http://dx.doi.org/10.7236/JIWIT.2012.12.4.201
JIWIT 2012-4-26

기간기반 복합 이벤트 패턴 검출

Detection of Complex Event Patterns over Interval-based Events

강만모*, 박상무*, 김상락*, 김강현**, 이동형**

Man-Mo Kang, Sang-Mu Park, Sank-Rak Kim, Kang-Hyun Kim, Dong-Hyeong Lee

요 약 시점기반 복합 이벤트 처리는 각 이벤트에 하나의 타임스탬프를 사용하여 즉각적인 이벤트를 처리한다. 하지만, 시점기반의 이벤트 처리로는 이벤트의 활동 기간이 중요한 역할을 하는 금융, 멀티미디어, 의학, 기상학 같은 분야에서 복합적인 시제 관계를 표현하기에는 불충분하다. 실세계의 애플리케이션 분야에서, 이벤트는 기간을 가지며, 두 종류 이상의 이벤트는 시간적으로 겹쳐질 수도 있고, 하나의 이벤트가 다른 이벤트를 포함할 수도 있다. 이런 종류의 이벤트들에 대한 관계는 시점기반 이벤트처럼 연속적이지 않을 수도 있다. 본 논문에서는 기간기반 이벤트를 사용하여 복합 이벤트의 패턴을 검출하는 방법을 설계하고 구현한다. 기간기반 이벤트는 시점기반 이벤트가 다룰 수 없는 이벤트들 사이의 겹침과 포함관계를 표현할 수 있다. 기간기반 이벤트 연산자는 시작 끝점과 종료 끝점을 사용하여 이벤트의 기간을 나타내고, 기간기반 이벤트의 시퀀스를 표현하여 복합 이벤트 패턴을 검출할 수 있다.본 논문에서는 복합 이벤트 패턴 검출의 효율성을 높이기 위해 활성 인스턴스 스택을 사용하는 알고리즘을 제시하며, 이벤트의 시퀀스를 구성할 때 중간 결과의 개수를 줄이기 위해 윈도우 푸시다운 기법을 적용하여 수행시간과 메모리의 효율을 높인다.

Abstract The point-based complex event processing handled an instantaneous event by using one time stamp in each event. However, the activity period of the event plays the important role in the field which is the same as the finance, multimedia, medicine, and meteorology. The point-based event is insufficient for expressing the complex temporal relationship in this field. In the application field of the real-time world, the event has the period. The events more than two kinds can be temporally overlapped. In addition, one event can include the other event. The relation about the events of kind of these can not be successive like the point-based event. This thesis designs and implements the method detecting the patterns of the complex event by using the interval-based events. The interval-based events can express the overlapping relation between events. Futhermore, it can include the others. By using the end point of beginning and end point of the termination, the operator of interval-based events shows the interval-based events. It expresses the sequence of the interval-based events and can detect the complex event patterns. This thesis proposes the algorithm using the active instance stack in order to raise efficiency of detection of the complex event patterns. When comprising the event sequence, this thesis applies the window push down technique in order to reduce the number of intermediate results. It raises the utility factor of the running time and memory.

Key Words : interval-based events, detection of complex event patterns, temporal constraint, correlation, time stamp

*정회원, 울산대학교 전기공학부
**정회원, 한국폴리텍 VII 울산캠퍼스 정보통신시스템과
접수일자 : 2012년 5월 22일, 수정완료 : 2012년 7월 5일
게재확정일자 : 2012년 8월 10일

Received: 22 May 2012 / Revised: 5 July 2012
Accepted: 10 August 2012
**Corresponding Author: manmoakng@ulsan.ac.kr
Dept. of Computer Engineering and Information Technology, University of Ulsan, Korea

Ⅰ. 서 론

비즈니스 환경 속에서 기업이 미처 예측하지 못한 이벤트의 발생, 새롭게 등장하는 위협과 기회를 효과적으로 감지하고 대응할 수 있는 전략에 대한 관심이 그 어느 때보다도 높다. 그 중에서도 비즈니스 통찰력을 갖고 시장의 상황을 신속하게 인식해 시스템이 빠르게 변화할 수 있도록 돕고 즉각적인 초치를 취할 수 있도록 하는 복합 이벤트 처리가 주목받고 있다[1][2][3].

