글로벌 속성에 대한 클로저 할당, "foo"라는 이름의 글로벌 속성의 로딩, 로딩된 속성의 호출, 및 미정의된 값의 반환과 같은 명령들을 포함할 수 있다.

[0066] [0068] 부가하여, 샘플 코드 (A)에 대한 실행가능 코드는 재배치 정보에 대한 레퍼런스들을 포함할 수 있으며, 여기서, 재배치 정보는, 스택 오버플로우 빌트-인이 빌트-인이라는 것, 스트링 리터럴 "foo"가 힙(heap) 상의 오브젝트라는 것, ( ) {return 1;}에 대한 함수 설명이 힙 상의 오브젝트라는 것, 함수 클로저 인스턴스화가 런-타임(run-time) 호출로 구현되고 따라서 외부 레퍼런스라는 것, 로딩 속성이 스터브에 대한 호출이라는 것, 호출 속성이 스터브에 대한 호출이라는 것, 및 미정의된 값이 힙 상의 오브젝트(예컨대, 도 3과 관련하여 앞서 논의된 고유 오브젝트)라는 것을 표시할 수 있다.

[0067] [0069] 샘플 코드 (A)에 대한 실행가능 코드는 또한, 예를 들어, "foo"가 맵 오브젝트에 의해 표시되는 "내재화된 스트링(internalized string)" 타입의 오브젝트라는 것, 스트링 컨텐츠가 "foo"라는 것, 및 스트링의 길이가 3이라는 것과 같은, 스트링 리터럴 "foo"와 연관된 메타데이터를 포함할 수 있다. 샘플 코드 (A)에 대한 실행가능 코드는 또한, 최상위-레벨 함수에 대한 함수 설명들 및 내부 함수 ( ) {return 1;}을 포함할 수 있다. 함수 설명들은, 맵 오브젝트에 의해 표시되는 "공유 함수 정보" 타입의 오브젝트들일 수 있다. 함수 설명들은, 소스 코드의 시작 및 끝 지점들에 대한 문자 포지션들을 포함할 수 있다. 함수 설명은 또한, 샘플 코드 (A)가 이미 컴파일된 경우, 컴파일된 코드를 가리킬 수 있다.

[0068] [0070] 처음에, 내부 함수 "foo"는 소극적으로 컴파일된다. 소극적(lazy) 컴파일들은, 최상위-레벨 코드가 컴파일될 때 내부 함수 "foo"는 컴파일되지 않지만 최상위-레벨 코드의 컴파일 동안 내부 함수 "foo"의 설명만이 생성된다는 것을 의미한다. 결국 내부 함수 "foo"가 (예컨대, 최상위-레벨 코드의 실행 동안) 호출될 때, 내부 함수 "foo"는 온-디멘드로 컴파일된다. 그러나, 내부 함수 "foo"가 컴파일되지 않는 한, 내부 함수 "foo"에 대한 실행가능 코드는 존재하지 않으며, 내부 함수에 대한 실행가능 코드는 직렬화될 수 없다.

[0069] [0071] 코드-캐싱 플랫폼(209)이 내부 함수 "foo"에 대한 컴파일된 코드를 캐싱된 데이터에 포함시키기 다른 방식들이 존재한다. 코드-캐싱 플랫폼(209)은, 내부 함수 "foo"가 최상위-레벨 코드 내에서 호출된다는 것을 인식할 수 있다("foo"는 샘플 코드 (A)에서 정의 및 호출됨). 이러한 인식을 가지면, 코드-캐싱 플랫폼(209)은 내부 함수 "foo"를 최상위-레벨 코드를 컴파일 하는 것의 일부로서 적극적으로(eagerly) 컴파일할 수 있고, 내부 함수 "foo"의 컴파일된 코드를 최상위-레벨 코드의 컴파일에 포함시킬 수 있으며, 최상위-레벨 코드의 컴파일 이후, 직렬화하기 위해, 컴파일된 코드를 사용한다.

[0070] [0072] 내부 함수 "foo"에 대한 컴파일된 코드를 직렬화된 코드에 포함시키기 위한 다른 방식은, 코드-캐싱 플랫폼(209)이 직렬화에 앞서 최상위-레벨 코드를 적어도 한 번 실행하는 것이다. 내부 함수 "foo"가 최상위-레벨 코드의 실행 동안 호출되므로, "foo"는 위에 논의된 바와 같이 소극적으로 컴파일될 수 있고, 컴파일된 코드는 메모리에 저장될 수 있다. 이러한 경우에서, 최상위-레벨 코드를 나중에 직렬화하는 경우, (최상위-레벨 코드의 초기 컴파일 동안 생성된) "foo"에 대한 내부 함수 설명은 내부 함수 "foo"의 소극적으로 컴파일된 코드를 가리킨다. 이것은, 코드-캐싱 플랫폼(209)으로 하여금, 최상위-레벨 코드를 직렬화할 때 내부 함수 "foo"를 직렬화하게 할 수 있다.

[0071] [0073] 직렬화 모듈(309)은, 오브젝트 그래프를 방문(visit)하고 레퍼런스들을 거쳐 최상위-레벨 코드에 대한 함수 설명에서 시작함으로써, 샘플 코드 (A)를 직렬화할 수 있다. 코드-캐싱 플랫폼(209)은, 오브젝트가 한 번을 초과하여 직렬화되지 않거나 무한 루프로 실행되지 않도록, 데이터 저장소(313) 내의 이미 방문된 오브젝트들의 이력을 유지할 수 있다. 이미 방문되고 직렬화된 오브젝트들은 백 레퍼런스(back reference)들로 표현(인코딩)될 수 있다. 오브젝트는 컨텐츠 또는 다른 오브젝트들에 대한 레퍼런스들 중 어느 하나를 포함할 수 있다. 오브젝트 컨텐츠들은 축어적으로 직렬화될 수 있다. 그러나, 다른 오브젝트들에 대한 레퍼런스들은, 알려진 오브젝트들에 대한 레퍼런스들 또는 또한 직렬화될 필요가 있는 오브젝트들에 대한 레퍼런스들 중 어느 하나일 수 있다. 예를 들어, 함수 설명은, "공유 함수 정보" 맵 및 함수에 대한 컴파일된 코드에 대한 레퍼런스를 가질 수 있다. 함수 설명은 또한, 축어적으로 직렬화된 문자 포지션들에 대한 정보를 포함할 수 있다. 직렬화 모듈(309)에 의해 인카운터되는 맵 오브젝트들은, 정규적 오브젝트들 및 루트 리스트의 일부일 수 있다. 직렬화 모듈(309)은, 루트 리스트에서 맵 오브젝트들을 발견하고, 오브젝트들을 표현하기 위해 루트 리스트 인덱스를 사용할 수 있다.

[0072] [0074] 최상위-레벨 함수에 대한 코드 오브젝트는, (코드 오브젝트들에 대한) 그의 맵에 대한 레퍼런스들 및 재배치 정보에 대한 레퍼런스를 제외하고는 축어적으로 직렬화될 수 있다. 명령 스트림은, 재배치 정보에 의해 설명되는, 임베딩된 포인터(pointer)들을 포함할 수 있다. 이들 포인터들은 또한, 직렬화 동안 방문된다. 포인터들이 역직렬화 시에 업데이트될 필요가 있으므로, 포인터들은 축어적으로 직렬화되기 전에 제로(0) 값들로 대체될 수 있다.

[0073] [0075] 최상위-레벨 함수의 코드 오브젝트는, 내부 함수 "foo"의 함수 설명을 가리킨다. 내부 함수 "foo"가 이 시점에 이미 컴파일된 경우, 이의 함수 설명은, 직렬화 모듈(309)이 최상위-레벨 함수로부터 내부 함수 "foo"까지를 거쳐 최상위-레벨 및 내부 함수들 둘 모두의 코드 오브젝트들을 직렬화할 수 있도록, 컴파일된 코드를 가리킨다. 부가하여, 코드 스터브들은 코드 스터브 키에 의해 표현될 수 있고, 빌트-인들은 빌트-인 ID들에 의해 표현될 수 있다. 스트링 컨텐츠는 (스트링 맵을 제외하고는) 축어적으로 직렬화될 수 있고, 외부 레퍼런스들은 외부 레퍼런스 ID들에 맵핑될 수 있다.

