

【발명의 설명】

【발명의 명칭】

로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템 및 그 제거 방법{Detection and Elimination System and Method of Redundant Access Patterns in Programs by Analyzing Log Data}

【기술분야】

<0001> 본 발명은 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템 및 그 제거 방법에 관한 것으로 더욱 상세하게는, 운영체제에서 동작하는 프로그램들이 레지스트리에 비효율적으로 접근하는 패턴을 로그 데이터 분석을 통해 탐지하여 중복 접근을 제거하는 기술에 관한 것이다.

【발명의 배경이 되는 기술】

<0002> 정보시스템 자산의 증가됨에 따라 운영체제(OS)가 설치된 클라우드 서비스, 서버, 데스크탑, 노트북 환경 또는 모바일 디바이스 내에서 다양한 프로그램들이 설치되어 운용되고 있다.

<0003> 이러한 운영체제 내에서 동작중인 다수의 프로그램들은 동시에 레지스트리에 접근하기 때문에 개별적인 프로그램의 불필요한 연산이 수반되고, 이로 인해 시스템 전체의 성능 저하를 야기하는 문제점이 있다.

<0004> 이러한 문제점을 해소하기 위한 방안으로 프로그램이 동작하는 방식이나 행위 즉, 소스코드 분석을 통해 중복을 제거하는 방법이 있으나, 소스코드 분석을 위해서는 많은 시간과 비용이 발생해 비효율적이다.



<0005> 이에 본 출원인은 운영체제에서 동작하는 프로그램들이 레지스트리에 비효율적으로 접근하는 패턴을 로그 데이터 분석을 통해 탐지하여 중복 접근을 제거함으로써, 개별 프로그램 및 운영체제 시스템 자체의 성능을 향상시키기 위한 시스템 및 중복 접근 패턴 제거 방법을 제안하고자 한다.

【선행기술문헌】
【특허문헌】

<0006> (특허문헌 1) 대한민국 공개특허 제10-2017-0058140호(2017.05.26. 공개)

【발명의 내용】
【해결하고자 하는 과제】

<0007> 본 발명의 목적은, 로그 데이터 분석을 통해 운영체제에서 동작하는 프로그램들이 레지스트리에 중복으로 접근하는 패턴을 탐지하여 제거함으로써, 개별 프로그램 및 운영체제 시스템 자체의 성능을 향상시키는데 있다.

<0008> 본 발명의 목적은, 프로그램의 행위를 기록한 로그 데이터 분석을 통해 추출한 중복 데이터를 제거하되, 접근하는 데이터에 대한 갱신이 발생하는 경우, Event-Driven 기법을 통해 데이터 갱신이 발생하는 순간 갱신된 데이터에 다시 접근하도록 제어함으로써, 중복 접근 패턴 제거로 인하여 접근하는 데이터가 갱신된 경우에 최신 데이터에 접근하지 못하는 문제를 미연에 방지하는데 있다.

【과제의 해결 수단】

<0009> 이러한 기술적 과제를 달성하기 위한 본 발명의 일 실시예는 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템으로서, 운영체제에서 동작하는



프로그램들의 연산 횟수 및 연산 시간에 대한 비중을 영역별로 추출하여 로그 분석 정보를 생성하는 로그 데이터 분석부; 로그 분석정보를 분석하여 특정 연산에 대한 중복 및 특정 패턴을 갖는 중복 데이터를 추출하는 중복 데이터 추출부; 중복 데이터 추출부에 의해 추출된 각 영역별 중복 데이터를 제거하여 로그 데이터를 갱신하는 중복 데이터 제거부; 및 중복 데이터가 제거된 이후 프로그램에서 요구되는 모듈화된 작업이 수행되는 경우, event-driven 기법을 통해 프로그램이 갱신된 로그 데이터에 부합하여 작업을 수행하도록 제어하는 프로그램 제어부를 포함하는 것을 특징으로 한다.

<0010> 바람직하게는, 로그 데이터 분석부는 운영체제에서 동작하는 프로그램들의 로그 데이터를 분류하는 로그 분류모듈; 및 분류된 로그 데이터 각각의 연산 횟수 및 연산 시간에 대한 비중을 영역별로 추출하여 로그 분석정보를 생성하는 비중 추출모듈을 포함하는 것을 특징으로 한다.

<0011> 중복 데이터 추출부는, 로그 데이터 분석부로부터 인가받은 로그 분석정보와 대응하는 레지스트리를 트리구조로 색인하여 중복 연산된 데이터를 추출하는 중복 연산 추출모듈; 로그 데이터 분석부로부터 인가받은 로그 분석정보와 대응하는 레지스트리를 트리구조로 색인하여 중복된 패턴 데이터를 추출하는 중복패턴 추출모듈; 및 중복 연산 데이터와 중복 패턴 데이터를 영역별로 분류 및 취합하여 중복 데이터를 특정하는 중복 데이터 취합모듈을 포함하는 것을 특징으로 한다.

<0012> 운영체제는 Windows인 것을 특징으로 한다.