이벤트란 현재 비즈니스 상태에서 특정하게 감지되는 변화가 발생했음을 나타내는 모든 신호와 메시지를 뜻한다. 고객의 개인 정보 변경, 경쟁사 제품 가격 변화, 작업장에서 정전 발생, 보험금 수령 사기 감지 등이 이벤트에 포함된다. 이러한 비즈니스 이벤트 감지 및 대응 역량을 강화하여 기업 비즈니스의 위험 요소를 줄이거나 비즈니스 기회를 포착할 수 있도록 할 뿐만 아니라 보다 향상된 기업의 민첩성을 달성하게 하는 핵심 기술이 복합 이벤트 처리 기술이다[4]. 이벤트 처리의 핵심은 단순히 많은 이벤트 사례를 확보하는 것이 아니라, 서로 다른 다양한 이벤트의 상관관계를 분석해 조치 가능한 패턴을 검출하는 것이다[3]. 기업 내에서 발생하는 많은 수의 이벤트가 가진 중요도는 동일하지 않으며 이벤트 메시지의 소스도 기업 내부가 아닌 외부일 수도 있고 시스템, 설비, 사람 등 다양한 곳에서 발생할 수 있다.

복합 이벤트 처리에서 중요한 것은 신속한 대응을 위하여 이벤트 패턴을 파악하고 복합 이벤트 패턴을 검출하는 것이며 기업 내에서 이벤트 처리 기술을 적용할 경우 간단한 이벤트 주도적 아키텍처(Event-Driven Architecture)[4][5]를 활용해 유연한 애플리케이션 개발 및 신속한 비즈니스 대응을 할 수 있다. 또한 보다 복잡한 이벤트 처리를 활용해 비즈니스 통찰력과 상황 인식을 통해 더 똑똑한 의사결정을 할 수 있다.

복합 이벤트 처리를 도입할 경우 신규 기회, 예를 들어 신규 고객 확보를 목표로 할 경우 높은 투자수익률을 거둘 수 있고, 프로세스 개선 및 최적화를 목적으로 할 경우 비용 절감과 프로세스 개선 효과 등 실제적인 효과를 얻을 수 있다. 또한 변화를 반영하기 위한 시스템의 변경 작업을 쉽고 빠르게 해 전체적인 총소유 비용을 낮추는 효과를 볼 수 있다.

복합 이벤트 처리에서는 각 이벤트의 상관관계가 중요하다. 이벤트 상관관계에는 이벤트 필터링, 이벤트 응집, 이벤트 근본 원인분석 등이 있으며, 원시 이벤트에서 발생하는 수많은 이벤트 스트림이나 이벤트 클라우드에서 이벤트를 감지하여 포착하고 불필요한 이벤트는 제거하고 유의미한 이벤트는 받아들여 복합 이벤트를 생성한다. 또한, 이벤트 상관관계에 시제적인 제약조건이 주어질 수 있으며, 시제적인 제약조건에는 시점기반 이벤트와 기간기반 이벤트가 있다.

논문의 구성은 다음과 같다. 2장에서는 본 연구의 배경과 관련 연구에 대하여 설명하고, 3장에서는 기간기반의 복합 이벤트 패턴 검출에 대하여 기술하고, 4장에서는 성능평가 방법과 실험결과에 대하여 기술하고, 5장에서는 결론 및 향후 연구과제에 대하여 기술한다.

Ⅱ. 관련 연구

1. 이벤트

가. 단순 이벤트 처리

발생한 이벤트들은 모두 유의미한 이벤트로 간주하고 각각의 이벤트 내용에 따라 액션을 수행하며 publish/subscribe 방식이나 중재 방식에 의해 이벤트 처리를 제공한다.

나. 복합 이벤트 처리

여러 이벤트 소스로부터 발생한 이벤트를 대상으로 이벤트들이 영향을 분석하여 대응되는 액션을 수행한다. 단순 이벤트 처리가 하나의 이벤트를 대상으로 한 반면, 복합 이벤트 처리는 여러 이벤트간의 다양한 관계를 분석한다. 복합 이벤트 처리는 BAM 패키지에 내장되고 제공되거나 별도의 복합 이벤트 처리 시스템으로 제공될 수도 있다[6].

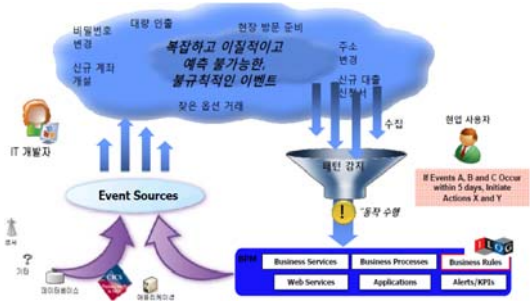

그림 1. 다양한 소스에서 복합 이벤트 처리
Fig. 1. Complex Event Processing in the various source

복합 이벤트 처리는 기업에서 발생하는 복잡한 이벤트들을 탐지하고 관리하기 위해 필요한 기술이다. 복합 이벤트 처리의 목적 중 하나는 복잡한 비즈니스 환경에서 수 없이 발생하는 이벤트들에 대한 이해를 돕는 것이다. 그림 1에서, 복합 이벤트 처리를 통해서 특정 이벤트가 무엇에 의해서 발생했는지 알 수 있고, 그것을 바탕으로 대응하는 룰을 만들어 실행에 옮길 수 있다. 결국, 복합 이벤트 처리는 어떠한 액션을 행할 수 있는 기본적인 바탕을 제공한다[3].

