# PSR : 효율적인 웹 서비스 컴포지션 검색을 위한 RDBMS 기반의 선 계산 기법

## (PSR: Pre-Computing Solutions in RDBMS for Efficient Web Services Composition Search)

권 준 호 † 박 규 호 †† 이 대 욱 ††† 이 석 호 ††††
(Joonho Kwon) (Kyuho Park) (Daewook Lee) (Sukho Lee)

**요 약** 최근 웹 서비스 컴포지션이 많은 관심을 받고 있다. 웹 서비스 컴포지션을 통해서, 현재 존재하는 웹 서비스를 이용하여 저장소에 존재하지 않는 새로운 웹 서비스를 제공할 수 있다. 이 논문에서는 웹 서비스 컴포지션 검색을 구성하기 위해서 관계형 데이타베이스(RDBMS)를 사용한 PSR 시스템을 제안한다. 웹 서비스 컴포지션을 선 계산하기 위한 조인과 인덱스를 사용하는 알고리즘을 제안한다. 또한 웹 서비스에서 추출한 온톨로지 정보도 테이블로 저장하고, PSR 시스템이 이를 사용하여 온톨로지 매칭 정도에 따라 사용자 질의와의 유사도를 통한 결과를 반환하도록 하였다. 실험을 통하여 RDBMS 상에서 웹 서비스 컴포지션을 선 계산하는 접근 방법이 많은 수의 웹 서비스와 사용자 질의를 처리할 때 빠른 실행 시간과 좋은 확장성을 가지고 있음을 보였다.

**키워드** : RDBMS, 웹 서비스, 웹 서비스 컴포지션 검색

***Abstract*** In recent years, the web services composition has received much attention. By web services composition, we mean providing a new service that does not exist on the repository. In this paper, we propose a new system called PSR for web services composition search using a relational database. We also propose algorithms for pre-computing web services composition using joins and indices. We store ontologies from web services in RDBMS, so that the PSR system returns web services composition in order of similarity with user query through the degree of the ontology matching. We demonstrated that our pre-computing web services composition approach in RDBMS yields lower execution time and good scalability when handling a large number of web services and user queries.

**Key words** : RDBMS, Web services, Web services composition search

· 본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터 육성·지원 사업(IITA-2008-C1090-0801-0031)의 연구결과로 수행되었음

† 학생회원 : 서울대학교 전기컴퓨터공학부
joonho@db.snu.ac.kr
†† 비 회 원 : 해군 훈련소 소위
parkqho@db.snu.ac.kr
††† 비 회 원 : 서울대학교 전기컴퓨터공학부
dwlee@db.snu.ac.kr
†††† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
shlee@snu.ac.kr

논문접수 : 2007년 5월 29일
심사완료 : 2008년 3월 3일



정보과학회논문지 : 데이타베이스 제35권 제4호(2008.8)

## 1. 서 론

최근 인터넷에 대한 수요가 증대하고 관련 기술이 발전하면서 웹에 대한 관심이 높아졌다. 웹과 관련된 여러 기술 중에서 웹 서비스(web services)와 같은 기술이 주목을 받고 있다. 웹과 관련하여 다양한 기술들이 나오게 되었는데 그 중 e-비즈니스와 웹 서비스 같은 전자상거래 기술이 주목을 받고 있다. 이 기술들은 웹 상에 e-비즈니스나 서비스 객체를 존재하게 하여 사용자가 원하는 기능을 수행하도록 한다. 특히 웹 서비스는 웹 서비스 제공자에 의해 UDDI(Universal Description Discovery Integration) 레지스트리[1]에 등록되고, 웹 서비스 요청자(사용자)가 검색을 요청하면 해당 웹 서비스가 검색된다. 그림 1은 그 구조를 나타내고 있다[2].

각각의 웹 서비스의 기능과 인터페이스들은 WSDL

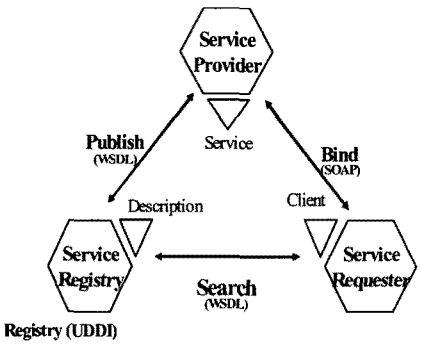



그림 1 웹 서비스 구조

```
WS1:  Web Service : PlaceLookup
      Operation : CityToZipCode
            Input   : City
            Output  : Zipcode
WS2:  Web Service : WeatherByZipcode
      Operation : WeatherInfo
            Input   : Zipcode
            Output  : Weather
WS3:  Web Service : TemperatureFetcher
      Operation : GetTemperature
            Input   : PostCode
            Output  : Temperature
WS4:  Web Service : SportsStatusViewer
      Operation : SportsByWeather
            Input   : Weather
            Output  : SportsOK
```

그림 2 웹 서비스 예제

(Web Services Description Language)[3]로 기술되며, WSDL 파일과 간단한 설명을 UDDI 레지스트리에 등록을 함으로써 웹 서비스를 공개한다. 그림 1은 웹 서비스의 구조를 보여준다. 웹 서비스의 정보는 웹 서비스의 이름, 해당 기능을 수행하는 연산(operation)과 그의 입력(input) 및 출력(output)으로 구성된다.

대다수의 웹 서비스 관련 연구는 웹 서비스를 효율적으로 저장하고 검색하는 방법을 제안하였다[4-6]. 이런 연구들은 하나의 독립된 웹 서비스를 대상으로 하는 연구이다. 최근, 하나의 독립된 웹 서비스만을 검색하는 것을 보완하기 위해서 웹 서비스 컴포지션 검색에 대한 연구들도 진행되었다. 웹 서비스 컴포지션 검색은 어떤 웹 서비스의 출력과 다른 웹 서비스의 입력이 같거나 비슷한 경우 전자의 출력과 후자의 입력 사이에 링크(link)를 만들어, 그 웹 서비스들을 연결시킴으로써 마치 하나의 독립된 웹 서비스처럼 사용자에게 결과를 반환한다. 웹서비스 컴포지션 검색은 사용자가 명세한 입력과 출력을 가진 웹 서비스가 UDDI에 등록되어 있지 않더라도 등록된 다른 웹 서비스들을 조합함으로써 사용자가 원하는 결과를 반환할 수 있어서 일반적인 웹 서비스 검색보다 더 유용하다.

예를 들어, 그림 2의 여러 웹 서비스 중에서 사용자가 입력이 "City"이고 출력이 "Weather"인 웹 서비스를 찾고자 한다고 가정하자. 웹 서비스 컴포지션 검색을 지원하지 않는다면, 사용자 질의에 해당하는 웹 서비스는 존재하지 않는다. 그러나 만약 웹 서비스 컴포지션 검색을 사용한다면, 컴포지션 검색의 결과로 웹 서비스의 시퀀스를 사용자에게 반환할 수 있다. 예를 들어, WS1의 입력은 "City"이고 WS2의 출력은 "Weather"이며, WS1의 출력과 WS2의 입력은 "Zipcode"로 같은 값을 가짐을 알 수 있다. 이런 경우 웹 서비스 시퀀스 <WS1, WS2>을 사용자 질의의 결과로 반환한다.

웹 서비스 컴포지션을 위한 연구들은 메인 메모리 알고리즘을 사용한다[7-11]. 이런 메인 메모리 기반의 알고리즘의 확장성은 이용 가능한 메모리의 양에 제약을 받고, 사용자가 결과로 원하는 웹 서비스의 입력과 출력을 다르게 줄 경우 매번 웹 서비스 컴포지션을 위한 구조를 계산해야 한다는 단점이 있다.