[0074] [0076] 역직렬화 모듈(319)에 의한 역직렬화의 프로세스는, 오브젝트 그래프를 방문함으로써 직렬화가 수행된 순서를 리트레이싱(retracing)하는 것을 포함할 수 있다. 이러한 트레이싱(tracing)은, 직렬화된 코드 표현들을, 역직렬화된 코드가 실행될 수 있는 실행 환경(333)에서의 대응부(counterpart)들로 다시 변환하는데 사용될 수 있다. 예를 들어, 미정의된 값이 직렬화 동안 루트 리스트 인덱스로서 인코딩된 경우, 역직렬화 모듈(319)은, 실행 환경(333)에서 미정의된 값을 발견하기 위해 루트 리스트 인덱스를 사용할 수 있다. 부가하여, 백 레퍼런스들은, 이미 역직렬화된 오브젝트들에 대한 레퍼런스들로 다시 변환될 수 있다. 역직렬화 결과는 최상위-레벨 함수에 대한 함수 설명일 수 있다.

[0075] [0077] 도 4a-4b는, 코드의 캐싱된 코드로의 직렬화가 제공될 수 있는 프로세스들의 예들을 예시한다. 도 4a-4b가 도 1, 도 2, 및 도 3을 참조하여 설명되지만, 본 기술은 그렇게 제한되지는 않으며, 다른 클라이언트 디바이스들 및 시스템들에 적용될 수 있다. 도 4a는 코드를 캐싱된 코드로 직렬화하기 위한 프로세스의 예를 예시한다. 블록(401)에서, 프론트-엔드(303)는, 실행을 대기하는 1차 소스 코드의 표시(예컨대, 링크, 어드레스, URL 등)를 수신한다. 예를 들어, 1차 소스 코드는 현재 브라우저 세션에서의 실행을 위한 웹 페이지 내의 스크립트일 수 있다. 도 1의 코드-캐싱 플랫폼(103a 또는 103n)(또는 도 2의 코드-캐싱 플랫폼(209))을 호스팅하는 클라이언트 디바이스(101a 또는 101n)의 사용자는, 현재 브라우저 세션에서 URL 어드레스를 입력함으로써 1차 소스 코드의 실행을 요청할 수 있다. URL 어드레스는, 임베딩된 1차 소스 코드를 포함하는, 웹 페이지를 로딩하기 위한 링크를 제공할 수 있다.

[0076] [0078] 웹 페이지를 로딩할 시, 브라우저는, 1차 소스 코드의 표시를 프론트-엔드(303)에 전송할 수 있다. 표시를 수신하는 것에 대한 응답으로, 블록(403)에서, 체크 모듈(305)은, 1차 소스 코드에 대응하는 캐싱된 데이터에 대해 데이터 저장소(313)의 리소스 캐시를 체크한다. 예를 들어, 웹 페이지는 현재 브라우저 세션 이전의 다른 브라우저 세션에서 로딩되었을 수 있고, 코드-캐싱 플랫폼(209)은 1차 소스 코드에 대응하는 실행가능 코드를 데이터 저장소(313)에 저장했을 수 있다. 데이터 저장소(313)가 1차 소스 코드에 대응하는 실행가능 코드를 포함하면, 블록(403)의 체크는 캐시 히트를 반환할 수 있다. 그러나, 블록(403)에서의 체크가 히트를 발견하지 못하면, 반환되는 결과는 캐시 미스일 수 있다(블록(405)으로 도시됨).

[0077] [0079] 블록(407)에서, 데이터 저장소(313)에서의 캐시 미스 시, 프론트-엔드(303)는, 1차 소스 코드로부터 컴파일된 실행가능 코드를 획득할 수 있다. 예를 들어, 프론트-엔드(303)는, 컴파일을 위해 1차 소스 코드를 컴파일러(327)에 전송할 수 있다.

[0078] [0080] 블록(409)에서, 프론트-엔드(303)는, 1차 소스 코드에서 참조되는 소스 코드의 부분들 또는 2차 소스 코드를 선택할 수 있다. 예를 들어, 2차 소스 코드는 1차 소스 코드에서 참조되는 함수일 수 있다. 소스 코드의 부분들의 선택은, 도 2 및 도 3과 관련하여 논의된 다양한 팩터들 및 기준들(경험칙들)에 기초할 수 있다. 블록(411)에서, 프론트-엔드(303)는, 2차 소스 코드로부터 컴파일된 실행가능 코드를 획득할 수 있다. 예를 들어, 프론트-엔드(303)는, 컴파일을 위해 2차 소스 코드를 컴파일러(327)에 전송할 수 있다. 몇몇 예시들에서, 프론트-엔드(303)는, 1차 소스 코드 및 1차 소스 코드에서 참조되는 2차 소스 코드의 세트를 포함하는 전체 소스 코드를 부분들을 선택함이 없이 컴파일러에 전송할 수 있다. 예를 들어, 캐시 미스를 수신할 시, 프론트-엔드(303)는 소스 코드 및/또는 소스 코드에서 참조되는 선택된 2차 소스 코드들을 컴파일러(327)에 전송할 수 있다.

[0079] [0081] 컴파일러(327)는 수신된 1차 소스 코드 또는 2차 소스 코드를 컴파일하고, 실행가능 코드를 직렬화 모듈(309)에 전송한다. 컴파일러(327)는, 실행가능 코드를 클라이언트 디바이스(101a 또는 101n)의 메모리에 저장할 수 있다. 실행가능 코드는, 소스 코드 또는 선택된 부분들의 오브젝트들에 대응하는 어드레스들 및 식별자들과 같은 레퍼런스들의 세트를 포함한다. 실행가능 코드에 대응하는 레퍼런스들의 세트는, 실행 환경(331)에 대응하는, 실행가능 코드에 임베딩된 메모리 어드레스들을 포함할 수 있다. 실행 환경들(331)(또는 333)은, 소스 코드가 실행되는 웹 브라우저 세션 내의 VM들일 수 있다. 레퍼런스들은, 실행 환경(331), 예컨대 실행가능 코드에 대한 실행 엔진(도시되지 않음)에 의해 생성된 VM 환경에서 실행가능 코드를 실행할 때, 실행가능 코드(329)에 대한 실행 데이터를 프로세서(201)에 제공한다.

[0080] [0082] 컴파일러(327)에 의해 실행가능 코드를 수신할 시, 블록(413)에서, 직렬화 모듈(309)은 실행가능 코드는 직렬화한다. 직렬화는, 도 2 및 도 3과 관련하여 앞서 논의된 바와 같이, 실행 환경(331)으로부터의 실행가능 코드 내의 레퍼런스들의 세트를 추상화하는 것 및 직렬화된 코드와 연관된 재배치 데이터를 추상화들을 반영하도록 정의하는 것을 포함한다. 직렬화 모듈(309)에 의한 실행 환경(331)으로부터의 레퍼런스들의 세트의 추상화는, 임베딩된 메모리 어드레스들을 추상 어드레스들로 대체하는 것을 포함할 수 있다. 예를 들어, 임베딩된

어드레스는, 미리정의된 함수들의 리스트의 특정 인덱스 내로의 VM의 함수로 변환될 수 있다. 몇몇 예시들에서, 동일한 코드를 가리키는 어드레스를 실행가능 코드가 포함하면, 실행가능 코드에 할당된 메모리 블록의 시작 어드레스로부터의 오프셋(offset) 어드레스로 메모리 어드레스를 대체함으로써 추상 어드레스가 생성될 수 있고, 임베딩된 어드레스는 코드의 시작으로부터의 상대적 오프셋으로 변환될 수 있다.