2. 이벤트 상태 표현

상태를 변화시키는 객체인 이벤트는 상태변화를 NFA(Nondeterminstic Finite Automata)로 표현할 수 있다[7][8]. 이벤트 A, B, C, D에 대한 시퀀스(A, B, D)에 대한 NFA와 런타임 스택의 시퀀스 스캔(SC; Sequence Scan)과 시퀀스 구성(SS; Sequence Construct)은 그림 2와 같다[8]. 그림 2에서, 이벤트 a1, c2, b3, a4가 도착하면 시퀀스 스캔에 의해 이벤트들은 런타임 스택에 저장되며, d5가 도착하면 시퀀스 구성에 의해 SEQ(a1, b3, d5)의 중간 결과를 구성한다. 이벤트 d7이 도착하면 SEQ(a1, b3, d7), SEQ(a1, b6, d7), SEQ(a4, b6, d7)의 중간 결과를 구성한다. 또한, 이벤트 d9가 도착하면 SEQ(a1, b3, d9), SEQ(a1, b6, d9), SEQ(a4, b6, d9)의 중간 결과를 구성한다.

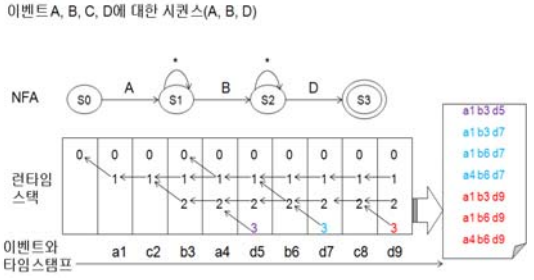

그림 2. 시퀀스 스캔과 구성
Fig. 2. Sequence Scan and Construction

3. 활성 인스턴스 스택을 이용한 시퀀스 스캔 구성

NFA와 활성 인스턴스 스택(Active Instance Stack)을 이용한 시퀀스 스캔 및 시퀀스 구성은 그림 3과 같다.

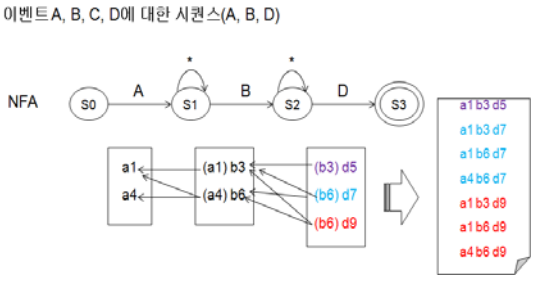

그림 3. 활성 인스턴스 스택을 이용한 SSC
Fig. 3. SSC using Active Instance Stack

기본적인 런타임 스택을 사용하는 것보다 활성 인스턴스 스택을 사용하는 이유는 스트림의 개수나 윈도우 크기(W)가 증가함에 따라 런타임 스택이 활성 인스턴스 스택을 사용하는 것보다 처리율이 감소하기 때문이다. 그 이유는 런타임 스택에서, 시퀀스 구성은 시퀀스 길이 및 윈도우 크기에 따른 비용을 초래한다. 하지만 활성 인스턴스 스택을 사용에 따른 비용은 시퀀스 길이(L)에 비례하며, 런타임 스택은 윈도우의 크기가 증가함에 따라 메모리 오버헤드가 발생한다. 하지만 활성 인스턴스 스택은 가장 최근의 이벤트의 인덱스를 가짐으로서 런타임 스택에서 사용하는 추가적인 메모리 사용을 줄이게 됨으로서 메모리 오버헤드를 줄일 수 있다.

Ⅲ. 기간기반의 복합 이벤트 패턴 검출

1. 전체 프레임워크

본 논문에서 제안하는 복합 이벤트 검출을 처리하기 위한 전체적인 구조는 그림 4와 같다. SOA 기반 서비스는 다수의 사용자, RFID 리더기, 다양한 센서들 그리고 모바일 폰(스마트 폰)등 입력받은 대용량데이터를 수집한다[9]. 이러한 데이터들은 다양한 프로토콜을 지원하는 ESB을 통해 EDA에 전달한다. EDA는 연속적인 요청 및 질의를 받게 되고, 연속적인 질의는 이벤트 생성기를 통해 이벤트가 생성되고 이벤트 엔진에 이들을 전달한다.