시맨틱 매칭 개념을 웹 서비스 컴포지션 검색에 추가한다면 사용자에게 더 유용한 검색 결과를 돌려줄 수 있다. 예를 들어 "Zipcode"와 "Postcode"가 같은 의미를 가지고 있고, "Weather"와 "Temperature"가 비슷한 의미를 가지고 있다는 온톨로지 정보가 시스템에 더 유지되고 있다면, <WS1, WS3>도 웹 서비스 컴포지션 검색의 결과로 반환할 수 있지만 온톨리지 정보가 없다면 그 시퀀스는 결과로 나타나지 않을 것이다.

이런 시맨틱 매칭 개념을 사용한 연구에는 클러스터링 기법[4]을 사용하거나 온톨로지에 기반을 둔 연구[5,6]가 있다. 이 또한 WSDL 파일 저장 구조에 부가적인 구조를 유지해야 하고, 단일 웹 서비스만을 검색하거나 웹 서비스들 간의 매칭 문제만을 주로 다루었다는 단점이 있다.

이 논문에서는 웹 서비스 컴포지션 검색을 구성하기 위해서 앞의 단점을 개선하기 위하여 관계형 데이타베이스에 기반을 하여 가능한 웹 서비스 컴포지션을 선계산하여 저장하는 PSR(Pre-Computing Solutions in RDBMS) 시스템을 제안한다. PSR 시스템에서는 웹 서비스 컴포지션 검색에 의미를 추가하기 위하여 온톨로지 정보를 사용하는데 이 온톨로지도 RDBMS에 저장한다. 이 PSR 시스템의 중요한 특징을 다음과 같이 요약할 수 있다.

첫째, 관계형 데이타베이스에서 동작하는 웹 서비스 컴포지션 검색 시스템을 제안했다는 것이다. 기존의 연구들과 같이 웹 서비스들을 정점(웹 서비스)과 간선들로 이루어진 그래프로 가정한다는 점은 동일하지만, 입력으로 들어온 웹 서비스(그래프의 정점)를 먼저 테이블 형태의 데이타로 저장하여 사용한다는 점이 다르다.

둘째, 테이블 형태로 저장한 웹 서비스를 이용하여 웹 서비스 컴포지션을 선 계산하는 두 가지 알고리즘과 검색을 위한 알고리즘을 제시하였다. 이 알고리즘 수행 중에 나오는 중간 결과들도 테이블 형태로 저장하여 사용한다. 웹 서비스 컴포지션 계산과 검색 알고리즘들은 SQL 문장들만을 사용하여 수행한다.

셋째, 온톨로지 정보도 테이블 형태로 저장하여, 웹 서비스의 컴포지션 선 계산 알고리즘에서 이 테이블을 조인 연산에 포함하여 사용한다. 또한 입출력과 매칭하는 온톨로지 개념의 유사도에 따른 가중치 정보를 이용하여 웹 서비스 컴포지션 검색에 활용한다.

이 논문의 구성은 다음과 같다. 2장에서 웹 서비스 컴포지션 관련 연구와 동기를 설명한다. 3장에서는 PSR 시스템의 전체 구조를 설명한다. 4장에서 PSR의 핵심 알고리즘인 RDBMS 기반의 웹 서비스 컴포지션 검색을 설명한다. 5장에서 실험 결과에 대해 분석하고, 마지막으로 6장에서 연구 결과를 정리하고 결론을 맺는다.

## 2. 관련 연구

지금까지 제안된 웹 서비스 관련 연구들은 주로 단일 웹 서비스를 검색 대상으로 하는 기법 자체에만 초점이 맞추어져 있었다[4-6]. 즉, 웹 서비스를 하나의 레코드처럼 보고 그 웹 서비스 레코드 하나를 찾는 기법에만 초점이 맞추어져 있었다. 그 중 가장 기본적인 키워드 검색은 이미 오래 전부터 연구되고 많은 발전이 있었다. 사용자에 의해 명세된 입력과 출력을 기반으로 검색하는 템플릿 검색도 키워드를 기반으로 수행된다. 하지만 정확히 일치하는 키워드가 아니면 검색이 되지 않는 한계점이 있다. 이에 여러 연구에서 클러스터링[4]이나 온톨로지[5,6] 같이 의미를 부여하는 시맨틱 웹과 관련된 기술들이 키워드 간 매칭에 이용되었다. 그러나 이들 역시 단일 웹 서비스만을 검색하거나 웹 서비스들 간의 매칭 문제만을 주로 다루었다.

하나의 독립된 웹 서비스만을 검색하는 시스템을 보완하기 위한 웹 서비스 컴포지션 알고리즘 제안하려는 많은 연구들이 행해졌다[7-11]. Colombo[7]는 자동 웹 서비스 컴포지션을 위한 프레임워크이다. 그러나 웹 서비스 컴포지션을 검색 목적으로 한 것이 아니라 웹 서비스들이 실행되어 나온 출력 값(value)을 워크플로우를 따라 넘겨줄 때 실제 웹 서비스 응용 서버 간에 메시징 기법을 사용하는 실시간 전달 기법을 제시했다. [8,9]에서는 웹 서비스 컴포지션 검색 문제를 그래프 검색 문제로 변환하여 풀려고 시도하였다. 웹 서비스들은 정점으로 생각하고 웹 서비스 컴포지션 검색은 정점들 사이에서 경로를 찾는 문제로 생각하였다. 사용자 질의가 요구하는 출력과 동일과 값을 가지는 웹 서비스에 도달할 때까지 깊이 우선 탐색(depth first search)를 반복적으로 수행하였다. [8]의 저자들은 컴포지션을 정렬하기 위하여 웹 서비스의 참조 빈도를 이용하는 PageRank 기법도 제안하였다. 그래프에 기반한 웹 서비스 컴포지션 검색을 [9]와 [10]의 논문에서도 이용하였다. 그러나 웹 서비스의 부가 정보를 이용하거나 웹 서비스를 컴포지션 하는 기준을 유사한 개념이 아닌 QoS를 사용한다.

[8-11]과 같은 논문들의 경우 모두 메인 메모리(main memory)에서 돌아가는 알고리즘(in-memory algorithm)을 사용한다. 웹 서비스 사이의 간선을 인접 행렬로 표시하여 행렬의 곱을 웹 서비스의 개수만큼 N번 하는 깊이 우선 탐색 방법으로 각 웹 서비스들 사이의 모든 가능한 컴포지션을 찾으므로, $O(N^3)$의 시간 복잡도를 보인다. 또한 이런 알고리즘은 웹 서비스의 개수가 물리적 메인 메모리에 저장할 수 있는 양을 넘어설 정도로 많아지게 되면 페이지 I/O의 횟수가 증가하여 성능이 매우 저하되는 단점이 있다. 더욱이 기존의 논문들은 다양한 사용자가 웹 서비스를 검색하기 위해서 서로 다른 입력과 출력을 줄 경우, 매번 다른 입력과 출력을 사용하여 웹 서비스 컴포지션을 위한 그래프를 생성하고 탐색하여 사용자에게 돌려주므로 응답 시간이 매우 길어진다.

## 3. PSR 시스템

이 장에서는 PSR 시스템의 기본 구조를 설명한다. 먼저 이 논문에서 해결하려고 하는 웹 서비스 컴포지션 문제를 정형화한다. 3.2절에서 PSR 시스템의 전체적인 구조를 보이고 PSR 시스템을 구성하는 요소들에 대해 설명한다. 3.3절에서는 이 논문에서 가정하고 있는 온톨로지의 구조를 설명한다.

### 3.1 문제 정의

웹 서비스 컴포지션 검색 문제는 다음과 같이 정의할 수 있다.

> UDDI 레지스트리에 있는 웹 서비스를 가지고 있는 집합 N과 웹 서비스의 입력과 출력을 명세하는 사용자 질의 Q가 주어질 때, 모든 웹 서비스 $ws \in N'$가 웹 서비스 컴포지션을 포함하여 Q에 매칭하는 서브셋 $N' \subseteq N$을 찾아라.