<0013> 전술한 시스템을 기반으로 하는 로그 데이터 분석을 통한 프로그램의 중복



접근 패턴 탐지 제거 방법은, 로그 데이터 분석부가 운영체제에서 동작하는 프로그램들의 로그 데이터 분석하여 로그 분석정보를 생성하는 (a) 단계; 중복 데이터 추출부가 로그 분석정보에 포함된 각 영역에 접근하는 프로그램의 접근 패턴을 분석하여 중복 데이터를 추출하는 (b) 단계; 중복 제거부가 추출된 각 영역별 중복 데이터를 제거하여 로그 데이터를 갱신하는 (c) 단계; 프로그램 제어부가 중복 데이터가 제거된 이후 프로그램에서 요구되는 모듈화된 작업이 수행되는지 여부를 판단하는 (d) 단계; 및 (d) 단계의 판단결과, 중복 데이터가 제거된 이후 프로그램의 작업이 수행이 감지되는 경우, 프로그램 제어부가 event-driven 기법을 통해 프로그램이 갱신된 로그 데이터에 부합하여 작업을 수행하도록 제어하는 (e) 단계를 포함하되, (b) 단계는, 중복 데이터 추출부의 중복연산 추출모듈이 로그 분석정보와 대응하는 레지스트리를 트리구조로 색인하여 중복 연산된 데이터를 추출하는 (b-1) 단계; 중복 데이터 추출부의 중복패턴 추출모듈이 로그 분석정보와 대응하는 레지스트리를 트리구조로 색인하여 중복된 패턴 데이터를 추출하는 (b-2) 단계; 및 중복 데이터 추출부의 중복 데이터 취합모듈이 중복 연산 데이터 및 중복 패턴 데이터를 영역별로 분류 및 취합하여 중복 데이터를 특정하는 (b-3) 단계를 포함하는 것을 특징으로 한다.

【발명의 효과】

상기와 같은 본 발명에 따르면, 로그 데이터 분석을 통해 운영체제에서 동작하는 프로그램들이 레지스트리에 중복으로 접근하는 패턴을 탐지하여 제거함으로써, 개별 프로그램 및 운영체제 시스템 자체의 성능을 향상시키는 효과가 있다.



<0015> 본 발명에 따르면, 프로그램의 행위를 기록한 로그 데이터 분석을 통해 추출한 중복 데이터를 제거하되, 접근하는 데이터에 대한 갱신이 발생하는 경우, Event-Driven 기법을 통해 데이터 갱신이 발생하는 순간 갱신된 데이터에 다시 접근하도록 제어함으로써, 중복 접근 패턴 제거로 인하여 접근하는 데이터가 갱신된 경우에 최신 데이터에 접근하지 못하는 문제를 미연에 방지하는 효과가 있다.

【도면의 간단한 설명】

<0016> 도 1은 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템을 도시한 블록도.

 도 2는 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템의 세부구성을 도시한 블록도.

 도 3은 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템의 특정 연산 중복 사례 및 특정 패턴 중복 사례를 도시한 예시도.

 도 4는 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템의 내부 중복 탐지 알고리즘 및 외부 중복 탐지 알고리즘을 도시한 예시도.

 도 5는 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템의 중복 제거 기법의 원리와 내부 및 외부 중복 제거 알고리즘을 도시한 예시도.

 도 6은 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중



복 접근 패턴 탐지 시스템의 event-driven 기법을 도시한 예시도.

도 7은 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템의 중복 데이터 제거로 인한 성능 향상 결과를 도시한 예시도.

도 8은 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 제거 방법을 도시한 순서도.

도 9는 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 제거 방법의 제S802단계의 세부과정을 도시한 순서도.

도 10은 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 제거 방법의 제S804단계의 세부과정을 도시한 순서도.

도 11은 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 제거 방법의 제S806단계의 세부과정을 도시한 순서도.

【발명을 실시하기 위한 구체적인 내용】

본 발명의 구체적인 특징 및 이점들은 첨부 도면에 의거한 다음의 상세한 설명으로 더욱 명백해질 것이다. 이에 앞서, 본 명세서 및 청구범위에 사용된 용어나 단어는 발명자가 그 자신의 발명을 가장 최선의 방법으로 설명하기 위해 용어의 개념을 적절하게 정의할 수 있다는 원칙에 입각하여 본 발명의 기술적 사상에 부합하는 의미와 개념으로 해석되어야 할 것이다. 또한, 본 발명에 관련된 공지 기능 및 그 구성에 대한 구체적인 설명이 본 발명의 요지를 불필요하게 흐릴 수 있다고 판단되는 경우에는, 그 구체적인 설명을 생략하였음에 유의해야 할 것이다.



<0018> 도 1을 참조하면 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템(100)은, 운영체제에서 동작하는 프로그램들의 로그 데이터를 분석을 통해 연산 횟수 및 연산 시간에 대한 비중을 영역별로 추출하여 로그 분석정보를 생성하는 로그 데이터 분석부(110)와, 로그 분석정보에 포함된 각 영역에 접근하는 프로그램의 접근 패턴을 분석하여 특정 연산에 대한 중복 및 특정 패턴을 갖는 중복 데이터를 추출하는 중복 데이터 추출부(120)와, 내부 중복 패턴 및 외부 중복 패턴 제거를 위한 알고리즘을 색인하여 중복 데이터 추출부(120)에 의해 추출된 각 영역별 중복 데이터를 제거하여 로그 데이터를 갱신하는 중복 데이터 제거부(130), 및 중복 데이터가 제거된 이후 프로그램에서 요구되는 모듈화된 작업이 수행되는 경우, event-driven 기법을 통해 프로그램이 갱신된 로그 데이터에 부합하여 작업을 수행하도록 제어하는 프로그램 제어부(140)를 포함하여 구성된다.

<0019> 이하에서는 그 구체적인 언급을 생략하겠으나, 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템(100)이 적용되는 운영체제는, Windows, Linux, Mac OS, Android 또는 iOS 중에 어느 하나로 구성되며, 바람직하게는 Windows 운영체제로 설정된다.

<0020> 이하, 도 2를 참조하여 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템(100)의 세부구성에 대해 살펴보면 아래와 같다.

<0021> 먼저, 로그 데이터 분석부(110)는 운영체제에서 동작하는 프로그램들의 로그



데이터를 분류하는 로그 분류모듈(112), 및 분류된 로그 데이터 각각의 연산 횟수 및 연산 시간에 대한 비중을 영역별로 추출하여 로그 분석정보를 생성하는 비중 추출모듈(114)을 포함하여 구성된다.