[0081] [0083] 블록(415)에서, 프론트-엔드(303)는, 직렬화된 코드를, 실행 환경(331)에서의 실행을 위해 제공될 캐싱된 데이터로서 데이터 저장소(313)의 리소스 캐시에 저장한다. 몇몇 예시들에서, 직렬화 모듈(309)에 의해 실행가능 코드를 직렬화하기에 앞서, 프론트-엔드(303)는, 직렬화를 수행하기 위한 프로세서(201)의 가용성을 결정할 수 있다. 이러한 예시들에서, 프로세서가 이용가능하지 않으면, 프론트-엔드(303)는, 프로세서가 이용가능해질 때까지 직렬화를 지연시킬 수 있다.

[0082] [0084] 소스 코드는, 다수의 2차 소스 코드들(예컨대, 함수들)에 대한 레퍼런스들을 포함할 수 있다. 이러한 경우들에서, 실행가능 코드는, 각각의 실행가능 2차 소스 코드가 소스 코드에서 참조되는 2차 소스 코드에 대응하도록, 다수의 실행가능 코드 2차 소스 코드들을 포함할 수 있다. 프론트-엔드(303)는, 직렬화를 위해, 실행가능 2차 소스 코드를 선택할 수 있다. 프론트-엔드(303)는 또한, 실행가능 2차 소스 코드가 직렬화된 코드 내에서 비-직렬화된 채로 유지될 것을 결정할 수 있다. 프론트-엔드(303)는, 실행가능 코드와 연관된 레퍼런스들, 재배치 데이터, 2차 소스 코드들의 타입, 2차 소스 코드들의 보안 식별자 등에 기초하여, 2차 소스 코드의 역직렬화를 결정할 수 있다. 소스 코드는 애플리케이션(207)의 최상위-레벨 스크립트일 수 있다. 프론트-엔드(303)는, 예를 들어, 미리정의된 조건들에 기초하여, 직렬화를 위해 애플리케이션(207)의 최상위-레벨 스크립트를 소스 코드로서 선택할 수 있다.

[0083] [0085] 몇몇 예시들에서, 프론트-엔드(303)는, 실행을 대기하는 1차 소스 코드의 표시(예컨대, 링크, 어드레스, URL 등)를 수신할 수 있다. 예를 들어, 1차 소스 코드는 웹 페이지 내의 스크립트일 수 있다. 표시를 수신하는 것에 대한 응답으로, 도 4a의 블록(403)과 관련하여 논의된 바와 같이, 체크 모듈(305)은, 1차 소스 코드 및 도 4a의 블록(409)에서 선택된 2차 소스 코드에 대응하는 캐싱된 데이터에 대해, 데이터 저장소(313)의 리소스 캐시를 체크한다. 데이터 저장소(313)의 리소스 캐시에서의 캐시 히트 시, 프론트-엔드(303)는, 직렬화된 코드를 포함하는 캐싱된 데이터를 데이터 저장소(313)의 리소스 캐시로부터 리트리브할 수 있다.

[0084] [0086] 역직렬화 모듈(319)은, 리트리브된 캐싱된 직렬화된 코드를 실행가능 코드로 역직렬화할 수 있다. 역직렬화 모듈(319)에 의한 역직렬화는, 리트리브된 캐싱된 직렬화된 코드를 사용하여, 실행가능 코드 내의 레퍼런스들의 세트를 실행 환경(333)에 기초하여 복원하는 것을 포함할 수 있다. 프론트-엔드(303)는, 실행 환경(333)에서의 실행을 위해 실행가능 코드 및 레퍼런스들을 제공할 수 있다.

[0085] [0087] 몇몇 예시들에서, 역직렬화 모듈(319)에 의한 역직렬화는, 예를 들어, 데이터 변질(corruption)로 인해 실패할 수 있음을 유의한다. 그러한 예시들에서, 프론트-엔드(303)는, 캐싱된 직렬화된 코드를 데이터 저장소(313)로부터 삭제할 수 있다. 따라서, 실행을 대기하는 소스 코드의 표시가 다음에 수신될 때, 도 4a의 블록(403)에 도시된 체크 모듈(305)에 의한 체크는 캐시 미스를 반환할 수 있고, 도 4a에 따라 직렬화 프로세스가 수행될 수 있다.

[0086] [0088] 몇몇 예시들에서, 블록(409)의 2차 소스 코드의 선택은, 블록(407)에서 획득된 실행가능 코드의 실행으로부터 획득되는 실행 결과들에 기초할 수 있다. 예를 들어, 블록(407)에서 획득되는 제 1 실행가능 코드는 2차 코드의 선택 이전에 한 번 또는 다수 회 실행될 수 있다. 그러한 예시들에서, 코드-캐싱 플랫폼(209)은, 제 1 실행 컨텍스트에서의 제 1 실행가능 코드의 실행으로부터의 실행 결과들을 획득하고, 실행 결과들에 기초하여 2차 소스 코드를 선택할 수 있다. 제 1 실행가능 코드의 실행 시, 코드-캐싱 플랫폼(209)은, 참조 및 실행되는 2차 소스 코드(예컨대, 함수들)를 결정하기 위해, 실행에 관련된 데이터를 수집할 수 있다.

[0087] [0089] 코드-캐싱 플랫폼(209)은, 전체 소스 코드를 캐싱하는 대신, 캐싱을 위한, 실행된 2차 소스 코드를 선택할 수 있다. 그 이유는, 1차 소스 코드 내의 다양한 조건들, 클라이언트 디바이스(200) 상의 실행 환경에 관련된 조건들, 사용자 입력 등에 따라, 일부 2차 소스 코드들(예컨대, 함수들)은 실행될 수 있고 일부는 실행되지 않을 수 있기 때문이다. 코드-캐싱 플랫폼(209)은, 클라이언트 디바이스(200) 상에서의 1차 소스 코드의 다수의 실행들에 관련된 데이터를 수집함으로써, 실행되는 2차 소스 코드들을 결정하고, 다른 2차 소스 코드들보다 더 자주 실행되는 2차 소스 코드 코드들을 결정할 수 있다. 코드-캐싱 플랫폼(209)은, 결정에 기초하여, 캐싱을 위한 2차 소스 코드를 선택할 수 있다. 코드-캐싱 플랫폼(209)은 또한, 각각의 함수 호출이 함수의 실행을 포함하는 1차 소스 코드 내의 함수 호출들을 결정할 수 있다. 코드-캐싱 플랫폼(209)은, 함수 호출들의 실행에 관한 정보를 수집하고, 캐싱을 위한 실행 동안 실행되는 함수들을 선택할 수 있다.

[0088] [0090] 코드-캐싱 플랫폼(209)은, 예를 들어, 2차 소스 코드의 사이즈, 2차 실행가능 코드의 사이즈, 2차 소스 코드가 1차 소스 코드에서 참조되는 횟수, 2차 소스 코드의 컴파일 시간 등과 같은 다양한 다른 팩터들에 기초하여 2차 소스 코드를 선택할 수 있다.

[0089] [0091] 도 4b는 조건들의 세트에 기초하여 코드를 캐싱된 코드로 직렬화하기 위한 프로세스의 예를 예시한다. 블록(421)에서, 프론트-엔드(303)는, 실행을 대기하는 1차 소스 코드의 표시(예컨대, 링크, 어드레스, URL 등)를 수신한다. 도 4a와 관련하여 앞서 논의된 바와 같이, 표시를 수신하는 것에 대한 응답으로, 블록(423)에서, 체크 모듈(305)은, 1차 소스 코드에 대응하는 캐싱된 데이터에 대해 데이터 저장소(313)의 리소스 캐시를 체크한다. 블록(423)에서의 체크가 히트를 발견하지 못하면, 반환되는 결과는 캐시 미스일 수 있다(블록(425)으로 도시됨).

[0090] [0092] 블록(427)에서, 데이터 저장소(313)에서의 캐시 미스 시, 프론트-엔드(303)는, 1차 소스 코드로부터 컴파일된 실행가능 코드를 획득할 수 있다. 예를 들어, 프론트-엔드(303)는, 컴파일을 위해 1차 소스 코드를 컴파일러(327)에 전송할 수 있다.