웹 서비스 컴포지션 검색 문제를 가중치를 가진, 방향 그래프를 사용하여 선 계산할 수 있다. 그래프는 다음과 같이 정의할 수 있다.

> 컴포지션 그래프 G=(V,E)는 가중치 함수 W를 가진 가중치 방향 그래프이다. V는 웹 서비스의 집합이고, E는 방향을 가진 링크(간선)이다. 웹서비스 $WS_x$의 출력이 웹서비스 $WS_y$의 입력과 같을 때, 웹 서비스 $WS_x$에서 $WS_y$로의 방향 간선이 존재한다. W는 온톨로지에 기반하여 간선들에 실제 가중치 값을 부여하는 가중치 함수이다.

컴포지션 그래프에서 웹 서비스들은 정점으로 간주되고, 한 웹 서비스의 출력이 다른 웹 서비스의 입력과 같으면 두 웹 서비스 사이에 방향 간선이 존재한다. 컴포지션 그래프의 경로에 포함된 중간 정점들은 웹 서비스 컴포지션을 구성하는 웹 서비스들이다.

웹 서비스 컴포지션을 선 계산하는 이 논문의 접근 방법은 컴포지션 그래프를 구성하는 단계와 컴포지션 그래프에서 답을 검색하는 단계로 구성된다. 구성하는 단계에서는 웹 서비스들 사이에서 가능한 모든 경로들을 만들기 위하여 모든 간선들을 이용한다. 검색 단계에서는 각각의 경로의 입력이 사용자 질의의 입력과 일치하고, 그 경로의 출력이 사용자 질의의 출력과도 일치하는지를 검사한다.

### 3.2 PSR 시스템의 구조

이 절에서는 PSR 시스템의 구성 요소들을 설명한다. 그림 3은 PSR 시스템의 전체 구조를 보여준다. 점선으로 표시된 부분이 PSR 시스템의 핵심 엔진이다.

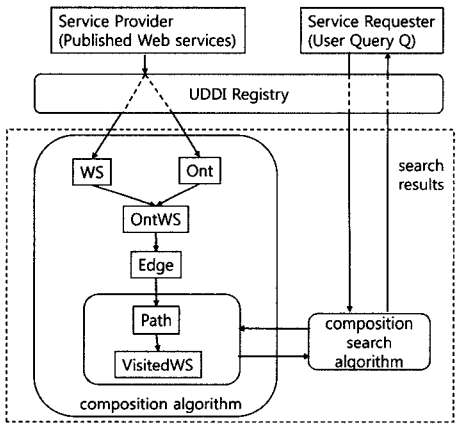



그림 3 PSR 시스템의 구조

서비스 제공자는 UDDI 레지스트리 서버에 웹 서비스를 제공한다. 웹 서비스가 UDDI에 추가될 때, 그 웹 서비스를 명세하는 WSDL 파일을 파싱하고 웹 서비스에 관련된 정보들을 추출한다. 웹 서비스의 연산(operation), 입력(input), 출력(output)들을 관계형 데이타베이스의 테이블에 저장한다. 웹 서비스와 온톨로지를 기반으로 생성한 컴포지션 그래프는 Edge 테이블에 저장한다. 가능한 웹 서비스 컴포지션은 Path와 VisitedWS 테이블에 저장한다. 서비스 요청자가 사용자 질의 Q를 제공하면, 테이블들을 검색한 후에 웹 서비스 컴포지션을 포함한 결과를 서비스 요청자에게 반환한다.

### 3.3 온톨로지 정보

이 논문에서 사용하는 온톨로지는 그림 4와 같이 개념적으로 계층구조를 이루고 분류가 되어 있는 온톨로지이다. 온톨로지는 UDDI 레지스트리에 등록되는 웹 서비스의 입력과 출력 키워드를 기반으로 만들어진다. 그림 4의 온톨로지는 그림 2에 나타난 웹 서비스에 나타난 입출력을 이용하여 생성한 예제이다. 최상위 루트에는 Thing이라는 ClassID가 위치하고, 하위 레벨에는 Ont1에서 Ont5가 위치한다. Ont1 개념에는 Zipcode와 Postcode가 같은 개념을 가지는 키워드로 포함되어 있다. 지금까지 연구된 바로는 수동적(manual)이거나 반자동적(semi-automatic)으로 온톨로지를 생성하는 기술밖에 없기 때문에 본 논문에서도 웹서비스에 추출한 키워드를 이용하여 온톨로지를 수동적으로 구축한다.

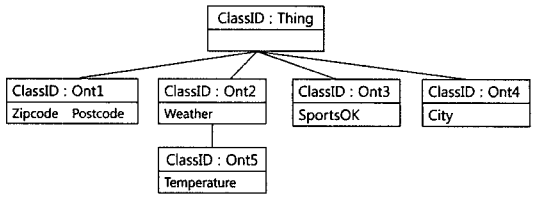



그림 4 온톨로지의 예

온톨로지를 이용하면 키워드는 다르지만 온톨로지 상에서 동일한 클래스 또는 부모(parent) 클래스 및 자식(child) 클래스에 있는 비슷한 개념들끼리 매칭을 할 수 있다. 계층구조 온톨로지를 이용한 매칭은 정확도에 따라 exact, plug-in, subsumes, fail로 나눌 수 있다[6].

exact는 비교되는 키워드들이 온톨로지에서 동일한 클래스에 있는 경우를 뜻한다. plug-in은 기준이 되는 키워드와 비교가 되는 키워드가 순서에 따라 개념적으로 포함되는 경우를 뜻하고 subsumes은 반대로 포함시키는 경우를 뜻한다. 웹 서비스 컴포지션에서는 한 서비스의 출력이 다른 서비스의 입력에 포함되어야 정보 손실이 없으므로 기준 키워드는 출력 키워드가 된다. fail은 클래스의 계층 관계가 없을 경우나 전혀 다른 경우 혹은 2세대 이상 떨어진 클래스에 비교되는 키워드들이 위치해 있을 경우를 뜻한다. exact, plug-in, subsumes까지를 유사한 매칭으로 볼 수 있다.

**예제 1.** Postcode, Zipcode와 zip은 온톨로지 상에서 각각 동일한 클래스에 분류되어 있기 때문에 exact 매칭 관계가 성립된다. 또한 Temperature와 Weather 사이에는 트리에서 조상-후손 계층 구조로 나타나고 있으므로 기준이 되는 개념에 따라서 subsumes 매칭이나 plug-in 매칭 관계로 볼 수 있다.

이 4가지의 매칭 관계의 유사도를 수치화하여 가중치로 줄 수 있다. 매칭 정도가 클수록 적은 가중치를 주고 유사도가 작을수록 큰 가중치를 주면, 사용자에게 결과를 돌려줄 때 유사도의 총합이 가장 작은 것부터 우선순위를 매겨 돌려줄 수 있다.

## 4. 웹 서비스 컴포지션 검색 알고리즘

PSR 시스템은 웹 서비스의 WSDL 파일, 컴포지션 그래프와 가능한 모든 웹 서비스 컴포지션을 RDBMS의 테이블에 저장한다. 이 장에서는 웹 서비스를 위한 릴레이션 표현과 웹 서비스 컴포지션 검색을 위한 알고리즘을 설명한다.

### 4.1 웹 서비스와 온톨로지의 릴레이션 표현

이 절에서는 웹 서비스를 표현하는 테이블과 웹 서비스 컴포지션을 선 계산하기 위한 테이블의 스키마를 설명한다.

그림 5는 웹 서비스를 표현하는 테이블의 구조와 어떻게 웹 서비스가 저장되는 지를 보여준다. 각각의 웹 서비스들은 WS 테이블의 투플로 표현된다. 웹 서비스의 연산(operation), 입력(input)과 출력(output)을 WSDL 파일에서 추출하여 WS 테이블의 컬럼으로 저장한다. 웹 서비스를 위한 유일한 식별자도 또한 저장한다. 온톨로지 정보는 Ont 테이블에 저장한다. 온톨로지의 트리 구조는 parent 컬럼과 LV 컬럼으로 표현한다.