이때, 비중 추출모듈(114)의 비중 추출 대상인 영역은 레지스트리, 파일시스템, 네트워크 또는 프로세스 중에 어느 하나를 포함하며, 로그 데이터 각각의 연산 횟수 및 연산 시간에 대한 비중은 아래의 [표 1]에 나타난 예와 같다.

【표 1】

Operation types	Occurrences	Duration (seconds)
Registry	11,463,692 (76.43%)	824.25 (31.42%)
File System	3,426,005 (22.84%)	1,799.22 (68.58%)
Network	93,651 (76.43%)	0.0001 (0.000004%)
Process	16,652 (76.43%)	0.0003 (0.00001%)
Total	15,000,000 (100.00%)	2,623.47 (100.00%)

한편, 도 3의 (a)는 특정 연산의 중복 사례를 도시한 예시도이고, 도 3의 (b)는 특정 패턴의 중복 사례를 도시한 예시도이며, 도 4의 (a)는 내부 중복 탐지 알고리즘을 도시한 예시도이고, 도 4의 (b)는 외부 중복 탐지 알고리즘을 도시한 예시도이다.

도 2 내지 도 4를 참조하면, 중복 데이터 추출부(120)는 로그 데이터 분석부(110)로부터 인가받은 로그 분석정보와 대응하는 레지스트리를 트리구조로 색인하여 중복 연산된 데이터를 추출하는 중복연산 추출모듈(122), 로그 데이터 분석부(110)로부터 인가받은 로그 분석정보와 대응하는 레지스트리를 트리구조로 색인하여 중복된 패턴 데이터를 추출하는 중복패턴 추출모듈(124), 및 중복연산 추출모



들(122)로부터 인가받은 중복 연산 데이터와 중복패턴 추출모듈(124)로부터 인가받은 중복 패턴 데이터를 영역별로 분류 및 취합하여 중복 데이터를 특정하는 중복 데이터 취합모듈(126)을 포함하여 구성된다.

<0026> 또한, 도 5의 (a)는 중복 제거 기법의 원리를 도시한 예시도이고, 도 5의 (b)는 내부 중복 제거 알고리즘을 도시한 예시도이며, 도 5의 (c)는 외부 중복 제거 알고리즘을 도시한 예시도이다.

<0027> 도 2 및 도 5를 참조하면, 중복 데이터 제거부(130)는 중복 데이터 추출부(120)로부터 각 영역별로 특정된 중복 데이터를 인가받고, 중복 데이터 제거를 위해 저장소에서 내부 중복 패턴 및 외부 중복 패턴 제거를 위한 알고리즘을 색인하는 알고리즘 색인모듈(132), 및 영역별로 특정된 중복 데이터를 제거하여 로그 데이터를 갱신하는 중복 데이터 제거모듈(134)을 포함하여 구성된다.

<0028> 이때, 로그 데이터 갱신은, 동일한 프로그램의 실행 개수와 대응하여 생성되는 로그 데이터들 중에 중복된 로그 데이터를 삭제하는 것으로 이해함이 바람직하다.

<0029> 그리고, 도 6은 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템의 event-driven 기법을 도시한 예시도이다.

<0030> 도 2 및 도 6을 참조하면, 프로그램 제어부(140)는 중복 데이터 제거부(130)로부터 중복 데이터 제거에 따라 갱신된 로그 데이터를 인가받고, 갱신된 로그 데이터와 대응하는 프로그램의 구동을 감지하는 작업 감지모듈(142), 및 감지된 프로그램이 갱신된 로그 데이터에 부합하여 구동하도록 스레드 링크를 매칭시키는 작업



제어모듈(144)을 포함하여 구성된다.

<0031> 여기서, 스레드 링크를 매칭이란, 동일한 프로그램의 실행 개수와 대응하여 생성되는 로그 데이터들 중에 중복된 로그 데이터가 삭제됨에 따라 다수의 동일한 프로그램이 동일한 로그 데이터 단일개와 매칭시키는 것으로 이해함이 바람직하다.

<0032> 이하, 도 7을 참조하여 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템에 의해 중복 데이터가 제거된 이후, 성능 향상 결과에 대해 살펴보면 아래와 같다.

<0033> 종래의 프로그램 성능 개선 기술들은 프로그램을 소스코드 또는 바이너리 수준에서 분석함에 따라 많은 시간과 비용이 발생하는 문제점이 있었다. 그러나, 본 발명의 일 실시예에 따라 프로그램의 중복 접근 패턴 탐지하여 중복 데이터를 제거한 경우, 운영체제의 성능이 개선된 것을 확인하였다.

<0034> 도 7에 나타난 바와 같이 운영체제의 성능 개선은, 단일개의 프로그램 구동시에 8.93%의 성능이 개선되었고, 5개의 프로그램 구동시 12.39%의 성능이 개선되었으며, 10개의 프로그램 구동시 19.18%의 성능이 개선되었고, 15개의 프로그램 구동시 21.93%의 성능이 개선되었으며, 20개의 프로그램 구동시 26.21%의 성능이 개선되었음을 확인하였다.

<0035> 측정결과, 운영체제 내에 구동되는 프로그램 실행 개수가 증가할수록 중복 데이터 제거에 따른 운영체제 성능이 향상되는 것을 알 수 있다.

<0036> 이하, 도 8을 참조하여 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 방법에 대해 살펴보면 아래와 같다.



<0037> 먼저, 로그 데이터 분석부가 운영체제에서 동작하는 프로그램들의 로그 데이터 분석하여 로그 분석정보를 생성한다(S802). 이때, 로그 분석정보는, 실행중인 프로그램의 연산 횟수 및 연산 시간에 대한 비중을 영역별로 추출한 데이터를 포함한다.

<0038> 이어서, 중복 데이터 추출부가 로그 분석정보에 포함된 각 영역에 접근하는 프로그램의 접근 패턴을 분석하여 특정 연산에 대한 중복 및 특정 패턴을 갖는 중복 데이터를 추출한다(S804).