[0091] [0093] 블록(429)에서, 직렬화 모듈(309)은, 1차 소스 코드의 사이즈, 미리결정된 시간 기간 내에 1차 소스 코드가 실행되는 횟수 또는 빈도, 1차 소스 코드의 컴파일 시간, 또는 이들의 결합에 기초하여, 실행가능 코드를 직렬화된 코드로 직렬화한다. 예를 들어, 1차 소스 코드가 일주일 동안 10회 미만으로 실행되면, 직렬화 모듈(309)은, 1차 소스 코드를 직렬화하는 것을 삼갈(forego) 수 있다. 그러나, 더 자주(예컨대, 일주일 동안 10회를 초과하여) 실행되는 1차 소스 코드는 직렬화될 수 있다. 블록(431)에서, 프론트-엔드(303)는, 직렬화된 코드를, 실행 환경(331 또는 333)에서의 실행을 위해 제공될 캐싱된 데이터로서 데이터 저장소(313)의 리소스 캐시에 저장한다.

[0092] [0094] 위에 기재된 바와 같이, 본 기술의 양상들은, 데이터가 일 웹 브라우징 세션에서 직렬화되고 다른 웹 브라우저 세션에서 역직렬화되는 웹 브라우저들의 인스턴스들과 같은 상이한 실행 환경들에서의 실행을 위한 실행가능 코드를 캐싱하는 관점들에서 설명되었다. 그러나, 본 기술은 웹 브라우저 환경으로 제한되지 않는다. 본 기술은, 임의의 실행 환경에서 실행가능 코드를 캐싱하는 것에 적용가능할 수 있다. 부가하여, 본 기술은, 도 1의 제 1 클라이언트 디바이스(101a-101n)(또는 도 1의 서버(109a-109m))에서 데이터를 직렬화 및 캐싱하고, 캐싱된 데이터가 제 2 클라이언트 디바이스(101a-101n)(또는 서버(109a-109m))에서 역직렬화 및 실행되도록, 캐싱된 데이터를 제 2 클라이언트 디바이스(101a-101n)(또는 서버(109a-109m))에 전송하는 것에 적용가능할 수 있다.

[0093] [0095] 위에서 설명된 특성들 및 애플리케이션들은 컴퓨터 판독가능 저장 매체(컴퓨터 판독가능 매체로도 지칭됨) 상에 기록되는 명령들의 세트로서 특정되는 소프트웨어 프로세스들로서 구현될 수 있다. 이러한 명령들이 프로세싱 유닛(들)(예컨대, 프로세서들, 프로세서들의 코어들, 또는 다른 프로세싱 유닛들)에 의해 실행되는 경우, 그 명령들은 프로세싱 유닛(들)으로 하여금 명령들에 표시된 동작들을 수행하게 한다. 컴퓨터 판독가능 매체들의 예들은 CD-ROM들, 플래시 드라이브들, RAM 칩들, 하드 드라이브들, EPROM들 등을 포함(그러나 이들로 제한되지 않음)한다. 컴퓨터 판독가능 매체들은 무선으로 또는 유선 연결들을 통해 전달하는 캐리어 파들 및 전자 신호들을 포함하지 않는다.

[0094] [0096] 본 명세서에서, "소프트웨어"란 용어는 판독-전용 메모리에 상주하는 펌웨어, 또는 프로세서에 의한 프로세싱을 위해 메모리로 판독될 수 있는 자기 저장부에 저장된 애플리케이션들을 포함하도록 의도된다. 부가하여, 몇몇 구현들에서, 별개의 소프트웨어 기술들을 유지하면서 다수의 소프트웨어 기술들이 더 큰 프로그램의 서브-부분들로서 구현될 수 있다. 몇몇 구현들에서, 다수의 소프트웨어 기술들은 또한 개별 프로그램들로 구현될 수 있다. 마지막으로, 여기서 설명된 소프트웨어 기술을 함께 구현하는 개별 프로그램들의 임의의 결합은 본 기술의 범위 내에 있다. 몇몇 구현들에서, 소프트웨어 프로그램들은, 전자 시스템들에서 동작하도록 설치되는 경우, 소프트웨어 프로그램들의 동작들을 실행하고 수행하는 특정 머신 구현들을 정의한다.

[0095] [0097] 컴퓨터 프로그램(소스 코드, 프로그램, 소프트웨어, 소프트웨어 애플리케이션, 스크립트 또는 코드로도 알려져 있음)은 컴파일되거나 해석되는 언어들, 선언형(declarative) 또는 절차형(procedural) 언어들을 비롯한 임의의 형태의 프로그래밍 언어로 기입될 수 있다. 부가하여, 컴퓨터 프로그램은 독립형 프로그램이나 또는 모듈, 컴포넌트, 서브루틴, 오브젝트 또는 컴퓨팅 환경에서 사용하기에 적합한 다른 유닛을 비롯한 임의의 형태로 전개될 수 있다. 컴퓨터 프로그램은 파일 시스템의 파일에 대응할 수 있지만 대응할 필요는 없을 수 있다. 프로그램은 다른 프로그램들 또는 데이터(예컨대, 마크업(markup) 언어 문서에 저장되는 스크립트들)를 보유하는 파일의 부분에, 해당 프로그램에 전용화된 단일 파일에, 또는 다수의 통합형 파일들(예컨대, 모듈들, 서브 프로

그램들 또는 코드의 부분들을 저장하는 파일들)에 저장될 수 있다. 컴퓨터 프로그램은 하나의 사이트에 위치되거나 다수의 사이트들에 걸쳐 분산되어 통신 네트워크에 의해 상호연결되는 다수의 컴퓨터들 상에서 실행되거나 또는 하나의 컴퓨터 상에서 실행되도록 전개될 수 있다.

[0096] [0098] 도 5는, 본 기술의 몇몇 구현들이 구현될 수 있는 예시적인 전자 시스템을 개념적으로 예시한다. 전자 시스템(500)은 컴퓨터, 폰, PDA, 또는 임의의 다른 종류의 전자 디바이스일 수 있다. 이러한 전자 시스템은 다양한 타입들의 컴퓨터 판독가능 매체들 및 다양한 다른 타입들의 컴퓨터 판독가능 매체들에 대한 인터페이스들을 포함한다. 전자 시스템(500)은 버스(508), 프로세싱 유닛(들)(512), 시스템 메모리(504), 판독-전용 메모리(ROM)(510), 영구 저장 디바이스(502), 입력 디바이스 인터페이스(514), 출력 디바이스 인터페이스(506), 및 네트워크 인터페이스(516)를 포함한다.

[0097] [0099] 버스(508)는 전자 시스템(500)의 다수의 내부 디바이스들을 통신가능하게 연결하는 모든 시스템, 주변장치 및 칩셋 버스들을 총괄적으로 나타낸다. 예를 들면, 버스(508)는 프로세싱 유닛(들)(512)을 ROM(510), 시스템 메모리(504), 및 영구 저장 디바이스(502)와 통신가능하게 연결한다.

[0098] [0100] 이러한 다양한 메모리 유닛들로부터, 프로세싱 유닛(들)(512)은 본 개시내용의 프로세스들을 실행하기 위해, 실행할 명령들 및 프로세싱할 데이터를 리트리브한다. 프로세싱 유닛(들)은 상이한 구현들에서 단일 프로세서 또는 멀티-코어 프로세서일 수 있다.

[0099] [0101] ROM(510)은 전자 시스템의 프로세싱 유닛(들)(512) 및 다른 모듈들에 의해 요구되는 정적 데이터 및 명령들을 저장한다. 반면에, 영구 저장 디바이스(502)는 판독-및-기입 메모리 디바이스이다. 이러한 디바이스는, 전자 시스템(500)이 오프(off)일 때조차 명령들 및 데이터를 저장하는 비-휘발성 메모리 유닛이다. 본 개시내용의 일부 구현들은 영구 저장 디바이스(502)로서 대용량-저장 디바이스(예컨대, 자기 또는 광학 디스크 및 그것의 대응하는 디스크 드라이브)를 사용한다.

