내포 결과를 이용한 복합 웹 서비스 실행의 비용 기반 최적화

(Cost-based Optimization of Composite Web Service Executions Using Intensional Results)

박 창 섭 †

(Chang-Sup Park)

요 약 웹 서비스는 인터넷 상에 분산되어 있는 이질적인 응용들 사이의 연동 및 통합을 위한 표준화된 수단을 제공한다. 본 논문에서는 계층적인 연동 관계가 존재하는 복합 웹 서비스들에 대해 서비스 결과로 전달되는 내포 데이타를 활용하여 웹 서비스들의 호출 및 복귀 작업을 서버 및 통신 비용에 따라 효과적으로 분산 수행함으로써 웹 서비스 시스템의 전체적인 성능을 향상시킬 수 있는 방안을 제시한다. 본 논문에서는 내포 결과를 이용한 적법한 웹 서비스 호출 실행 계획 및 이에 대한 비용 기반 최적화 문제를 정의하고, 최적 호출 실행 계획을 찾기 위한 휴리스틱 탐색 방법과 효율적으로 수행될 수 있는 그리디 알고리즘을 제안한다. 실험 결과, 제안한 그리디 알고리즘은 빠른 시간 내에 최적 해에 가까운 효율적인 호출 실행 계획을 생성하며, 복잡한 웹 서비스 연동 관계에 대해서 우수한 확장성을 보였다.

키워드 : XML, 웹 서비스, 내포 데이타, 최적화

Abstract Web service technologies provide a standard means for interoperation and integration of heterogeneous applications distributed over the Internet. For efficient execution of hierarchically interacting composite web services, this paper proposes an approach to distribute web service invocations over peer systems effectively, exploiting intensional XML data embedding external service calls as a result of web services. A cost-based optimization problem on the execution of web services using intensional results was formalized, and a heuristic search method to find an optimal solution and a greedy algorithm to generate an efficient invocation plan quickly were suggested in this paper. Experimental evaluation shows that the proposed greedy algorithm provides near-optimal solutions in an acceptable time even for a large number of Web services.

Key words : XML, Web Services, Intensional Data, Optimization

1. 서 론

웹 서비스(web services)는 XML 및 웹 프로토콜을 기반으로 인터넷 상에 분산된 이질적인 응용(application)들을 연동 및 통합하기 위한 기술로서, 최근 학계 및 산업계에서 연구와 적용이 활발히 이루어지고 있다. 서로 다른 언어 및 플랫폼을 사용하는 응용들을 웹 서비스화 함으로써 쉽게 연동시킬 수 있으며, 기존 웹 서비스들을 조합(composition)하여 새로운 응용을 쉽게 개발할 수 있다. 이러한 웹 서비스 기술들은 소프트웨어 개발의 효율성과 신뢰성을 높이기 위한 서비스 지향 구조(Service-Oriented Architecture) 패러다임의 기반이 된다[1].

최근 기존 XML 데이타 내에 외부 웹 서비스들에 대한 호출을 포함하는 내포 XML 데이타(intensional XML data)의 개념과 이를 활용한 응용들이 제안되었다[2-4]. 내포 XML 데이타를 이용함으로써 웹 서비스 호출 실행의 주체 및 호출 시점을 실행 시간에 동적으로 결정할 수 있다. 예를 들어, 특정 서버에 저장된 내포 XML 데이타에 대한 검색 요청 시, 서버가 그 안에 포함된 외부 웹 서비스들에 대한 호출을 실행하고 그 결과들을 통합하여 최종 결과를 생성 및 반환하거나, 또는 내포 XML 데이타를 그대로 반환하여 클라이언트로 하여금 웹 서비스 호출들을 실행하도록 할 수 있다. 또 서버와 클라이언트가 웹 서비스들의 호출 실행을 적절히

† 정 회 원 : 수원대학교 인터넷정보공학과 교수
park@suwon.ac.kr
논문접수 : 2006년 3월 15일
심사완료 : 2006년 9월 21일

분담하는 것도 가능하다. 뿐만 아니라, 호출된 웹 서비스는 그것의 실행 결과로 다른 웹 서비스들에 대한 호출을 포함하는 내포 데이타를 반환할 수도 있으므로, 일반적으로 하나의 내포 문서를 완전히 실체화(materialization)하기 위한 다양한 호출 실행 방법이 존재한다[5].