<0039> 뒤이어, 중복 제거부가 추출된 각 영역별 중복 데이터를 제거하여 로그 데이터를 갱신한다(S806).

<0040> 그리고, 프로그램 제어부가 중복 데이터가 제거된 이후 프로그램에서 요구되는 모듈화된 작업이 수행되는지 여부를 판단한다(S808).

<0041> 제S808단계의 판단결과, 중복 데이터가 제거된 이후 프로그램에서 요구되는 모듈화된 작업이 수행되는 경우, 프로그램 제어부가 event-driven 기법을 통해 프로그램이 갱신된 로그 데이터에 부합하여 작업을 수행하도록 제어한다(S810).

<0042> 이하, 도 9를 참조하여 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 방법의 제S802단계의 세부과정에 대해 살펴보면 아래와 같다.

<0043> 먼저, 로그 데이터 분석부의 로그 분류모듈이 운영체제에서 동작하는 프로그램들의 로그 데이터를 분류한다(S902).

<0044> 그리고, 로그 데이터 분석부의 비중 추출모듈이 분류된 로그 데이터 각각의



연산 횟수 및 연산 시간에 대한 비중을 영역별로 추출하여 로그 분석정보를 생성한다(S904).

<0045> 이하, 도 10을 참조하여 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 방법의 제S804단계의 세부과정에 대해 살펴보면 아래와 같다.

<0046> 제S802단계 이후, 중복 데이터 추출부의 중복연산 추출모듈이 로그 분석정보와 대응하는 레지스트리를 트리구조로 색인하여 중복 연산된 데이터를 추출한다(S1002).

<0047> 이어서, 중복 데이터 추출부의 중복패턴 추출모듈이 로그 분석정보와 대응하는 레지스트리를 트리구조로 색인하여 중복된 패턴 데이터를 추출한다(S1004).

<0048> 그리고, 중복 데이터 추출부의 중복 데이터 취합모듈이 중복 연산 데이터 및 중복 패턴 데이터를 영역별로 분류 및 취합하여 중복 데이터를 특정한다(S1006).

<0049> 이하, 도 11을 참조하여 본 발명의 일 실시예에 따른 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 방법의 제S806단계의 세부과정에 대해 살펴보면 아래와 같다.

<0050> 제S804단계 이후, 중복 제거부의 알고리즘 색인모듈이 각 영역별로 특정된 중복 데이터를 인가받고, 저장소에서 내부 중복 패턴 및 외부 중복 패턴 제거를 위한 알고리즘을 색인한다(S1102).

<0051> 그리고, 중복 제거부의 중복 데이터 제거모듈이 영역별로 특정된 중복 데이터를 제거하여 로그 데이터를 갱신한다(S1104).



상기와 같은 본 발명에 따르면, 로그 데이터 분석을 통해 운영체제에서 동작하는 프로그램들이 레지스트리에 중복으로 접근하는 패턴을 탐지하여 제거함으로써, 개별 프로그램 및 운영체제 시스템 자체의 성능을 향상시키고, 접근하는 데이터에 대한 갱신이 발생하는 경우, Event-Driven 기법을 통해 데이터 갱신이 발생하는 순간 갱신된 데이터에 다시 접근하도록 제어함으로써, 중복 접근 패턴 제거로 인하여 접근하는 데이터가 갱신된 경우에 최신 데이터에 접근하지 못하는 문제를 미연에 방지할 수 있다.

이상으로 본 발명의 기술적 사상을 예시하기 위한 바람직한 실시예와 관련하여 설명하고 도시하였지만, 본 발명은 이와 같이 도시되고 설명된 그대로의 구성 및 작용에만 국한되는 것이 아니며, 기술적 사상의 범주를 일탈함이 없이 본 발명에 대해 다수의 변경 및 수정이 가능함을 당업자들은 잘 이해할 수 있을 것이다. 따라서 그러한 모든 적절한 변경 및 수정과 균등물들도 본 발명의 범위에 속하는 것으로 간주되어야 할 것이다.

【부호의 설명】

100: 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템	
110: 로그 데이터 분석부	
112: 로그 분류모듈	114: 비중 추출모듈
120: 중복 데이터 추출부	
122: 중복연산 추출모듈	124: 중복패턴 추출모듈
126: 중복패턴 추출모듈	



130: 중복 데이터 제거부

132: 알고리즘 색인모듈

140: 프로그램 제어부

142: 작업 감지모듈

134: 중복 데이터 제거모듈

144: 작업 제어모듈



【청구범위】

【청구항 1】

운영체제에서 동작하는 프로그램들의 연산 횟수 및 연산 시간에 대한 비중을 영역별로 추출하여 로그 분석정보를 생성하는 로그 데이터 분석부;

상기 로그 분석정보를 분석하여 특정 연산에 대한 중복 및 특정 패턴을 갖는 중복 데이터를 추출하는 중복 데이터 추출부;

상기 중복 데이터 추출부에 의해 추출된 각 영역별 중복 데이터를 제거하여 로그 데이터를 갱신하는 중복 데이터 제거부; 및

중복 데이터가 제거된 이후 프로그램에서 요구되는 모듈화된 작업이 수행되는 경우, event-driven 기법을 통해 프로그램이 갱신된 로그 데이터에 부합하여 작업을 수행하도록 제어하는 프로그램 제어부를

포함하는 것을 특징으로 하는 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템.

【청구항 2】

제1항에 있어서,

상기 로그 데이터 분석부는,

운영체제에서 동작하는 프로그램들의 로그 데이터를 분류하는 로그 분류모듈; 및

분류된 로그 데이터 각각의 연산 횟수 및 연산 시간에 대한 비중을 영역별로 추출하여 로그 분석정보를 생성하는 비중 추출모듈을



포함하는 것을 특징으로 하는 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템.