[0100] [0102] 다른 구현들은 영구 저장 디바이스(502)로서 착탈식 저장 디바이스(예컨대, 플로피 디스크, 플래시 드라이브 및 그것의 대응하는 디스크 드라이브)를 사용한다. 영구 저장 디바이스(502)와 같이, 시스템 메모리(504)는 판독-및-기입 메모리 디바이스이다. 그러나, 저장 디바이스(502)와 달리, 시스템 메모리(504)는 랜덤 액세스 메모리와 같은 휘발성 판독-및-기입 메모리이다. 시스템 메모리(504)는 프로세서가 런-타임에 필요로 하는 명령들 및 데이터의 일부를 저장한다. 몇몇 구현들에서, 본 개시내용의 프로세스들은 시스템 메모리(504), 영구 저장 디바이스(502), 또는 ROM(510)에 저장된다. 예를 들어, 다양한 메모리 유닛들은 몇몇 구현들에 따라 웹 엘리먼트들을 제공하기 위한 명령들을 포함한다. 이러한 다양한 메모리 유닛들로부터, 프로세싱 유닛(들)(512)은 일부 구현들의 프로세스들을 실행하기 위해, 실행할 명령들 및 프로세싱할 데이터를 리트리브한다.

[0101] [0103] 버스(508)는 또한 입력 및 출력 디바이스 인터페이스들(514 및 506)에 연결된다. 입력 디바이스 인터페이스(514)는 사용자가 정보 및 선택 커맨드들을 전자 시스템에 통신하는 것을 가능하게 한다. 입력 디바이스 인터페이스(514)와 함께 사용되는 입력 디바이스들은, 예를 들어, 알파벳 숫자 키보드들 및 포인팅 디바이스들(또한 "커서 제어 디바이스들"로 지칭됨)을 포함한다. 출력 디바이스 인터페이스들(506)은, 예를 들어, 전자 시스템(500)에 의해 생성되는 이미지들의 디스플레이를 가능하게 한다. 출력 디바이스 인터페이스(506)와 함께 사용되는 출력 디바이스들은, 예를 들어, 프린터들 및 디스플레이 디바이스들, 예컨대 CRT(cathode ray tubes) 또는 LCD(liquid crystal display)들을 포함한다. 몇몇 구현들은, 디바이스들, 예컨대 입력 및 출력 디바이스들 양쪽 모두로서 기능하는 터치스크린을 포함한다.

[0102] [0104] 마지막으로, 도 5에 도시된 바와 같이, 버스(508)는 또한 네트워크 인터페이스(516)를 통해 전자 시스템(500)을 네트워크(도시되지 않음)에 커플링한다. 이러한 방식으로, 컴퓨터는 컴퓨터들의 네트워크(예컨대, "LAN(local area network)", "WAN(wide area network)" 또는 인트라넷 또는 네트워크들의 네트워크, 예컨대 인터넷)의 부분일 수 있다. 전자 시스템(500)의 임의의 또는 모든 컴포넌트들은 본 개시내용과 함께 사용될 수 있다.

[0103] [0105] 위에서 설명된 이러한 기능들은 디지털 전자 회로에서, 컴퓨터 소프트웨어, 펌웨어 또는 하드웨어에서 구현될 수 있다. 기술들은 컴퓨터 프로그램 제품들을 사용하여 구현될 수 있다. 프로그래밍가능 프로세서들 및 컴퓨터들은 모바일 디바이스들에 포함되거나 또는 모바일 디바이스들로서 패키징될 수 있다. 프로세스들 및 로직 흐름들은 프로그래밍가능 프로세서들에 의해 그리고 프로그래밍가능 로직 회로에 의해 수행될 수 있다. 범용 및 특수 목적 클라이언트 디바이스들 및 저장 디바이스들이 통신 네트워크들을 통해 상호연결될 수 있다.

[0104] [0106] 몇몇 구현들은 전자 컴포넌트들, 예를 들어, 머신-판독가능 또는 컴퓨터-판독가능 매체(대안적으로는 컴

퓨터-판독가능 저장 매체들, 머신-판독가능 매체들 또는 머신-판독가능 저장 매체들로 지칭됨)에 컴퓨터 프로그램 명령들을 저장하는 메모리, 저장부 및 마이크로프로세서들을 포함한다. 그러한 컴퓨터-판독가능 매체들의 몇몇 예들은 RAM, ROM, CD-ROM(read-only compact discs), CD-R(recordable compact discs), CD-RW(rewritable compact discs), 판독-전용 디지털 다기능 디스크들(예컨대, DVD-ROM, 듀얼-계층 DVD-ROM), 다양한 기록가능/재기입가능 DVD들(예컨대, DVD-RAM, DVD-RW, DVD+RW 등), 플래시 메모리(예컨대, SD 카드들, 미니-SD 카드들, 마이크로-SD 카드들 등), 자기 또는 고체 상태 하드 드라이브들, 판독-전용 및 기록가능 Blu-Ray ® 디스크들, 초 고밀도(ultra density) 광학 디스크들, 임의의 다른 광학 또는 자기 매체들, 및 플로피 디스크들을 포함한다. 컴퓨터-판독가능 매체들은, 적어도 하나의 프로세싱 유닛에 의해 실행가능하고 다양한 동작들을 수행하기 위한 명령들의 세트들을 포함하는 컴퓨터 프로그램을 저장할 수 있다. 컴퓨터 프로그램들 또는 컴퓨터 코드의 예들은, 예를 들어, 컴파일러에 의해 생성되는 머신 코드, 및 인터프리터를 사용하여 컴퓨터, 전자 컴포넌트 또는 마이크로프로세서에 의해 실행되는 상위-레벨(higher-level) 코드를 포함하는 파일들을 포함한다.

[0105] [0107] 위의 논의가 소프트웨어를 실행하는 마이크로프로세서 또는 다중-코어 프로세서들을 주로 참조하지만, 몇몇 구현들은 집적 회로들, 예를 들어 ASIC(application specific integrated circuit)들 또는 FPGA(field programmable gate array)들에 의해 수행된다. 몇몇 구현들에서, 그러한 집적 회로들은 그 회로 자체에 저장되는 명령들을 실행한다.

[0106] [0108] 본 명세서에서 그리고 본 출원의 임의의 청구항들에서 사용되는 바와 같이, "컴퓨터", "서버", "프로세서" 및 "메모리"라는 용어들 전부는 전자 또는 다른 기술적 디바이스들을 지칭한다. 이러한 용어들은 사람들 또는 사람들의 그룹들을 배제한다. 본 명세서의 목적들을 위해, "디스플레이" 또는 "디스플레이하는"이란 용어들은 전자 디바이스 상에 디스플레이하는 것을 의미한다. 본 명세서에서 그리고 본 출원의 임의의 청구항들에서 사용되는 바와 같이, "컴퓨터 판독가능 매체" 및 "컴퓨터 판독가능 매체들"이란 용어들은 전적으로 컴퓨터에 의해 판독가능한 형태로 정보를 저장하는 유형의(tangible) 물리적 오브젝트들로 제약된다. 이러한 용어들은 임의의 무선 신호들, 유선 다운로드 신호들 및 임의의 다른 단기성(ephemeral) 신호들을 배제한다.

[0107] [0109] 사용자와의 상호작용을 제공하기 위해, 본 명세서에 설명된 발명의 대상의 구현들은 사용자에게 정보를 디스플레이하기 위한 디스플레이 디바이스, 예를 들어 CRT(cathode ray tube) 또는 LCD(liquid crystal display) 모니터, 및 사용자가 컴퓨터에 입력을 제공할 수 있게 하는 키보드 및 포인팅 디바이스, 예를 들어 마우스 또는 트랙볼을 갖는 컴퓨터 상에서 구현될 수 있다. 사용자와의 상호작용을 제공하기 위해 다른 종류들의 디바이스들이 또한 사용될 수 있는데; 예를 들어, 사용자에게 제공되는 피드백은 임의의 형태의 감지 피드백, 예를 들어 시각적 피드백, 청각적 피드백, 또는 촉각적 피드백일 수 있고; 사용자로부터의 입력은 청각적, 음성, 또는 촉각적 입력을 비롯한 임의의 형태로 수신될 수 있다. 부가하여, 컴퓨터는 사용자에 의해 사용되는 디바이스에 문서들을 전송하고 디바이스로부터 문서들을 수신함으로써, 예컨대, 웹 브라우저로부터 수신된 요청들에 대한 응답으로 사용자의 클라이언트 디바이스 상의 웹 브라우저에 웹 페이지들을 전송함으로써, 사용자와 상호작용할 수 있다.