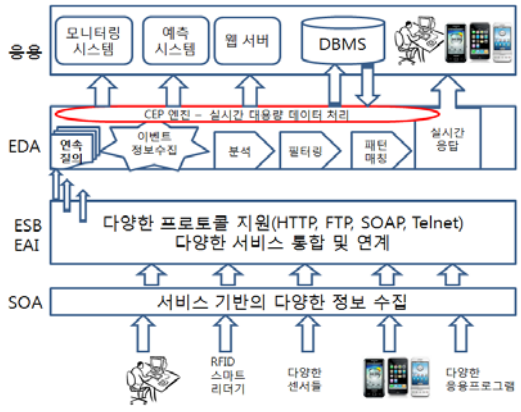

그림 4. 복합 이벤트 처리를 위한 프레임워크
Fig. 4. Framework for Complex Event Processing

2. 이벤트 처리 과정

이벤트 처리 과정은 전사적인 관점에서 그림 5와 같이 크게 3단계로 나눌 수 있다. 첫째 이벤트들을 받는 단계, 둘째 이벤트들을 받고 분석하고 걸러내는 단계, 최종적으로 다시 새로운 액션으로 연결시키는 단계로 나눌 수 있다.

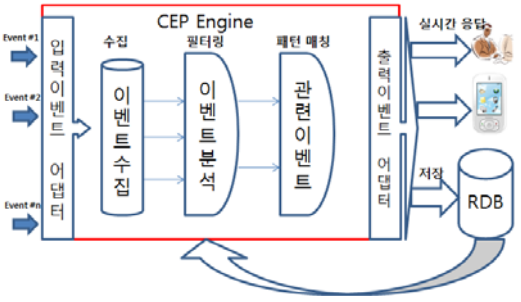

그림 5. 이벤트 처리 과정
Fig. 5. Event Processing

3. 시점기반 이벤트

시점기반 이벤트는 단일 시점에서의 이벤트 발생을 나타내며, 매우 짧은 시점에서 유효한 이벤트를 말하고, 기간기반 이벤트는 시작과 종료시점이 존재하는 일정 기간 동안 유효한 이벤트를 말한다[7]. 시점기반 이벤트와 기간기반 이벤트의 특징을 정리하면 표 1과 같다.

표 1. 시점기반 이벤트와 기간기반 이벤트 비교
Table 1. Comparison over point event and interal event

속성	시점 이벤트	기간 이벤트
타임스탬프	시작 시점	시작 시점과 종료 시점
종료시점	알 수 없음	알 수 있음
유효시간	매우 짧은 시간	특정 시간의 기간
가능한 표현	before, after, equal	Allen의 13개 시제
이벤트의 예	웹 클릭, 주식 시세, 윈도우 로그, 전자메일 도착 등	경매의 입찰 기간, 주식의 호가가 특정 기간 동안 유효한 시세 등

가. Interval_SEQ 연산자과 끝점(endpoint) 관계

이벤트 기간 사이의 시제적인 관계를 표현할 수 있는 끝점 기반의 기호화 메커니즘의 기본 생각은 시제적인 제한(TR; Temporal Restriction)을 and 연산자로 연결하는 것이다. 이러한 and 연산자로 연결한 표현을 시제적인 표현 목록(TRList)라고 하고 각 끝점간의 관계를 표현한 연산자를 Interval_SEQ라 한다. Interval_SEQ는 시제적인 연산자이며, 아래와 같이 표현한다.

$$\text{Interval\_SEQ}[\text{TRList}](E_1,E_2,\ldots,E_m\ ;\ W)[H] = \{< e_1\ e_2\ \ldots\ e_m > \mid (\text{TRList}(e_1, e_2, \ldots, e_m)) \wedge (< e_1\ e_2\ \ldots\ e_m > \in E_1[H] \times E_2[H] \ldots \times E_m[H]) \wedge (\max(e_i.\text{end\_timestamp})_{i\in\{1,2,\ldots,m\}} - \min(e_j.\text{start\_timestamp})_{j\in\{1,2,\ldots,m\}} < W)\}.$$

"이벤트 A는 이벤트 B 이전에 발생한다"의 표현은 시제적인 관계 R에 대한 TRList는 "$A^- < B^-$"로 나타내며, 이는 "(A before B) ∨ (A meets B) ∨ (A overlap) ∨ (A finished by B) ∨ (A contains B)"와 같다. (A^-는 이벤트 A의 출발 끝점, A^+는 이벤트 A의 종료 끝점이다)

나. 기간 이벤트 도착

그림 6에서 이벤트 패턴 질의 Q = Interval_SEQ[$A^- < B^+ < C^+ < D^+$](A, B, C, D)이며 기간 이벤트 추적 H = "$b^{3|6}$, $d^{6|8}$, $b^{7|9}$, $c^{4|10}$, $a^{7|10}$, $d^{9|12}$, $b1^{0|13}$, $d^{13|16}$"이다.