【청구항 3】

제1항에 있어서,

상기 중복 데이터 추출부는,

상기 로그 데이터 분석부로부터 인가받은 로그 분석정보와 대응하는 레지스트리를 트리구조로 색인하여 중복 연산된 데이터를 추출하는 중복연산 추출모듈;

상기 로그 데이터 분석부로부터 인가받은 로그 분석정보와 대응하는 레지스트리를 트리구조로 색인하여 중복된 패턴 데이터를 추출하는 중복패턴 추출모듈;
및

상기 중복 연산 데이터와 중복 패턴 데이터를 영역별로 분류 및 취합하여 중복 데이터를 특징하는 중복 데이터 취합모듈을

포함하는 것을 특징으로 하는 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템.

【청구항 4】

제1항에 있어서,

상기 운영체제는,

Windows인 것을 특징으로 하는 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템.

【청구항 5】



(a) 로그 데이터 분석부가 운영체제에서 동작하는 프로그램들의 로그 데이터 분석하여 로그 분석정보를 생성하는 단계;

(b) 중복 데이터 추출부가 로그 분석정보에 포함된 각 영역에 접근하는 프로그램의 접근 패턴을 분석하여 중복 데이터를 추출하는 단계;

(c) 중복 제거부가 추출된 각 영역별 중복 데이터를 제거하여 로그 데이터를 갱신하는 단계;

(d) 프로그램 제어부가 중복 데이터가 제거된 이후 프로그램에서 요구되는 모듈화된 작업이 수행되는지 여부를 판단하는 단계; 및

(e) 상기 (d) 단계의 판단결과, 중복 데이터가 제거된 이후 프로그램의 작업이 수행이 감지되는 경우, 프로그램 제어부가 event-driven 기법을 통해 프로그램이 갱신된 로그 데이터에 부합하여 작업을 수행하도록 제어하는 단계를 포함하되,

상기 (b) 단계는,

(b-1) 중복 데이터 추출부의 중복연산 추출모듈이 로그 분석정보와 대응하는 레지스트리를 트리구조로 색인하여 중복 연산된 데이터를 추출하는 단계;

(b-2) 중복 데이터 추출부의 중복패턴 추출모듈이 로그 분석정보와 대응하는 레지스트리를 트리구조로 색인하여 중복된 패턴 데이터를 추출하는 단계; 및

(b-3) 중복 데이터 추출부의 중복 데이터 취합모듈이 중복 연산 데이터 및 중복 패턴 데이터를 영역별로 분류 및 취합하여 중복 데이터를 특정하는 단계를

포함하는 것을 특징으로 하는 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 제거 방법.



【요약서】

【요약】

본 발명은 로그 데이터 분석을 통한 프로그램의 중복 접근 패턴 탐지 시스템 및 그 제거 방법에 관한 것으로서, 운영체제에서 동작하는 프로그램들의 연산 횟수 및 연산 시간에 대한 비중을 영역별로 추출하여 로그 분석정보를 생성하는 로그 데이터 분석부; 로그 분석정보를 분석하여 특정 연산에 대한 중복 및 특정 패턴을 갖는 중복 데이터를 추출하는 중복 데이터 추출부; 중복 데이터 추출부에 의해 추출된 각 영역별 중복 데이터를 제거하여 로그 데이터를 갱신하는 중복 데이터 제거부; 및 중복 데이터가 제거된 이후 프로그램에서 요구되는 모듈화된 작업이 수행되는 경우, event-driven 기법을 통해 프로그램이 갱신된 로그 데이터에 부합하여 작업을 수행하도록 제어하는 프로그램 제어부를 포함한다.

로그 데이터 분석을 통해 운영체제에서 동작하는 프로그램들이 레지스트리에 중복으로 접근하는 패턴을 탐지하여 제거함으로써, 개별 프로그램 및 운영체제 시스템 자체의 성능을 향상시키는 효과가 있다.