[0108] [0110] 본 명세서에 설명된 발명의 대상의 구현들은 컴퓨팅 시스템에서 구현될 수 있는데, 컴퓨팅 시스템은, 예를 들어 데이터 서버로서 백 엔드(back end) 컴포넌트를 포함하거나, 예컨대 애플리케이션 서버로서 미들웨어(middleware) 컴포넌트를 포함하거나, 또는 프론트-엔드 컴포넌트, 예를 들어, 본 명세서에 설명된 발명의 대상의 구현과 사용자가 상호작용할 수 있게 하는 웹 브라우저 또는 그래픽 사용자 인터페이스를 갖는 클라이언트 컴퓨터를 포함하거나, 또는 이러한 백 엔드, 미들웨어 또는 프론트 엔드 컴포넌트들의 임의의 결합을 포함한다. 시스템의 컴포넌트들은, 임의의 형태 또는 매체의 디지털 데이터 통신, 예컨대 통신 네트워크에 의해 상호연결될 수 있다. 통신 네트워크들의 예들은 "LAN(local area network)", "WAN(wide area network)", 인터-네트워크(예를 들어, 인터넷), 및 피어-투-피어 네트워크들(예컨대, 애드 혹 피어-투-피어 네트워크들)을 포함한다.

[0109] [0111] 컴퓨팅 시스템은 클라이언트들 및 서버들을 포함할 수 있다. 클라이언트 및 서버는 일반적으로 서로 떨어져 있으며, 통신 네트워크를 통해 상호작용할 수 있다. 클라이언트와 서버의 관계는, 각각의 컴퓨터들 상에서 실행되고 서로에 대해 클라이언트-서버 관계를 갖는 컴퓨터 프로그램들에 의해 발생한다. 몇몇 구현들에서, 서버는 (예컨대, 클라이언트 디바이스와 상호작용하는 사용자에게 데이터를 디스플레이하고, 클라이언트 디바이스와 상호작용하는 사용자로부터 사용자 입력을 수신하는 목적들을 위해) 데이터(예컨대, HTML 페이지)를 클라이언트 디바이스에 송신한다. 클라이언트 디바이스에서 생성된 데이터(예컨대, 사용자 상호작용의 결과)는 클라이언트 디바이스로부터 서버에서 수신될 수 있다.

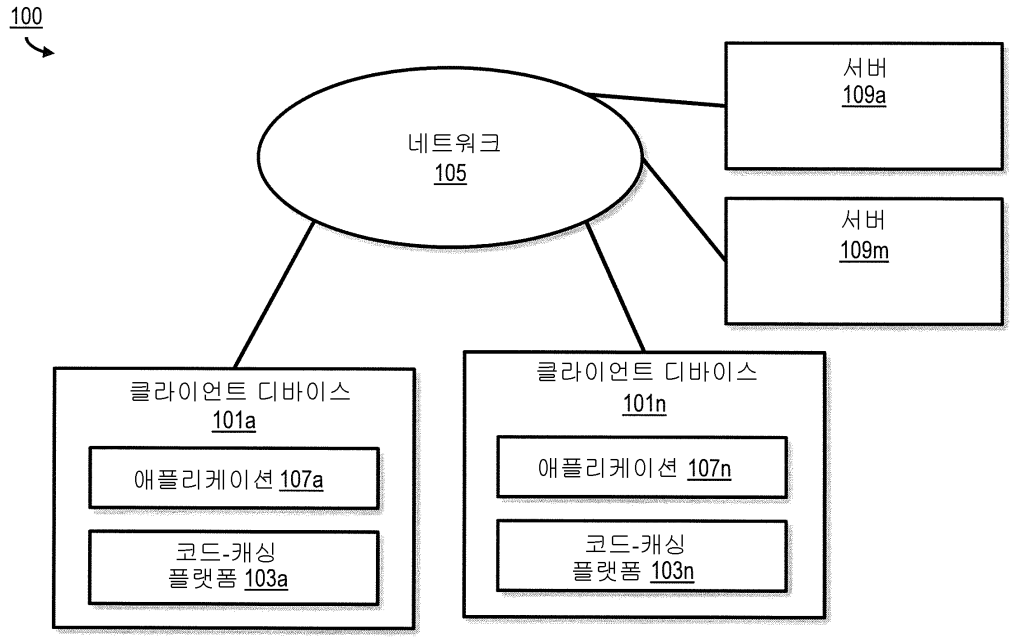
[0110] [0112] 개시된 프로세스들에서 단계들의 임의의 특정 순서 또는 계층은 예시적인 접근법의 예시라는 것이 이해된다. 설계 선호도들에 기초하여, 프로세스들에서의 단계들의 특정 순서 또는 계층이 재배열될 수 있다는 것 또는 모든 예시된 단계들이 수행될 수 있다는 것이 이해된다. 단계들 중 일부는 동시적으로 수행될 수 있다. 예를 들어, 특정 환경들에서는, 멀티태스킹 및 병렬 프로세싱이 유리할 수 있다. 또한, 위에 설명된 구현들의 다양한 시스템 컴포넌트들의 분리는 모든 구현들에서 그러한 분리를 필요로 하는 것으로 이해되지 않아야 하며, 설명된 프로그램 컴포넌트들 및 시스템들은 일반적으로 단일 소프트웨어 제품에 함께 통합되거나 또는 다수의 소프트웨어 제품들로 패키징될 수 있다는 것이 이해되어야 한다.

[0111] [0113] 이전의 설명은 임의의 당업자가 본원에 설명된 다양한 양상들을 실시할 수 있도록 제공된다. 이들 양상들에 대한 다양한 수정들이 당업자들에게는 용이하게 명백할 것이며, 본원에 정의된 일반적인 원리들은 다른 양상들에 적용될 수 있다. 따라서, 청구항들은 본원에 도시된 양상들로 제한되도록 의도되는 것이 아니라, 청구항 문언에 부합하는 최대 범위를 부여하려는 것이며, 여기서, 단수형의 엘리먼트에 대한 참조는 구체적으로 그렇게 언급되지 않으면 "하나 및 오직 하나"를 의미하기보다는 오히려 "하나 또는 그 초과"를 의미하도록 의도된다. 달리 구체적으로 언급되지 않으면, 용어 "몇몇"은 하나 또는 그 초과를 지칭한다. 남성에 관한 대명사(예컨대, 그의)는 여성 및 중성(예컨대, 그녀의 및 그것의)을 포함하며, 그 반대도 가능하다. 제목들 및 부제목들은, 존재한다면, 단지 편의를 위해 사용되며, 본 개시내용을 제한하지 않는다.