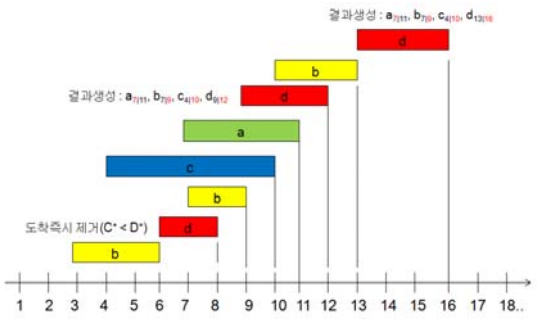

그림 6. 타임 라인에 도착하는 기간 이벤트
Fig. 6. Interval Event reaching in the time line

다. 활성 인스턴스 스택을 이용한 시퀀스 스캔 및 구성

그림 7에서, 도착순서는 종료 끝점을 기준으로 하며, 기간 이벤트 인스턴스 $b^{3|6}$이 도착하면 상태 S2에 대한 활성 인스턴스 스택에 저장되며 이벤트 인스턴스 a를 가 가리키는 인덱스는 null 값을 가진다. $d^{6|8}$은 도착과 동시에 스택에 저장하지 않고 즉시 제거시킨다. 그 이유는 이벤트 패턴 $[A^- < B^+ < C^+ < D^+]$를 만족하지 않기 때문이다.

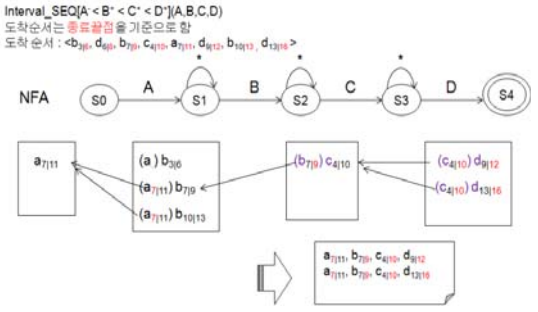

그림 7. 활성 인스턴스 스택을 이용한 SSC
Fig. 7. SSC using Activity Instance Stack

그림 7에서, $b^{7|9}$가 도착하면 상태 S2에 대한 활성 인스턴스 스택에 저장되며 이벤트 인스턴스 a를 가리키는 인덱스는 null 값을 가진다. $c^{4|10}$은 상태 S3에 대한 활성 인스턴스 스택에 저장되며 이벤트 인스턴스 b를 가리키는 인덱스는 $b^{7|9}$가 되며 b의 종료 끝점 9를 가리키게 된다. a7|11이 도착하면 상태 S1에 대한 활성 인스턴스 스택에 저장되며 먼저 도착한 $b^{7|9}$는 a를 가리키는 인덱스 $a^{7|11}$을 가지며 a의 시작 끝점 7을 가리키게 된다. $d^{9|12}$가 도착하면 시퀀스 구성($a^{7|11}$, b7|9, $c^{4|10}$, $d^{9|12}$)이 이루어진다. $d^{13|16}$이 도착하면 또 다른 결과($a^{7|11}$, $b^{7|9}$, $c^{4|10}$, $d^{13|16}$)를 구성하게 된다.

활성 인스턴스 스택을 사용한 기간 이벤트의 스퀀스 스캔과 시퀀스 구성 알고리즘은 그림 8과 같다. 그림 8에서, 라인 1은 각 도착한 이벤트에 대한 양 끝점(시작 끝점과 종료 끝점)에 연산을 수행하는 부분으로 도착한 이벤트가 시작 끝점인지 종료 끝점인지를 파악한다. 라인 2는 도착하는 이벤트가 스트림의 끝인지 확인한다. 라인 3은 도착한 이벤트와 이전에 저장된 가장 작은 시작 타임스탬프의 차가 윈도우 크기와 비교한다. 라인 4는 이벤트를 스택에 저장하고, 라인 5는 가장 최근에 스택에 저장된 이벤트의 인덱스를 세팅한다. 라인 6에서 13까지는 시퀀스 구성을 나타낸다.

```
// Interval_SEQ[TRList](e1, e2,...en) :n은 시퀀스의 길이
// 새롭게 도착한 이벤트를 e라 한다.
// 이벤트 e는 (e1, e2, ...en) 중에 하나임
01 endpoint(TRList)  // 양 끝점 관계를 수행
02 while( not EOS) //EOS는 End Of Stream
03     if(e.ts - e.ts(min) < W) //W는 윈도우 크기
04         push(Si, e) // Si는 i 번째 상태, 이벤트를 스택에 저장
05         set RIP /* RIP는 이전 상태의 스택에 대한 가장 최근에  저장된
이벤트로 인덱스로서 사용함*/
06         if(e.final_state==true)
07             if(e.endpoint > b_e.endpoint)  // b_e는 이전 상태의 이벤트
08                 if(b_e is not buffered) // 스택에 저장된 이전 이벤트 없음
09                     purge(e) // 도착한 이벤트 제거
10                 else
11                     SC <- e// 시퀀스 구성
12                     pop(e)  // 구성한 시퀀스를 출력
13                 end if
14         else
15             e -> SS // 스퀀스 스캔
16         end if
17     else
18         purge(e.ts(min)) // 스택에 저장된 시작 끝점이 최소인 이벤트 제거
19     end if
20 end while
```