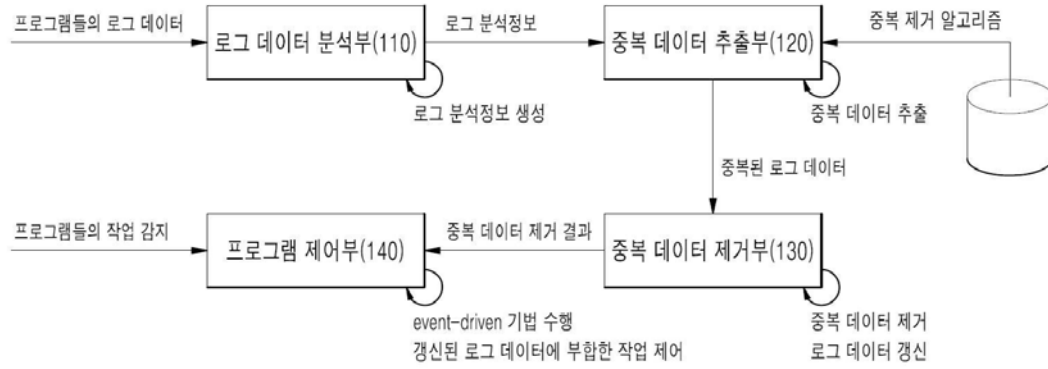
【대표도】

도 1



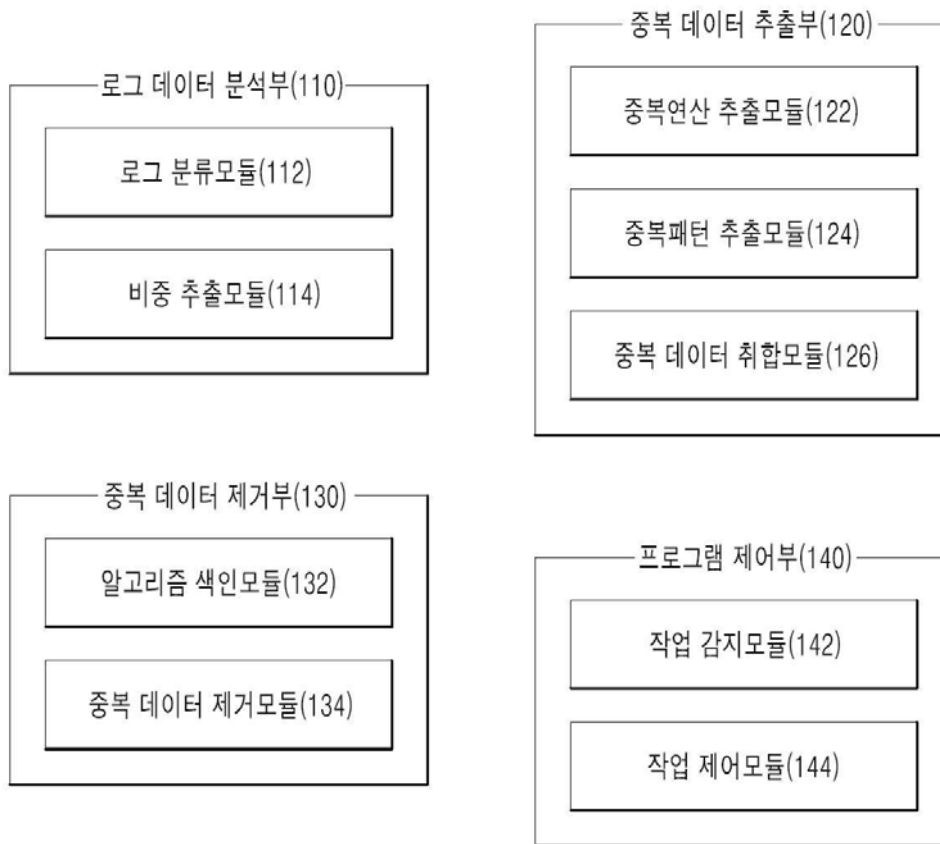
【도면】

【도 1】















100















【도 2】



【도 3】

Process Name	Operation	Path
 Explorer.EXE	 RegQueryKey	HKCU\Software\Classes
 Explorer.EXE	 RegQueryKey	HKCU\Software\Classes
 Explorer.EXE	 RegQueryKey	HKCU\Software\Classes
 Explorer.EXE	 RegQueryKey	HKCU\Software\Classes
 Explorer.EXE	 RegQueryKey	HKCU\Software\Classes
 Explorer.EXE	 RegQueryKey	HKCU\Software\Classes

(a)

Process Name	Operation	Path
 Explorer.EXE	 RegOpenKey	HKCU\Software\Microsoft\OneDrive\Accounts
 Explorer.EXE	 RegQueryValue	HKCU\Software\Microsoft\OneDrive\Accounts\LastUpdate
 Explorer.EXE	 RegCloseKey	HKCU\Software\Microsoft\OneDrive\Accounts
 Explorer.EXE	 RegQueryKey	HKLM
 Explorer.EXE	 RegOpenKey	HKCU\Software\Microsoft\OneDrive\Accounts
 Explorer.EXE	 RegQueryValue	HKCU\Software\Microsoft\OneDrive\Accounts\LastUpdate
 Explorer.EXE	 RegCloseKey	HKCU\Software\Microsoft\OneDrive\Accounts

(b)



Internal redundancy detection

```

1  def data_preprocessing (process) :
2      data = entire log data collected from ProcessMonitor
3      process_data = [] // each element consists of (BS_ID, Operation, Path)
4      BS_ID = 0
5      for i in range(0, len(data)) :
6          if (data[i]['ProcessName'] == process) :
7              if (data[i]['Operation'] == 'RegOpenKey') :
8                  j = i
9                  while (! (data[j]['Operation'] == 'RegCloseKey'
10                        and data[j]['Path'] == data[i]['Path'])) :
11                      process_data.append((BS_ID, data[j]['Operation'], data[j]['Path']))
12                      j = j + 1
13              BS_ID = BS_ID + 1
14      return process_data
15
16  def detect_internal_redundancy (process_data) :
17      internal_redundant_patterns = {}
18      i = 0
19      while (i < len(process_data)) :
20          if (process_data[i]['Operation'] == 'RegOpenKey') :
21              j = i
22              while (! (process_data[j]['Operation'] == 'RegCloseKey'
23                    and process_data[j]['Path'] == process_data[i]['Path'])
24                    and process_data[j]['Operation'] != 'Reg_WriteOperations') :
25                  if (process_data[i]['Path'] in process_data[j]['Path']):
26                      internal_pattern = (process_data[j]['BS_ID'], process_data[j]['Operation'],
27                                         process_data[j]['Path'])
28                      if (process_data[j] not in internal_redundant_patterns.keys()) :
29                          internal_redundant_patterns.append(internal_pattern, 1)
30                      else :
31                          (internal_pattern, count) = internal_redundant_patterns.search(internal_pattern)
32                          internal_redundant_patterns.update(internal_pattern, count+1)
33                  j = j + 1
34              i = i + 1
35      return internal_redundant_patterns