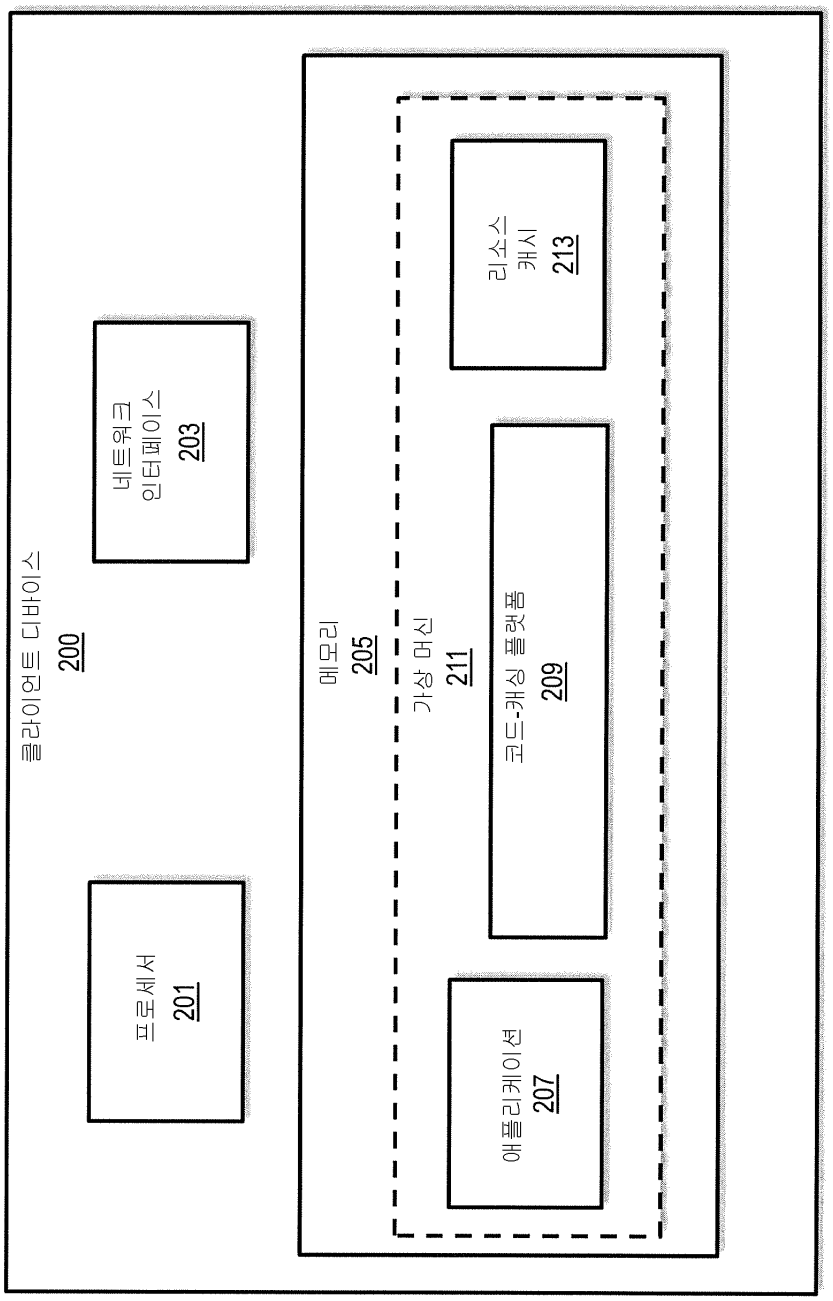
[0112] [0114] "양상"과 같은 문구는, 이러한 양상이 본 기술에 필수적인 것이거나 또는 이러한 양상이 본 기술의 모든 구성들에 적용된다는 것을 암시하지 않는다. 양상과 관련되는 개시내용은 모든 구성들에 적용될 수 있거나 또는 하나 또는 그 초과의 구성들에 적용될 수 있다. 양상과 같은 문구는 하나 또는 그 초과의 양상들을 나타낼 수 있으며, 그 반대도 가능하다. "구성"과 같은 문구는 이러한 구성이 본 기술에 필수적인 것이거나 또는 이러한 구성이 본 기술의 모든 구성들에 적용된다는 것을 암시하지 않는다. 구성과 관련되는 개시내용은 모든 구성들에 적용될 수 있거나 또는 하나 또는 그 초과의 구성들에 적용될 수 있다. 구성과 같은 문구는 하나 또는 그 초과의 구성들을 나타낼 수 있으며, 그 반대도 가능하다.