WS 테이블과 Ont 테이블을 조인하여 관련있는 온톨로지 정보만 가지고 있는 웹 서비스를 저장하는 OntWS 테이블을 얻을 수 있다. 이 때 조인 조건으로 WS의 Input과 Output 컬럼이 Ont 테이블의 Term 컬럼과 일치하는 지 여부를 이용한다. 다음과 같은 SQL을 사용하면 OntWS 테이블을 위한 데이타를 얻을 수 있다.

```
SELECT WS.ID AS ID, WS.Operation AS Operation,
       WS.Input AS Input, ont1.ClassID AS ClassID_I,
       ont1.Parent AS Parent_I, ont1.LV AS LV_I,
       WS.Output AS Output, ont2.ClassID AS ClassID_O,
       ont2.Parent AS Parent_O, ont2.LV AS LV_O
FROM WS, Ont AS ont1, Ont AS ont2
WHERE WS.Input = ont1.Term AND WS.Output = ont2.Term
ORDER BY ID ASC;
```

그림 6(a)는 Edge 테이블을 보여준다. Edge 테이블은 웹 서비스로부터 변환한 컴포지션 그래프를 저장한다. Edge 테이블로부터 그림 6(b)에 표현된 웹 서비스의 가중치를 가진 방향 그래프를 얻을 수 있고, 그 반대 동작도 가능하다. Edge 테이블의 각각의 투플은 정점(웹 서비스)들 사이의 가중치를 가진 방향 간선을 의미한다. ID_S와 ID_E 컬럼은 간선의 시작 정점과 끝 정점을 의미하고, Input 컬럼과 Output 컬럼은 간선의 입력과 출력을 의미한다. 컬럼 DM(Degree of Matching)은 3.3절에서 설명한 온톨로지에 기반한 매칭의 정확성을 표현한다. DM 컬럼의 값이 간선의 가중치가 되는데, 값 0은 온톨로지 상에서 'exact'를 뜻하고, 값 2는 'plug-in'을 의미한다.

**예제 2.** 그림 1에 있는 웹 서비스들과 그림 6(a)에 있는 Edge 테이블을 고려하자. Edge 테이블에 있는 첫 번째 투플은 WS1에서 WS2로 향하는 방향 간선이 존재한다는 것을 의미한다. 간선의 입력 값은 "City"이고 출력 값은 "Weather"이다. WS1의 출력 값은 "Zipcode"이고 WS2의 입력도 "Zipcode"이기 때문에 DM 컬럼은 "exact" 매칭을 의미하는 값 '0'을 가진다.

Path 테이블은 웹 서비스들 간의 모든 경로를 저장한다. 이것은 가능한 웹 서비스 컴포지션을 선 계산하고

**WS** table

| ID | Operation | Input | Output |
|---|---|---|---|
| 1 | CityToZipcode | City | Zipcode |
| 2 | WeatherInfo | ZipCode | Weather |
| 3 | GetTemperature | Postcode | Temperature |
| 4 | SportsByWeather | Weather | SportsOK |

**Ont** table

| ClassID | Parent | LV | Term |
|---|---|---|---|
| Ont1 | Thing | 1 | Zipcode |
| Ont1 | Thing | 1 | Postcode |
| Ont2 | Thing | 1 | Weather |
| Ont3 | Thing | 1 | SportsOK |
| Ont4 | Thing | 1 | City |
| Ont5 | Ont2 | 2 | Temperature |

**OntWS** table

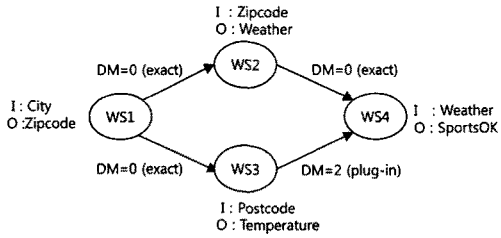
| ID | Operation | Input | CLASSID_I | Parent_I | LV_I | Output | CLASSID_O | Parent_O | LV_O |
|---|---|---|---|---|---|---|---|---|---|
| 1 | CityToZipcode | City | Ont4 | Thing | 1 | Zipcode | Ont1 | Thing | 1 |
| 2 | WeatherInfo | ZipCode | Ont1 | Thing | 1 | Weather | Ont2 | Thing | 1 |
| 3 | GetTemperature | Postcode | Ont1 | Thing | 1 | Temperature | Ont5 | Ont2 | 2 |
| 4 | SportsByWeather | Weather | Ont2 | Thing | 1 | SportsOK | Ont3 | Thing | 1 |

그림 5 웹 서비스와 온톨로지를 저장하는 테이블들

**Edge** table

| EdgeID | ID_S | ID_E | Input | Output | DM |
|---|---|---|---|---|---|
| E1 | 1 | 2 | City | Weather | 0 |
| E2 | 1 | 3 | City | Temperature | 0 |
| E3 | 2 | 4 | Zipcode | SportsOK | 0 |
| E4 | 3 | 4 | Postcode | SportsOK | 2 |

(a) Edge 테이블




(b) 컴포지션 그래프 표현

그림 6 Edge 테이블과 컴포지션 그래프

저장한다는 것을 의미한다. Path 테이블에 저장한 투플은 시작 웹 서비스와 종점 웹 서비스 사이의 이행 폐쇄(transitive closure)를 의미한다. 웹 서비스 컴포지션에 참여하는 모든 웹 서비스들을 반환하기 위해서, 경로의 중간 정점들을 알 필요가 있다. VisitedWS 테이블은 경로의 중간 정점(웹 서비스)들을 저장한다. 이 두 테이블은 Edge 테이블을 이용하여 얻을 수 있다. 자세한 알고리즘은 4.2장에서 설명한다.

**예제 3.** 그림 7(a) Path 테이블의 첫 번째 컬럼과 네 번째 컬럼을 고려하자. 경로 P1은 두 개의 웹 서비스 WS1과 WS2을 연결하는 간선 하나만으로 이루어져 있다. 경로 P1의 입력과 출력은 각각 "City"와 "Weather"이다. 경로 P4는 두 개의 간선으로 구성되어 있다. 시작 웹 서비스와 끝 웹 서비스는 WS1과 WS4이다. 경로 P4에 참여하는 웹 서비스들은 그림 7(b)의 VisitedWS 테이블에서 찾을 수 있다. 경로 P4는 3개의 웹 서비스 WS1, WS2와 WS4로 구성된다.

| PathID | WS_S | WS_E | Path_I | Path_O | DM | NumEdge |
|---|---|---|---|---|---|---|
| P1 | 1 | 2 | City | Weather | 0 | 1 |
| P2 | 1 | 3 | City | Temperature | 0 | 1 |
| P3 | 2 | 4 | Zipcode | SportsOK | 0 | 1 |
| P4 | 1 | 4 | City | SportsOK | 0 | 2 |
| P5 | 3 | 4 | Postcode | SportsOK | 0 | 1 |
| P6 | 1 | 4 | City | SportsOK | 2 | 2 |

(a) **Path** 테이블

| PathID | Edge_Start | Edge_End | Order |
|---|---|---|---|
| P4 | 1 | 2 | 1 |
| P4 | 2 | 4 | 2 |
| P6 | 1 | 3 | 1 |
| P6 | 3 | 4 | 2 |

(b) **VisitedWS** 테이블

그림 7 Path 테이블과 VisitedWS 테이블

### 4.2 웹 서비스의 컴포지션의 선 계산

이 절에서는 Edge 테이블에서 어떻게 웹 서비스 컴포지션을 선 계산하여 Path 테이블과 VisitedWS 테이블에 저장하는지 설명한다. 이를 위해서 NaïveEP-Join 알고리즘과 EP-Join 알고리즘을 제안한다.