```



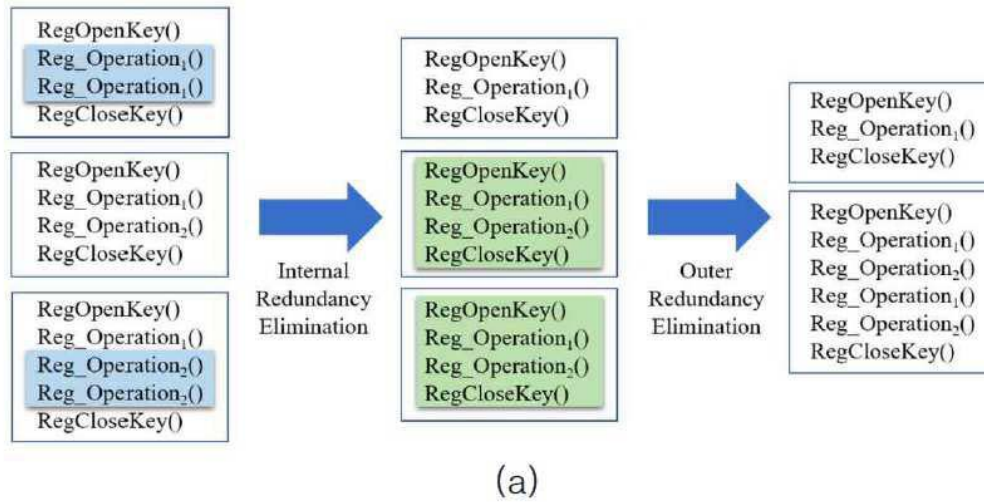
Outer redundancy detection

```

1  def detect_outer_redundancy (process_data) :
2      outer_redundant_patterns = []
3      i = 0
4      while ( i < len(process_data) ) :
5          if (process_data['Operation'][i] == 'RegOpenKey') :
6              outer_pattern = []
7              j = i
8              while ( ! (process_data['Operation'][j] == 'RegCloseKey'
                           and process_data['Path'][j] == process_data['Path'][i] ) ) :
9                  if (process_data['Path'][i] in process_data['Path'][j]):
10                     outer_pattern.append((process_data['BS_ID'][j],
                                             process_data['Operation'][j], process_data['Path'][j]))
11                     j = j + 1
12                 if (outer_pattern not in outer_redundant_patterns.keys()):
13                     outer_redundant_patterns.append(outer_pattern, 1)
14                 else :
15                     (outer_pattern, count) = outer_redundant_patterns.search(outer_pattern)
16                     outer_redundant_patterns.update(outer_pattern, count+1)
17             i = i + 1
18  return outer_redundant_patterns

```





(a)

Original pattern	Internal redundancy elimination
1 // Perform both registry operations and	1 // Perform registry operations once
2 // main operation for n times	2 RegOpenKey()
3 RegOpenKey()	3 Reg_Operations()
4	4
5 for i in n :	5 // Perform only main operations for n times
6 Reg_Operations()	6 for i in n :
7 MainOperations()	7 MainOperations()
8	8
9 RegCloseKey()	9 RegCloseKey()
10	10

(b)

Original pattern	Outer redundancy elimination
1 // Perform both registry operations and	1 // Perform RegOpenKey once
2 // main operation for n times	2 RegOpenKey()
3	3
4 for i in n :	4 // Perform Reg_Operations and
5 RegOpenKey()	5 // main operations for n times
6 Reg_Operations()	6 for i in n :
7 MainOperations()	7 Reg_Operations()
8 RegCloseKey()	8 MainOperations()
9	9
10	10 // Perform RegCloseKey once
11	11 RegCloseKey()

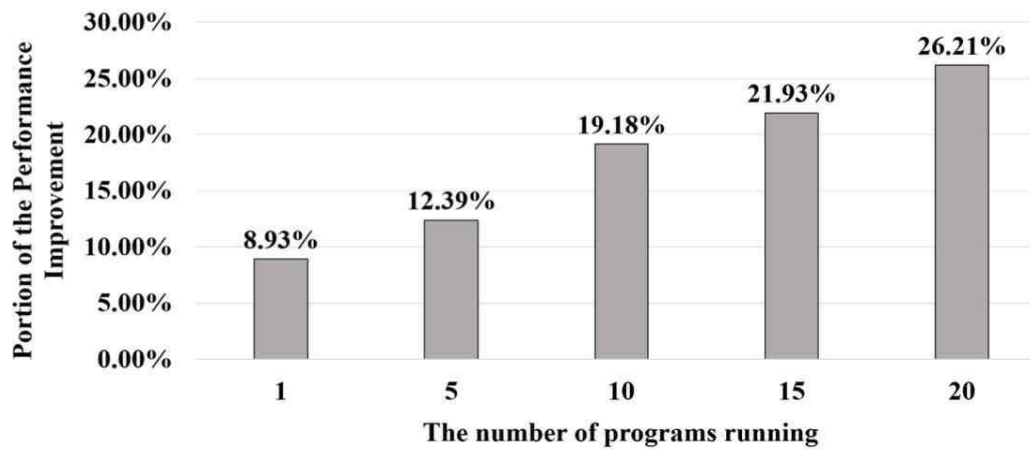
(c)