그림 8. 스퀀스 스캔과 시퀀스 구성 알고리즘
Fig. 8. Algorithm for sequence scan and construction

Ⅳ. 실험 및 결과

복합 이벤트 패턴 검출은 이벤트들 사이의 시제적인 상관관계가 많은 영향을 미친다. 본 실험은 CPU 2.6GHz,

메모리 4G에서 실시하였으며, 제안된 기간 이벤트 스트림을 처리하기 위해 입력 데이터로, 20개의 서로 다른 이벤트를 생성하였다. 스트림의 개수는 10만에서 100만개를 사용한다. 적용된 질의들은 성능평가를 위해 두 가지 항목을 비교 실험한다.

1) 런타임 스택을 이용한 Interval_SEQ(naive)
2) 활성 인스턴스 스택의 인덱스를 가진 Interval_SEQ (index)

1. 질의의 길이에 따른 측정

이 실험은 질의의 길이가 기간기반 복합 이벤트 처리에 얼마나 영향을 미치는지 측정한다. 본 실험에서는 질의의 길이는 2에서 10이다. 끝점 인덱스는 시작 끝점과 종료 끝점이 각각 5:5 비율이다. 예를 들어 시퀀스의 길이가 6이라면 Interval_SEQ[A^- < B^- < C^- < D^+ < E^+ < F^+](A, B, C, D, E, F)가 수행된다.

가. 메모리 소비량

그림 9에서, x축은 질의 길이를 나타내며, y축은 각 이벤트에 대한 누적된 메모리 소모량을 나타낸다. 질의의 길이가 증가할수록 인덱스 접근방법이 naive 접근방법보다 메모리 소비량이 효율적임을 알 수 있다.

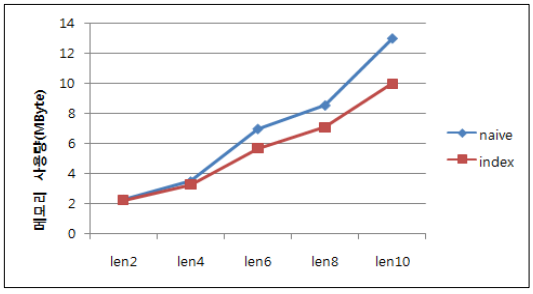

그림 9. 질의의 길이에 따른 메모리 소비량
Fig. 9. Memory consumption over query length

나. CPU 수행시간

그림 10에서 x축은 질의의 길이를 나타내며, y축은 각 질의에 대한 수행시간을 나타낸다. 좀 더 긴 길이를 가진 질의는 CPU 자원을 더 많이 사용한다.

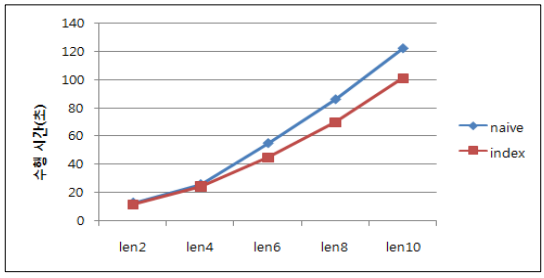

그림 10. 질의의 길이에 따른 수행시간 측정
Fig. 10. Runtime Measure over Query Length

2. 윈도우 크기에 따른 이벤트 처리 수

표 2는 윈도우의 크기에 따른 이벤트 패턴 검출의 개수와 원시 알고리즘과 활성 인스턴스 스택의 인덱스를 이용한 알고리즘의 초당 이벤트 처리 개수를 나타낸다. 윈도우의 크기가 커질수록 이벤트 패턴 검출의 수는 증가하며 초당 처리할 수 있는 이벤트의 처리 개수는 줄어든다.