4.2.1 NaiveEP-Join 알고리즘

NaïveEP-Join 알고리즘은 Edge 테이블을 반복적으로 사용하여 모든 웹 서비스 컴포지션을 계산하는 알고리즘이다. NaïveEP-Join 알고리즘은 메인 메모리 기반의 알고리즘[8]을 RDBMS 기반으로 적용한 것이다. NaïveEP-Join 알고리즘의 핵심은 i개의 간선을 가지는 경로를 저장하는 테이블 $P_i$를 $P_1$과 $P_{i-1}$ 테이블을 조인하여 반복적으로 생성하는 것이다. 알고리즘 1은 NaïveEP-Join 알고리즘을 보여준다.

| | |
|---|---|
| | **Input** : Edge – Edge 테이블 |
| | **Output** : Path – Path 테이블 |
| 1 | Relation $P_1$ ← Relation *Edge*; |
| 2 | **for** i ← 2 **to** *N-1* **do** |
| | // Pi는 I 개의 간선을 가진 경로를 저장 |
| 3 | $P_i \leftarrow P_i \bowtie P_{i-1}$ ; |
| | **endfor** |
| 4 | Relation *Path* ← $P_1 \cup P_2 \cup \ldots \cup P_{N-2} \cup P_{N-1}$ ; |

알고리즘 1 NaïveEP-Join

N개의 웹 서비스가 주어졌다면, 3 행에서 각각의 조인 연산의 시간 복잡도는 $O(N^2)$이다. 따라서 NaïveEP-Join을 사용하여 모든 경로를 계산할 때의 시간 복잡도는 $O(N^3)$이다.

4.2.1 EP-Join 알고리즘

웹 서비스 컴포지션을 선 계산하기 위한 EP-Join 알고리즘의 기본 개념은 다음과 같다. Edge 테이블에서 첫 번째 투플(간선)을 가져와서 그 간선을 포함하는 모든 경로를 생성하고, 생성된 경로를 Path 테이블에 저장한다. 경로가 2개 이상의 간선들로 구성된다면, 경로를 이루는 중간 정점(웹 서비스)들은 VisitedWS 테이블에 저장한다. Edge 테이블에서 모든 투플을 다 사용할 때까지 위 절차를 반복한다. EP-Join 알고리즘의 구체적인 동작은 알고리즘 2와 같다. EP-Join 알고리즘은

```
Input : (Edge, Path) – 테이블들

1  foreach e in the Edge table do
2      insert e into the Path table;
3      foreach p in the Path table do
4          if case 1 then
5              newPath ← connect e at the start of p;
6          else if case 2 then
7              newPath ← connect e at the end of p;
8          else
9              newPath ← connect e at the middle of p;
           endif
       endfch
10     insert the newPath into the Path Table;
11     if numEdge(newPath) ≥ 2 then
12         insert the intermediate vertices of newPath into the VisitedWS table;
       endif
   endfch
```

알고리즘 2 EP-Join

[12]의 알고리즘을 기반으로 구현되었다. 그러나 EP-Join 알고리즘은 웹 서비스 컴포지션에 대한 결과를 찾는 알고리즘이지만 [12]의 알고리즘은 그래프에서 모든 쌍 최단 경로(all-pairs shortest path)를 찾는 알고리즘이라는 차이가 있다.

그림 8의 세가지 경우들은 알고리즘 2를 수행하는 데 있어서 핵심적인 부분이다. 간선 e와 경로 p가 주어졌을 때, 새로운 경로를 생성하는 세가지 경우가 존재한다. 첫 번째는 경로 p의 마지막 정점이 간선 e의 시작 정점과 일치하는 경우로서, 경로 p의 끝에 간선 e를 연결하여 새로운 경로 newPath를 생성한다. 두 번째는 경로 p의 시작 정점이 간선 e의 끝 정점과 일치하는 경우로서, 간선 e를 경로 p의 시작 정점에 연결하여 새로운 경로 newPath를 생성한다. 세 번째는 간선 e의 시작 정점과 끝 정점이 경로 p의 중간 정점들과 일치 하는 경우로서, 경로 p의 중간 정점들에 간선 e를 연결하여 새로운 경로 newPath를 생성한다.

N개의 웹 서비스가 주어졌다면, EP-Join 알고리즘의 시간 복잡도는 $O(N^2)$이다.

**예제 4.** 그림 6에 있는 Edge 테이블의 투플들을 고려하고, Path 테이블은 비어있다고 가정하자. 첫 번째 투플은 (1, 2, "City", "Weather", 0)이고 두 번째 투플은 (1, 3, "City", "Temperature")이다. 하나의 간선을 가진 경로 P1과 P2를 Path 테이블에 삽입한다. P1과 P2는 그림 8의 경우에 해당하지 않으므로 새로 생성되는 경로는 없다.

세 번째 투플은 (2, 4, "Zipcode", "SportsOK", 0)이다. 하나의 간선을 가진 경로 P3를 Path 테이블에 삽입한다. 경로 P1과 간선 E3는 그림 8의 경우 3에 해당하므로 서로 연결을 하여 새로운 경로 P4를 생성한다. 경로 P4도 Path 테이블에 삽입하는데, P4는 두 개 이상의 간선을 가지고 있으므로 중간 정점(웹 서비스)에 대한 정보를 VisitedWS 테이블에 저장한다. 이러한 과정을 edge 테이블의 모든 정점을 다 사용할 때까지 반복한다.

### 4.3 웹 서비스의 컴포지션

EP-Join 알고리즘을 사용하여 웹 서비스 컴포지션을 선 계산하는 전체 절차는 알고리즘 3이 보여준다. 5행에 있는 EP-Join 알고리즘 대신하여 NaïveEP-Join 알고리즘을 사용할 수도 있다. 그림 9는 알고리즘 3에 대한 질의 계획을 표시하고 있다. 알고리즘 3은 SQL과 PL/SQL 문장으로 표현되었다.

### 4.4 웹 서비스 컴포지션 검색

웹 서비스 컴포지션을 선 계산한 후에, 사용자의 웹 서비스를 검색 질의에 해당하는 결과들을 테이블에서 검색하여 반환할 수 있다. 검색 절차는 두 단계로 구성

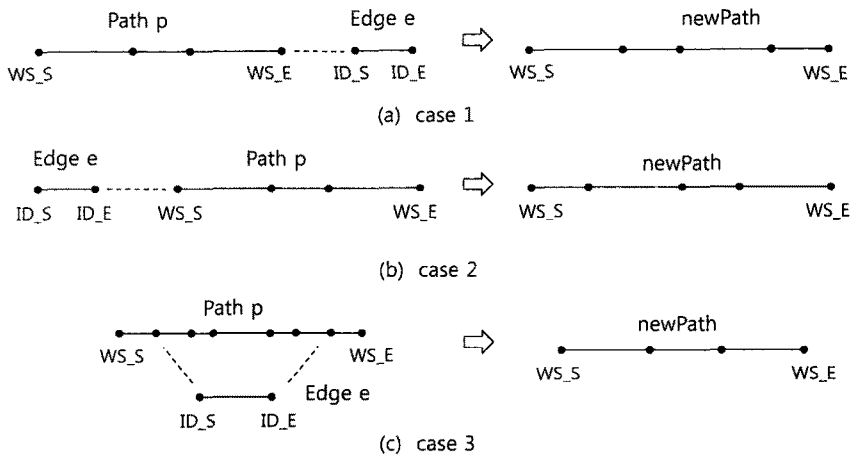



그림 8 EP-Join 알고리즘의 경우들

```
Input : { WS, Ont, Edge, Path, VisitedWS } - 테이블

// UDDI에서 웹 서비스 정보를 가져와서 WS 테이블에 저장
1  WS ← { ws | ws = (ID, Op, I, O), 1≤ k ≤ N} ;
// 온톨로지 정보를 Ont 테이블에 저장
2  Ont ← { ont = (ID, class, parent, distance, keyword), 1≤ k ≤ N} ;
// 온톨리지 정보와 웹 서비스 연결
3  OntWS ← { WS ⋈ Ont };
// 웹 서비스들 사이에서 존재하는 모든 가능한 경로를 Edge 테이블에 저장
4  Edge ← { OntWS ⋈ OntWS };
// 모든 경로를 발견하고 Path 테이블에 저장, 중간 노드들은 VisitedWS 테이
   블에 저장
5  Do EP-Join or NaïveEP-Join Algorithm;
```