도면

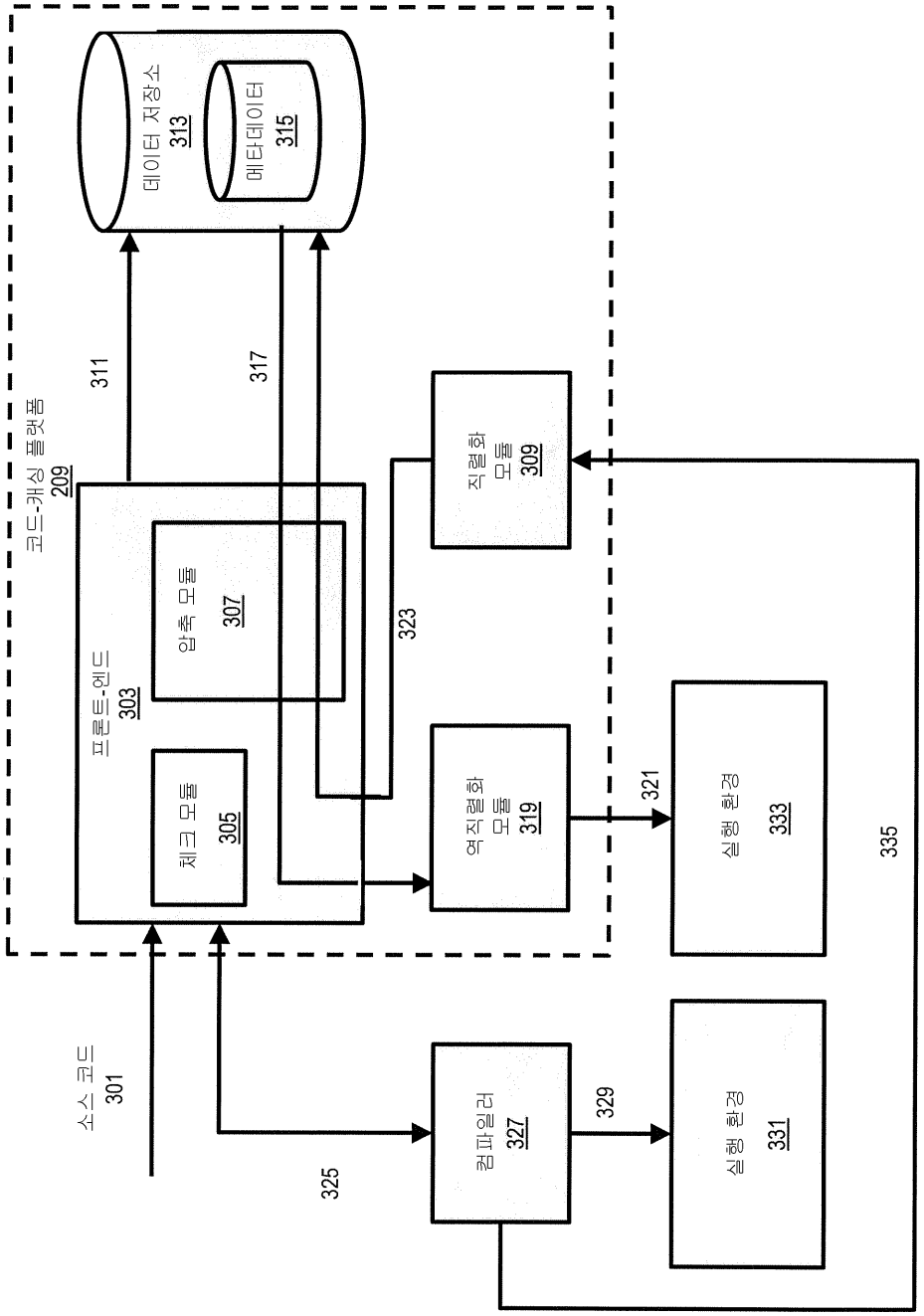
도면1



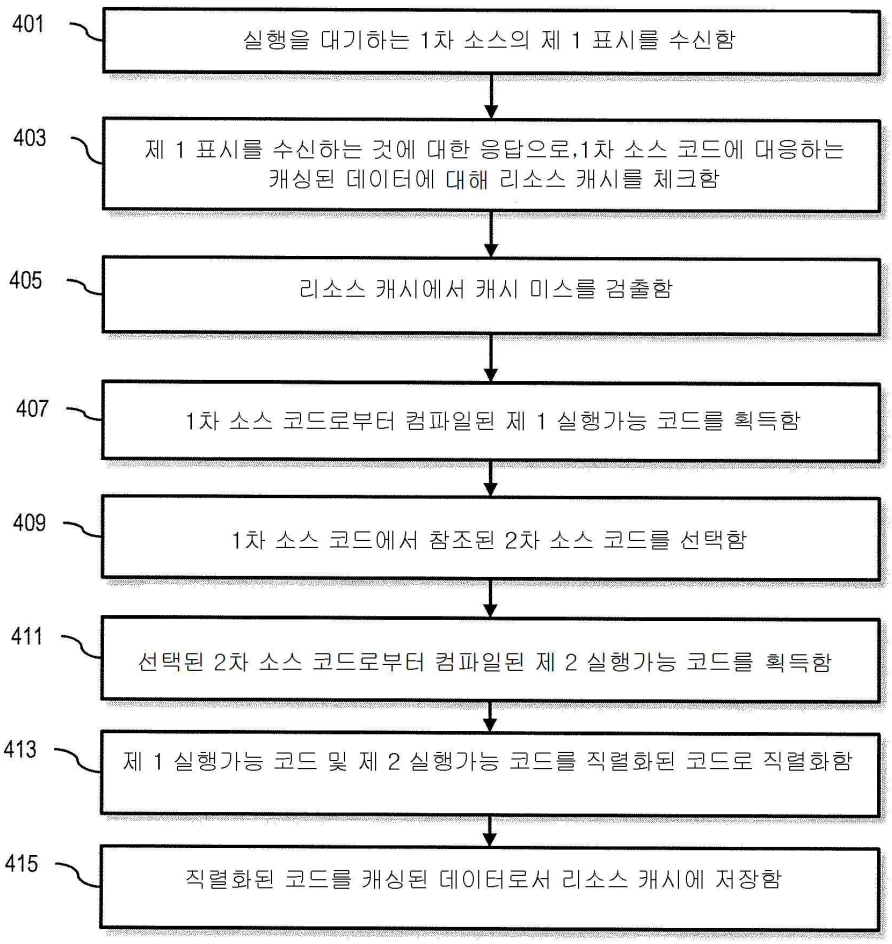
도면2


클라이언트 디바이스
200
프로세서
201
네트워크
인터페이스
203
메모리
205
가상 머신
211
애플리케이션
207
코드-캐싱 플랫폼
209
리소스
캐시
213

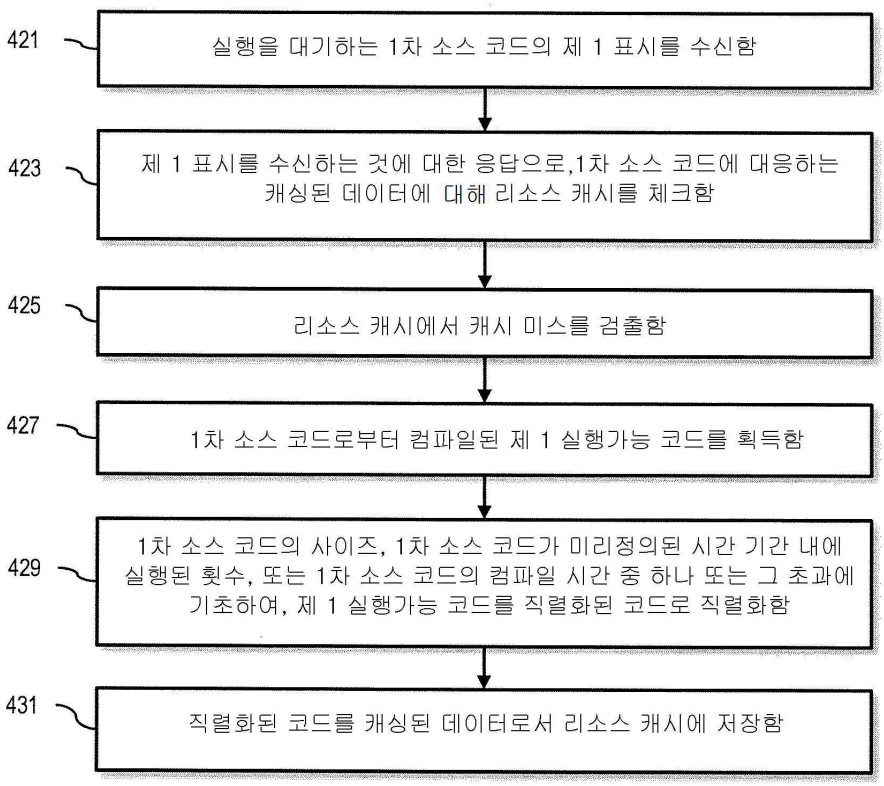
도면3


데이터 저장소
313
메타데이터
315
311
317
코드-캐싱 플랫폼
209
프론트-엔드
303
압축 모듈
307
체크 모듈
305
직렬화
모듈
309
323
역직렬화
모듈
319
321
실행 환경
333
335
소스 코드
301
325
컴파일러
327
329
실행 환경
331

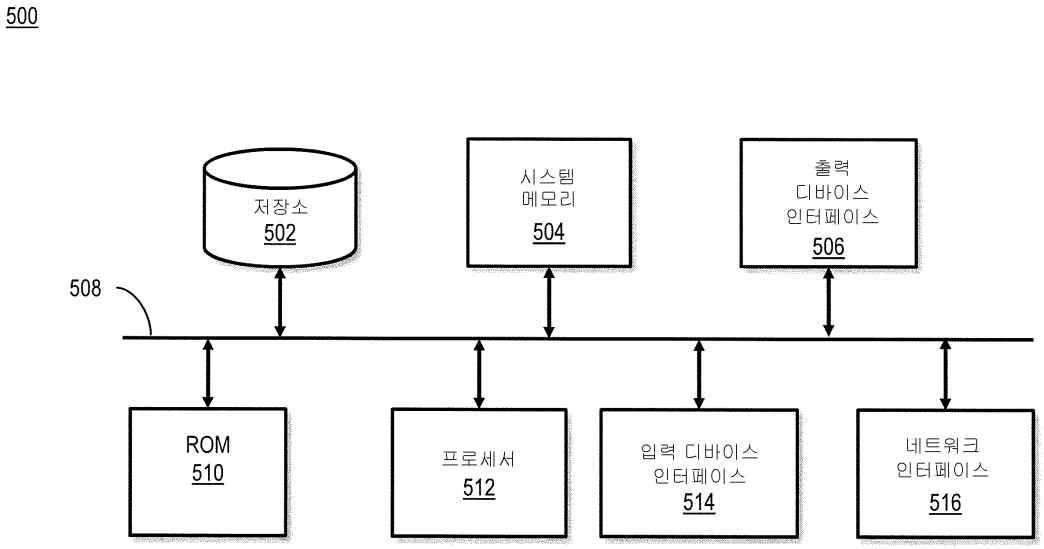
도면4a


401
실행을 대기하는 1차 소스의 제 1 표시를 수신함
403
제 1 표시를 수신하는 것에 대한 응답으로,1차 소스 코드에 대응하는 캐싱된 데이터에 대해 리소스 캐시를 체크함
405
리소스 캐시에서 캐시 미스를 검출함
407
1차 소스 코드로부터 컴파일된 제 1 실행가능 코드를 획득함
409
1차 소스 코드에서 참조된 2차 소스 코드를 선택함
411
선택된 2차 소스 코드로부터 컴파일된 제 2 실행가능 코드를 획득함
413
제 1 실행가능 코드 및 제 2 실행가능 코드를 직렬화된 코드로 직렬화함
415
직렬화된 코드를 캐싱된 데이터로서 리소스 캐시에 저장함

도면4b


421
실행을 대기하는 1차 소스 코드의 제 1 표시를 수신함
423
제 1 표시를 수신하는 것에 대한 응답으로,1차 소스 코드에 대응하는 캐싱된 데이터에 대해 리소스 캐시를 체크함
425
리소스 캐시에서 캐시 미스를 검출함
427
1차 소스 코드로부터 컴파일된 제 1 실행가능 코드를 획득함
429
1차 소스 코드의 사이즈, 1차 소스 코드가 미리정의된 시간 기간 내에 실행된 횟수, 또는 1차 소스 코드의 컴파일 시간 중 하나 또는 그 초과에 기초하여, 제 1 실행가능 코드를 직렬화된 코드로 직렬화함
431
직렬화된 코드를 캐싱된 데이터로서 리소스 캐시에 저장함

도면5


500
저장소
502
시스템
메모리
504
출력
디바이스
인터페이스
506
508
ROM
510
프로세서
512
입력 디바이스
인터페이스
514
네트워크
인터페이스
516