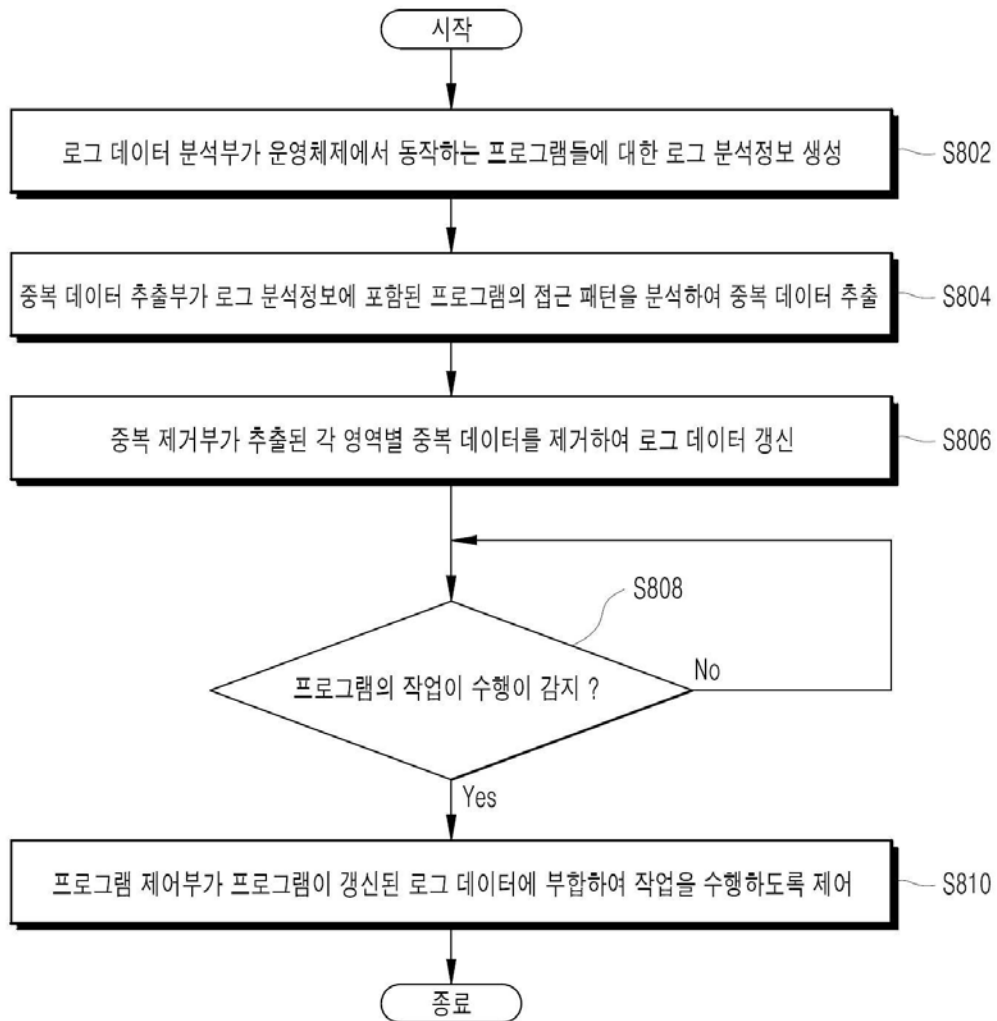
【도 6】

Main Thread		Event Handler Thread	
1	// Perform registry operations once	1	// Register event to catch the updates
2	Reg_Operations()	2	// for a target registry
3	// Call event handler in the other thread	3	RegNotifyChangeKeyValue()
4	Call_EventHandler()	4	
5		5	// Work infinitely to catch the registered event
6	// Perform only main operations for n times	6	while (1) :
7	for i in n :	7	// Wait until the registered event is caught
8	MainOperations()	8	WaitForSingleObject()
9		9	
10		10	// Read updated registry values
11		11	Reg_Operations()

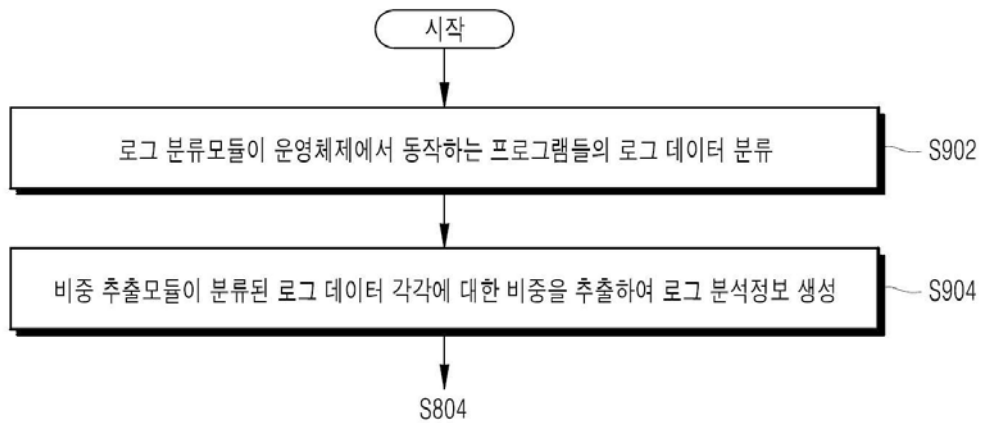
【도 7】



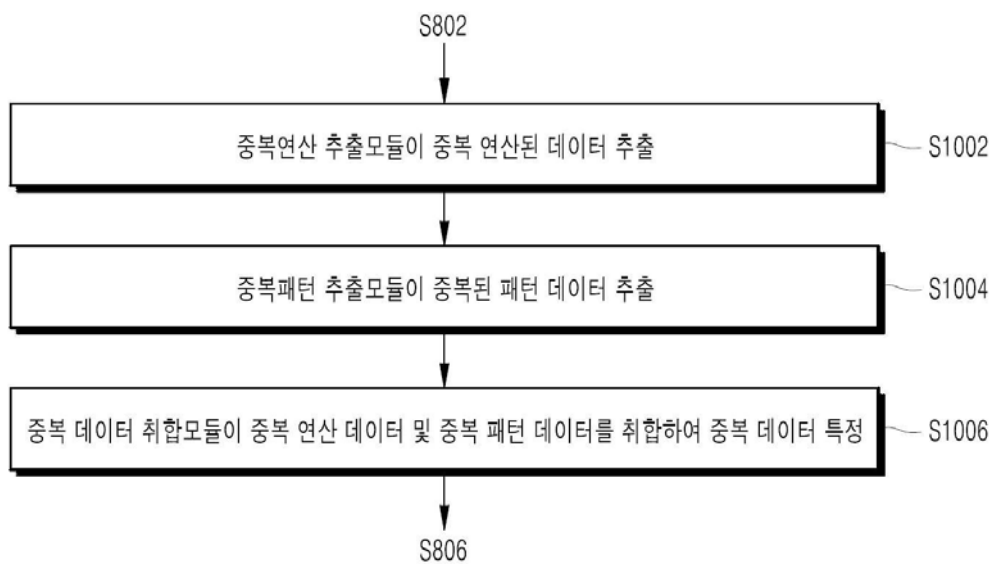
【도 8】



【도 9】



【도 10】



【도 11】