알고리즘 3 웹 서비스 컴포지션의 선 계산

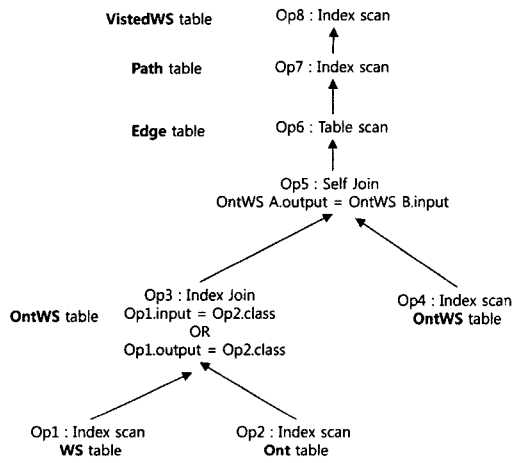



그림 9 웹 서비스 컴포지션 선 계산의 질의 계획

된다. 첫 번째 단계에서 Path 테이블에서 사용자 질의와 일치하는 경로들의 PathID를 찾는다. 두 번째 단계에서는 PathID를 이용하여 VisitedWS 테이블에서 경로를 구성하는 중간 정점(웹 서비스)들을 모두 알아낸다. 중간 점점들을 시퀀스로 표현하면 사용자 질의에 대한 웹 서비스 컴포지션 결과가 된다. 알고리즘 4는 웹 서비스 컴포지션 검색 과정을 설명한다.

```
Input : Q - 사용자 질의
Output : pWS - 웹 서비스들의 시퀀스

1  PathID ← Index scan on Path table using the query Q;
2  pWS ← Index Scan on VisitedWS table using PathID;
3  return pWS;
```

알고리즘 4 웹서비스 컴포지션 검색

**예제 5.** 사용자가 입력이 "City"이고 출력이 "SportsOK"인 웹 서비스를 찾기를 원한다고 가정하자. Path 테이블에 있는 경로 P4와 P5가 사용자 질의를 만족한다. P4와 P5는 두 개 이상의 간선을 가지고 있으므로, VisitedWS 테이블을 검색하여 각 경로의 중간 경로들을 얻어야 한다. 웹 서비스 컴포지션 검색의 최종 결과는 (WS1, WS2, WS4)와 (WS1, WS3, WS4)이다.

## 5. 실 험

이 장에서는 먼저 이 논문에서 제안하는 RDBMS 기반의 웹 서비스 컴포지션 알고리즘과 메인 메모리 기반의 알고리즘[8]의 실행 시간을 비교한다. 또한 RDBMS 기반의 알고리즘은 NaïveEP-Join과 EP-Join의 효용성을 비교한다.

### 5.1 실험 환경

이 논문에서는 메인 메모리 알고리즘과 PSR 시스템을 비교하였다. 실험은 1GB의 메모리를 가지고 있고 Windows XP Professional이 설치되어 있는 Intel Pentium IV 3.0 GHz 컴퓨터에서 수행하였다. PSR 시스템의 웹 서비스 컴포지션 검색은 Oracle 10g Standard Edition의 SQL과 PL/SQL을 이용하여 구현하였다. 웹 서비스 컴포지션을 위한 메인 메모리 알고리즘 Visual C++ 컴파일러를 사용하여 C 언어로 구현하였다.

메인 메모리 기반 알고리즘과 PSR 시스템은 그 실행 기반이 상이하여 성능 비교를 위한 환경은 메모리 크기나 디스크 스페이스의 크기를 동일하게 하는 것은 의미가 없다. 따라서 이 논문에서는 동일한 데이타셋과 동일한 수의 웹 서비스를 갖고 실험을 하였다.

실험을 위하여 간단한 구조를 가진 최대 100,000개의 웹 서비스로 구성된 Synthetic 데이타셋을 사용하였다. 표 1은 웹 서비스를 생성하기 위하여 사용한 파라미터에 대한 설명과 그 값들을 보여준다. 웹 서비스 컴포지션에 참여하는 간선들은 파라미터 pe의 값에 의하여 결정되며, 실험에서는 총 웹 서비스의 20% 정도가 연결되었다고 가정하였다. 입력이나 출력으로 사용할 키워드의 개수의 범위는 웹 서비스 개수의 5배로 설정하였고, 실제 사용한 개수는 pe. 값에 영향을 받는다. 그리고 웹 서비스들에게 각각 하나의 입출력 값을 할당하였다.

웹 서비스 컴포지션 검색을 위하여 질의를 랜덤하게 생성하여 수행하였으며, 실험에 사용된 질의의 최대 개수는 1,000개이다.

온톨로지 정보를 이용한 실험에는 다음과 같은 방법으로 생성한 온톨로지 트리를 이용하였다. 키워드의 총 수와 온톨로지 개념의 총 수는 웹 서비스의 총 수의 10

표 1 실험 데이타를 위한 파라미터들

| 파라미터 | 설명 | 값 |
|---|---|---|
| $N_w$ | 웹 서비스의 개수 | 1,000에서 100,000 |
| $N_q$ | 질의의 개수 | 1에서 1,000 |
| $p_e$ | 연결된 간선의 확률 | 0.2 |

배와 5배로 설정하였다. 온톨로지 트리의 최대 깊이는 $logN_w$로 설정하였다. 예를 들어 10,000개의 웹 서비스를 이용한다며, 키워드의 총 수는 100,000개이고 온톨로지 개념의 총 수는 50,000개이며 온톨로지 트리의 높이는 4이다. 웹 서비스의 입력과 출력을 온톨로지의 트리의 노드에 랜덤하게 할당하였다.

### 5.2 성능 평가

이 절에서는 PSR 시스템과 메인 메모리 알고리즘[8]의 성능을 질의 수행시간에 대하여 실행 시간으로 분석하였다.

5.2.1 질의 수행 시간 비교

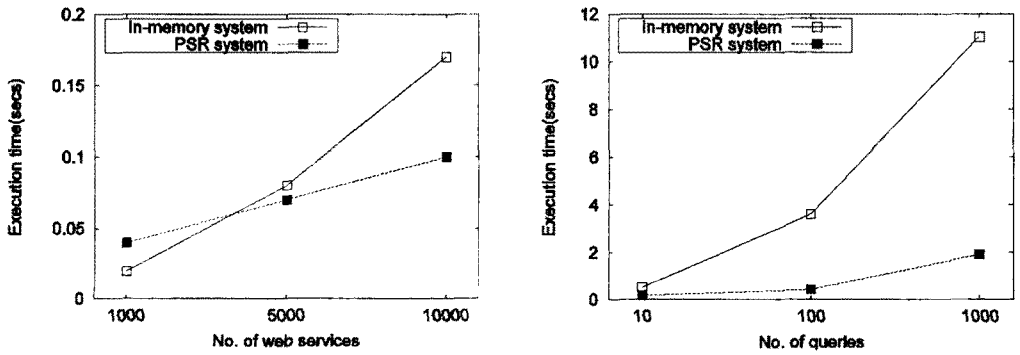
이 절에서는 RDBMS 기반의 EP-Join 알고리즘과 메인 메모리 알고리즘의 성능을 질의 수행시간에 대하여 실행 시간으로 분석하였다. 이 실험에서는 exact 매칭만을 고려하였다. 그림 10(a)는 웹 서비스의 개수를 1,000개에서 10,000개로 변화 시킬 때 질의 수행 시간을 보여준다. 이 때 질의의 개수는 1개로 고정하였다. 웹 서비스의 개수가 적을 때는 메인 메모리 기반의 알고리즘이 더 빨리 수행된다. 웹 서비스의 개수가 증가함에 따라 두 알고리즘의 수행 시간도 증가함을 알 수 있다. 그러나 RDBMS 기반의 EP-Join 알고리즘의 수행 시간은 메인 메모리 기반의 알고리즘 보다 덜 증가함을 알 수 있다. 또한 웹 서비스의 개수가 많을 때는 EP-Join 알고리즘이 메인 메모리 기반의 알고리즘 보다 더 빨리 수행됨을 알 수 있다. 메모리 기반 알고리즘은 질의가 들어올 때 마다 웹 서비스 컴포지션 그래프를 온라인으로 생성하고 그 그래프를 탐색하여 컴포지션의 결과로 반환한다. 그런데 인접 행렬을 이용한 메인 메모리 알고리즘이 $O(N^3)$의 시간 복잡도 보이기 때문에, 웹 서비스의 수가 증가할 때 마다 컴포지션을 위한 그래프 생성과 탐색에 시간이 많이 걸린다. 이에 비하여, EP-Join 알고리즘에서는 메인 메모리 기반의 시스템과 달리 컴포지션을 사전에 계산해 둠으로써 실제 사용자의 검색이 웹 서비스 개수의 증가에 크게 영향을 받지 않는다.