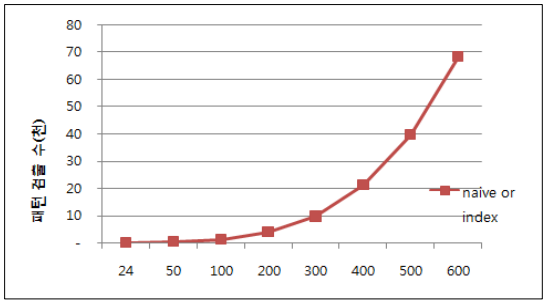

그림 11. 윈도우 크기에 따른 패턴 검출 개수
Fig. 11. Pattern detection count about window size

표 2. 윈도우 크기와 이벤트 패턴 검출
Table 2. Windows Size and Event Pattern Detection

윈도우 (단위:초)	패턴 검출 수	초당이벤트 처리수 (naive)	초당이벤트 처리수(index)
24	36	658,742	668,742
50	300	430,267	601,239
100	1,239	351,600	532,579
200	3,965	276,095	493,254
300	9,613	202,398	444,995
400	21,146	162,004	365,687
500	39,500	103,241	305,421
600	68,326	80,021	268,712

그림 11은 표 2의 이벤트 패턴 검출 개수를 그래프로 나타내고 있으며 원시 알고리즘과 활성 인스턴스 스택을 사용한 알고리즘 모드 같은 수의 패턴을 검출한다. 그림 10은 표 2의 초당 이벤트 처리 개수를 나타내고 있다. 초당 처리 개수는 활성 인스턴스 스택의 인덱스를 사용한 알고리즘이 원시 알고리즘보다 처리율이 높으며, 윈도우의 크기가 커질수록 원시 알고리즘에 비해 좀 더 완만한 곡선을 그린다. 스트림의 개수 S = 10만개, 시퀀스의 길이 L = 6, 이벤트 당 평균 기간 I = 4이며 시작 끝점과 종료 끝점의 비율은 5:5이다.

표 2에서, 윈도우의 크기가 증가함에 따라 인덱스를 사용한 이벤트 패턴 검출은 원시 이벤트 패턴 검출보다 초당 이벤트 처리에서 최대 60% 이상의 효율을 가진다.

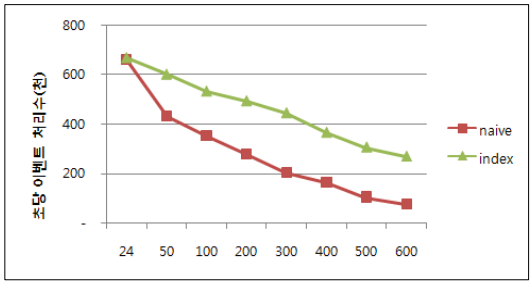

그림 12. 윈도우 크기에 따른 초당 이벤트 처리 개수
Fig. 12. Event processing count over window size per second

그림 12에서, 런타임 스택(naive)을 사용한 알고리즘의 복잡도는 O(LS)이다. L은 시퀀스의 길이(NFA에서 상태의 개수)이며, S는 이벤트 개수(스트림의 개수)이다. 동적으로 윈도우 제약조건을 사용할 경우 S는 W로 설정할 수 있으므로 복잡도는 O(LW)가 된다.

시퀀스 구성에서, 런타임 스택 알고리즘은 윈도우 크기 W에 선형적인 비용을 초래하는 반면, 활성 인스턴스 스택의 인덱스를 사용한 알고리즘은 스택의 깊이(Depth) 만큼의 비용을 초래한다. 그러므로 활성 인스턴스 스택의 인덱스를 사용하는 알고리즘의 복잡도는 O(LD)이다.

V. 결론 및 향후 연구과제

본 논문에서는 기간기반(Interval-based)의 시제적인 제약조건을 부여하여 이벤트들 사이의 상관관계를 유추함으로서 시스템의 성능을 향상시킬 수 있는 기간기반 복합 이벤트 패턴을 검출하는 방법론을 제시하였다. 기존의 시점기반 복합 이벤트 처리는 연속적인 시퀀스 이벤트 처리에는 적합하지만, 이벤트 사이의 겹침을 표현할 수 없으며, 이벤트 사이의 포함관계를 처리하기에는 불충분하다. 기간기반 복합 이벤트 처리는 시점기반 복합 이벤트 처리의 문제점을 보완하며 좀 더 유연한 복합 이벤트 처리를 수행할 수 있다.

이전에 연구된 시점기반 복합 이벤트 처리는 각 이벤트에 하나의 타임스탬프를 사용하여 즉각적인 이벤트를 처리하였다. 하지만, 시점기반의 이벤트 처리로는 이벤트의 활동 기간이 중요한 역할을 하는 금융, 멀티미디어, 의학, 기상학 같은 분야에서 복합적 시제 관계를 표현하기에는 불충분하다. 본 논문에서는 시점기반의 복합 이벤트 처리의 단점을 보완한 기간기반의 복합 벤트 처리를 제안하였다. 많은 양의 스트림에서 입력되는 이벤트에 기존의 타임스탬프를 사용하는 대신에, 각 이벤트에 시작 타임스탬프와 종료 타임스탬프를 부여하여 이벤트 사이의 겹침이 발생하거나 이벤트가 다른 이벤트를 포함하는 경우, 이를 해결하는 방안을 제시하였다.