그림 10(b)는 웹 서비스의 개수를 10,000개로 고정하고, 질의의 수를 10개에서 1,000개로 변화시켰을 때의 질의 수행시간을 보여준다. EP-Join 알고리즘이 메인 메모리 기반의 알고리즘 보다 질의를 빨리 처리함을 알 수 있다. 질의의 수가 증가함에 따라 EP-Join 알고리즘과 메인 메모리 기반 알고리즘의 수행 시간의 차이가 더 벌어지는 것을 확인할 수 있다. 이는 그림 10(a)에서 보이는 실행시간의 차가 사용자 질의 개수만큼 그 차가 누적되어 증가한 결과이다. 이는 PSR 시스템이 메인 메모리 기반의 시스템보다 더 좋은 확장성을 가지고 있다는 것을 보여준다.

입력과 출력이 각기 다른 1,000개의 질의에 대한 웹 서비스 컴포지션 검색을 하려고 한다면, 메인 메모리 기반의 알고리즘은 매번 웹 서비스 컴포지션을 위한 그래프를 생성하고 탐색하여 그 결과를 사용자에게 돌려주어야 하지만, PSR 기법은 모든 가능한 웹 서비스 컴포지션을 미리 계산하여 테이블에 저장해 두고 있기 때문에 저장한 테이블에 대한 검색만을 수행하면 된다. 이런 이유 때문에 그림 10(b)와 같은 결과가 나타난다.

5.2.2 웹 서비스 컴포지션의 선 계산 시간 비교

이 절에서는 NaïveEP-Join과 EP-Join 알고리즘의 웹 서비스 컴포지션의 선 계산 시간을 비교하였다. 웹 서비스의 개수를 1,000개에서 10,000개로 변화시키면서 선 계산이 완료 될 때까지의 실행 시간을 측정하였다. 이 실험에서도 정확 매칭만을 고려하였다. 그림 11은 그 결과를 보여준다. 예상했던 대로 EP-Join의 실행 시간이 NaïveEP-Join의 실행 시간보다 훨씬 더 좋으며, 두 알고리즘의 시간 차이는 웹 서비스의 개수가 많아짐에 따라 더 증가함을 볼 수 있다.




(a) 웹 서비스 개수의 변화에 따른 실행 시간 (b) 질의 개수에 대한 변화 따른 실행 시간

그림 10 웹 서비스 컴포지션의 선 계산 시간

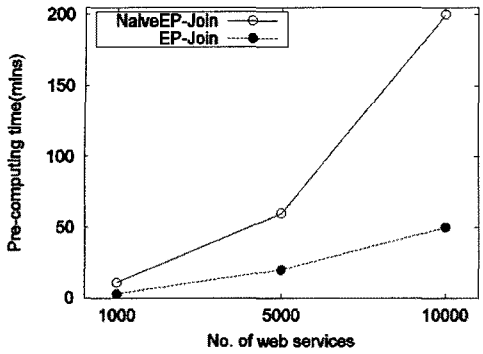



그림 11 선 계산 시간 비교

5.2.3 점증적인 삽입(incremental insertion)에 따른 선 계산 시간

처음에는 모든 웹 서비스를 한번에 대량 삽입(bulk insertion)하여 시스템을 생성하겠지만, 일반적으로는 시간에 지남에 따라 웹 서비스를 점증적으로 삽입(incremental insertion)하여 시스템을 유지한다. 이 절에서는 EP-Join 알고리즘의 웹 서비스 점증적인 입력에 따른 선 계산 시간을 정확 매칭만을 고려하여 측정하였다. 각각의 실험 동안에 1,000개의 웹 서비스를 삽입하여 점증적인 선 계산 시간을 측정하였고 이를 10번 반복하여 평균 값을 구하였다. 그림 12는 점증적인 삽입에 따른 실행 시간을 보여주는데, 뒤로 갈수록 삽입 시간이 증가하는 경향을 보인다. 이는 웹 서비스의 개수가 많아짐에 따라 연결된 웹 서비스의 수도 약간씩 증가하기 때문이다. 1,000개가 들어올 때 마다 평균 5분의 시간이 걸림을 알 수 있다.

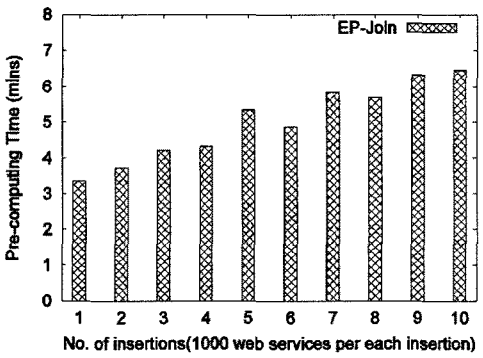



그림 12 웹 서비스의 점진적인 삽입

5.2.4 온톨로지를 포함한 웹 서비스 컴포지션 검색

이번 절에서는, 온톨로지를 고려한 경우와 고려하지 않은 경우에 PSR 시스템의 성능이 어떻게 영향을 받는지를 분석한다.

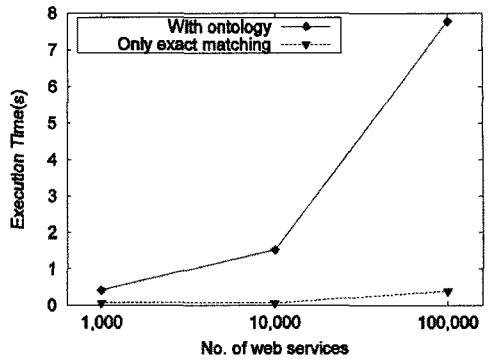



그림 13 온톨로지를 포함한 경우의 웹 서비스 컴포지션 선 계산 시간 비교

표 2 가능한 웹 서비스의 컴포지션의 수

| $N_w$ | exact 매칭만 고려한경우 | | 온톨로지를 포함한 경우 | |
|---|---|---|---|---|
| | 수 | 비율 | 수 | 비율 |
| 1,000 | 123 | 0.123 | 1,413 | 1.413 |
| 10,000 | 1,095 | 0.1095 | 21,599 | 2.160 |
| 100,000 | 11,333 | 0.113 | 143,847 | 14.385 |

먼저, exact 매칭만 고려한 경우와 온톨로지를 포함한 경우의 선 계산 시간의 차이를 측정하였다. 웹 서비스의 개수를 1,000개, 10,000개와 100,000개로 나누어서 그 수행 시간을 측정하였다. 그림 13은 그 결과를 보여준다. 예상했던 대로, 온톨로지 개념도 고려하여 웹 서비스 컴포지션을 선 계산하는 경우의 실행 시간이 exact 매칭만 고려하는 경우보다 훨씬 느린 결과를 보였다. exact 매칭만 고려한 경우 선 계산 시간은 약간의 변화만 있는 반면에, 온톨로지를 고려한 경우에는 선 계산 시간의 상당한 변화를 볼 수 있다. 이런 차이는 표 2에서 나타난 가능한 웹 서비스 컴포지션의 비교에서 유추해 볼 수 있다. exact 매칭만 고려한 경우 가능한 웹 서비스 컴포지션의 수를 총 웹서비스의 수로 나눈 비율의 값이 약간만 변화하지만, 온톨로지를 고려한 경우 온톨로지를 포함한 경우 그 비율의 값이 대략 10배 정도 변화함을 알 수 있다.