본 논문에서는 복합 이벤트 패턴 검출을 최적화하기 위해 활성 인스턴스 스택을 사용하는 알고리즘을 제안하였으며, 이벤트의 시퀀스를 구성할 때 중간 결과의 수를 줄이기 위해 윈도우 푸시다운 기법을 적용하여 수행시간과 메모리의 효율을 높인다. 실험 결과는 제안한 알고리즘이 원시 알고리즘보다 효율적임을 알 수 있다.

향후 연구과제로는 네트워크 지연 등에 의해 발생할 수 있는 순서가 잘못된 이벤트를 포착하여 올바르게 순서화 시키는 메커니즘에 대한 연구가 필요하다. 항상 순서적으로 이벤트가 도착함을 보장할 수 없으며, 이를 순서화 시켜 손실이 없는 복합 이벤트를 검출하는 것은 중요하다. 또한, 시점 이벤트와 기간 이벤트를 동시에 처리할 수 있는 하이브리드 복합 이벤트 패턴을 검출할 수 있는 연구가 필요하다.

참 고 문 헌

[1] David Luckham. "The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems." Pearson

Education, Inc. 2002
[2] Alves A. "Extensions to logic programming inference engines to support cep." RuleML. 2009
[3] Chen S.K, Jeng J.J, Chang, "Complex Event Processing using Simple Rule-based Event Correlation Engines for Business Performance Management." In: CEC/EEE 2006
[4] Schiefer J, Szabolcs R, Saurer G, Rauscher C. "Event-Driven Rules for Sensing and Responding to Business Situations." International Conference on Distributed Event-Based Systems, Toronto 2007
[5] Paschke A, Kozlenkov A, Boley H. "A homogenous reaction rules language for complex event processing." International Workshop on Event Drive Architecture for Complex Event Process. ACM, New York 2007
[6] Barga R.S, Goldstein, J, Ali M.H, Hong M. "Consistent streaming through time: A vision for event stream processing." CIDR 2007
[7] James F. Allen. "Maintaining knowledge about temporal intervals. Commun." ACM. pages 832-843. 1983.
[8] Eugene Wu, Yanlei Diao, and Shariq Rizvi. "High-performance complex event processing over streams." In SIGMOD. pages 407-418. 2006.
[9] ManMo Kang, Jarok Koo, DongHyeong Lee, "High-Volume Data Processing using Complex Event Processing Engine in the Web of Next Generation", Korea Institute of Information Scientists and Engineers, Database Vol.37 No. 6. pages 300-307, 2010

저자 소개

강 만 모(정회원)

- 1998 울산대학교 전자계산학과 학사 졸업
- 2000 울산대학교 전자계산학과 석사 졸업
- 2011 울산대학교 정보통신공학 박사 졸업
- 2006~2009 울산대학교 객원교수
- 2009 ~현재 대광산업 기술연구소 책임연구원, 울산대학교 전기공학부 강사

<주관심분야 : 멀티에이전트, 소프트웨어공학, 빅데이터 분석>

박 상 무(정회원)

- 1995 울산대학교 컴퓨터공학과 학사 졸업
- 1997 울산대학교 컴퓨터공학과 석사 졸업
- 2010 울산대학교 컴퓨터공학과 박사 졸업
- 2011~현재 울산대학교 전기공학부 객원교수

<주관심분야: 인공지능, 신경회로망, 임베디드 시스템>

김 상 락(정회원)

- 2012 울산대학교 정보통신공학 박사 졸업
- 2010 ~ 현재 비케이앤씨 대표, 울산대학교 전기공학부 강사
- 2000 ~ 2009 (주)아이티스타 연구소장

<주관심분야 : SLA, 분산병렬처리시스템, 인포그라픽스, 빅데이터>

김 강 현(정회원)

- 1994 경성대학교 전산통계학과 학사 졸업
- 1996 경성대학교 전자계산학 석사 졸업
- 1999 울산대학교 컴퓨터공학 박사 수료
- 1999~2000 영산대학교 전임강사
- 2001 ~현재 한국폴리텍Ⅶ대학 부교수

<주관심분야 : 유니파이드 커뮤니케이션, 멀티미디어 데이타베이스>

이 동 형(정회원)

- 1996 울산대학교 컴퓨터공학과 학사 졸업
- 1998 울산대학교 컴퓨터공학과 석사 졸업
- 2009 울산대학교 컴퓨터공학과 박사 졸업
- 2001 ~ 현재 한국폴리텍VII대학 울산 캠퍼스 정보통신시스템과 부교수

<주관심분야 : 네트워크, 신경망, 지능형 로봇>