그림 14는 온톨로지를 포함한 경우와 exact 매칭만 고려한 경우의 질의 수행 시간의 차이를 보여준다. 웹 서비스의 개수를 1,000에서 100,000개로 변화하였으며, 시간 측정에 10개의 질의를 이용하였다. 예상했던 바대로, 온톨로지에 있는 클래스도 고려하여 검색을 하는 경우 수행 시간이 더 오래 걸림을 알 수 있으며, 온톨로지를 포함한 경우에 수행 시간의 변화도 훨씬 큼을 알 수 있다. exact 매칭만 고려한 경우는 질의 시간의 변화가 거의 없었으나 온톨로지를 고려한 경우에는 상당히 많이 변화함을 알 수 있다. 표 3은 결과로서 나오는 웹 서

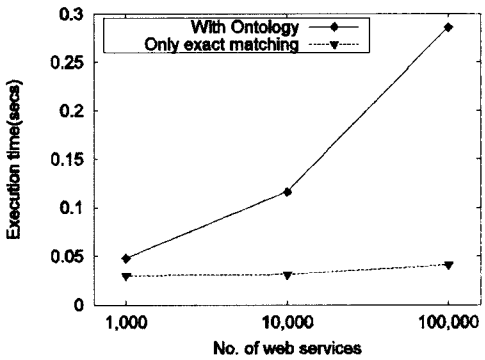



그림 14 온톨로지를 포함한 경우의 질의 검색 시간 비교

표 3 웹 서비스의 컴포지션의 최대 길이

| $N_w$ | exact 매칭만 고려한 경우 | 온톨로지를 포함한 경우 |
|---|---|---|
| 1,000 | 3 | 14 |
| 10,000 | 3 | 19 |
| 100,000 | 4 | 20 |

비스 컴포지션의 최대 길이를 보여주는데, exact 매칭만 고려한 경우에는 거의 변화가 없다. 온톨로지를 고려한 경우에는 14에서 20으로 결과가 변화함을 알 수 있다. 10,000개와 100,000개의 경우 최대 길이의 차이는 거의 존재하지 않으나 표 2의 비율의 7배 정도 차이가 나는 것을 알 수 있다. 이 비율의 차이로 인해 질의 실행 시간의 차이가 발생한다.

## 6. 결 론

이 논문에서는 웹 서비스 컴포지션 검색 시스템 PSR을 제시하였다. PSR은 메인 메모리 기반의 알고리즘을 사용하지 않고 RDBMS를 사용하여 웹 서비스 컴포지션을 계산하는 최초의 연구이다. 선 계산하는 알고리즘은 웹 서비스를 변환한 방향을 가진 가중치 그래프 위에서 수행된다. 웹 서비스 컴포지션은 그래프에서 웹 서비스들 사이의 경로를 구성하는 중간 정점들로 생각할 수 있다. 이 변환된 그래프들은 테이블 형태로 저장이 되고, 테이블 사이에서 조인 연산을 수행하여 웹 서비스 컴포지션을 구할 수 있다. 또한 웹 서비스에서 추출한 온톨로지 정보도 테이블로 저장하고, PSR 시스템이 이를 사용하여 온톨로지 매칭 정도에 따라 사용자 질의와의 유사도를 통한 결과를 반환하도록 하였다. 실험을 통하여 PSR 시스템이 많은 수의 웹 서비스와 질의들을 다룰 때 메인 메모리 기반의 시스템보다 더 좋은 확장성을 가지고 있음을 보였다.

## 참 고 문 헌

[1] Luc Clement, Andrew Hately,Claus von Riegen, and Tony Rogers. UDDI Version 3.0.2, Oct. 2004. http://uddi.org/pubs/uddi v3.htm.

[2] Web Services architecture overview. http://www.ibm.com/developerworks/library/w-ovr/

[3] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1, Mar. 2001. http://www.w3.org/TR/wsdl.

[4] Xin Dong and Alon Y. Halevy and Jayant Madhavan and Ema Nemes and Jun Zhang. Similarity search for web services. In *VLDB04*, pages 372-383, Toronto, Canada, Aug 2004.

[5] K. Sivashanmugam, K. Verma, A. P. Sheth, and J. A. Miller. Adding semantics to web services standards. In *ICWS03*, pages 395-401, Las Vegas, Nevada, USA, June 2003.

[6] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Semantic matching of web services capabilities. In *First International Semantic Web Conference*, pages 333-347, Sardinia, Italy, 2002.

[7] D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In *VLDB05*, pages 613-624, Trondheim, Norway, Aug. 2005.

[8] J. Gekas and M. Fasli. Automatic web service composition based on graph network analysis metrics. In *International Conference on Ontologies, Databases and Applications of SEmantics*, pages 1571-1587, Agia Napa, Cyprus, 2005.

[9] J. Gekas and M. Fasli. Automatic web service composition using web connectivity analysis techniques. In *W3C Workshop on Frameworks for Semantics in Web Services*, Innsbruck, Austria, 2005.

[10] E. Sirin, J. A. Hendler, and B. Parsia. Semi-automatic composition of web services using semantic descriptions. In *Proceedings of the 1st Workshop on Web Services: Modeling, Architecture and Infrastructure*, pages 17-24, Angers, France, 2003.

[11] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven web services composition. In *WWW03*, pages 411-421, Budapest, Hungary, 2003.

[12] C. Pang, G. Dong, and K. Ramamohanarao. Incremental maintenance of shortest distance and transitive closure in first-order logic and SQL. *ACM Trans. Database Syst.*, 30(3):698-721, 2005.

권 준 호

1999년 서울대학교 컴퓨터공학과 졸업 2001년 서울대학교 전기컴퓨터공학부 석사학위 취득. 2001년~현재 서울대학교 전기컴퓨터공학부 박사과정. 관심분야는 XML 인덱싱, XML 문서 필터링, 웹 서비스등

박 규 호

2005년 서울대학교 컴퓨터공학부 졸업 2007년 서울대학교 전기컴퓨터공학부 석사학위 취득. 2007년~현재 해군 군 복무중. 관심분야는 XML 인덱싱, 웹 서비스 등

이 대 욱

1995년 경북대학교 컴퓨터공학과(학사) 1997년 경북대학교 대학원 컴퓨터공학과(석사). 1997년~2003년 (주)퓨쳐인포넷 부설 연구소 연구원. 2003년~현재 서울대학교 전기컴퓨터공학부 박사과정. 관심분야는 XML 인덱싱, XML 문서 필터링, 웹 서비스 등

이 석 호

1964년 연세대학교 정치외교학과 졸업 1975년, 1979년 미국 텍사스대학교 전산학 석사와 박사학위 취득. 1979년~1982년 한국과학원 전산학과 조교수. 1982년~1986년 한국정보과학회 논문 편집위원장 1986년~1989년 미국 IBM T.J. Watson 연구소 객원교수. 1988년~1990년 데이타베이스연구회 운영위원장. 1989~1991년 서울대학교 중앙교육연구전산원 원장. 1994년 한국정보과학회 회장. 1997년~1999년 한국학술진흥재단부설 첨단학술정보센터 소장. 1982년~현재 서울대학교 컴퓨터공학부 교수