기존의 복합 웹 서비스(composite web service)들을 이용하여 새로운 복합 웹 서비스를 구성하는 서비스 조합이 반복적으로 이루어 지면 그림 1과 같이 다수의 웹 서비스들 사이에 계층적인 연동 관계, 즉, 호출 및 복귀 관계가 형성된다. 이러한 환경에서 웹 서비스들 사이에 내포 데이타를 교환하여 다른 웹 서비스들에 대한 호출 실행을 위임함으로써 다양한 호출 실행 방법이 가능하다. 즉, 웹 서비스가 다른 웹 서비스를 호출할 때 입력 매개변수(parameter)로서 내포 데이타를 전달할 수 있으며, 호출된 웹 서비스는, 그것이 복합 웹 서비스인 경우, 요소 웹 서비스 호출들의 일부를 포함한 내포 데이타를 서비스 결과(result)로서 반환할 수 있다. 피호출 웹 서비스가 다른 웹 서비스에 대한 호출을 포함하는 내포 결과를 호출자 웹 서비스에 반환한 경우, 그 호출자 웹 서비스는 내포 결과에 포함된 웹 서비스 호출을 실행하거나 또는 그것을 포함하는 결과를 자신의 호출자에게 반환할 수 있다. 예를 들어, 그림 1의 실행 사례에서는 WS 2가 WS 4와 WS 6을 직접 호출하지 않고 그 호출문들을 포함한 내포 결과를 자신의 호출자, 즉, WS 1에게 반환한다. WS 1은 WS 6에 대한 호출을 직접 실행한 후 WS 4에 대한 호출을 포함하는 내포 결과를 생성하여 클라이언트에게 전송하고, 클라이언트는 WS 4에 대한 호출을 직접 실행한다. 클라이언트가 컴퓨팅 성능, 보안성 등의 제약으로 인해 내포 데이타의 처리가 어려운 경우에는 에이전트 시스템[6] 등을 이용하여 간접적으로 내포 결과의 처리 작업을 수행하는 방안을 고려할 수 있다.

웹 서비스에 대한 호출 및 복귀는 XML로 정의되는 SOAP 메시지 포맷과 SOAP 프로토콜을 이용하여 수행된다. 이러한 웹 서비스 연동 방식은 XML 데이타의 인코딩, 디코딩, 파싱, 통합 등의 부가 작업을 필요로 하며 이러한 작업들은 웹 서비스를 제공하는 서버와 그것을 호출하는 클라이언트의 성능에 상당한 영향을 미친다[7-9]. 웹 서비스를 제공하는 피어(peer) 시스템들은 일반적으로 서로 다른 성능과 부하량(workload)을 가지며 두 피어 시스템 사이의 통신 비용도 모두 다르다. 따라서 웹 서비스를 호출하는 클라이언트 측면에서 볼 때, 웹 서비스 호출 및 복귀 작업을 효율적으로 수행할 수 있는 피어 시스템을 클라이언트로 선택함으로써 웹 서비스 호출 실행 비용을 줄일 수 있다.

본 논문에서는 내포 데이타를 활용하여 웹 서비스들에 대한 호출을 효율적으로 실행하기 위한 방안을 제시한다. 본 논문에서 제안하는 방법은 각 피어 시스템에서의 웹 서비스 호출 처리 비용을 고려하고 A* 탐색 및 그리디 방식에 기반하여 주어진 웹 서비스들에 대한 최적 호출 실행 계획 및 효율적인 호출 실행 계획을 생성한다. 다양한 웹 서비스 연동 관계들에 대한 모의 실험 결과, 제안한 방법에 의해 생성된 호출 실행 계획의 비용은 평균적으로 원래 정의된 웹 서비스 연동 관계에 따른 호출 실행 비용의 약 62% 수준에 해당하는 것으로 나타나 최적화에 의해 실행 비용이 크게 절감될 수 있음을 보였다. 그리디 알고리즘은 실험 데이타의 71%에 대해 최적 해와 동일한 결과를 도출하였으며, 최적해를 생성하지 못한 경우에도 평균적인 비용이 최적 해의 약 97% 수준으로 최적 해에 매우 근접한 결과를 생성하였다. 또한 그리디 알고리즘은 웹 서비스의 개수의 증가에 대해 최적화 수행 시간이 크게 늘어나지 않고 비교적 빠른 시간 내에 수행을 완료함으로써 좋은 성능 및 확장성을 가짐을 보였다. 본 논문에서 제안한 방법을 통해 웹 서비스들 간의 계층적인 연동 환경에서 각 웹 서비스의 호출 처리 비용을 고려하여 웹 서비스 호출들을 효과적으로 분산 수행함으로써 웹 서비스 시스템의 전체적인 수행 성능을 향상시킬 수 있는 효과를 얻을 수 있다.

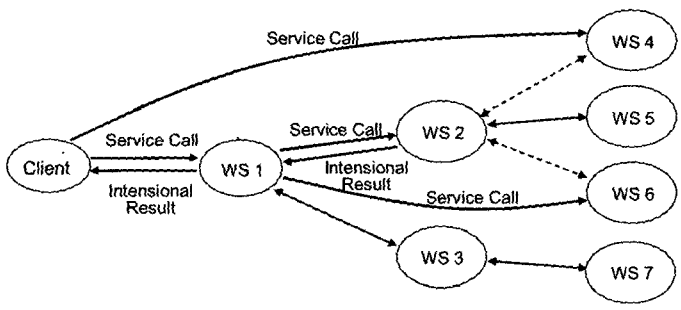


그림 1 내포 결과를 이용한 웹 서비스 연동 예

본 논문의 구성은 다음과 같다. 2장에서는 웹 서비스 및 내포 XML 문서의 활용에 대한 관련 연구들을 소개한다. 3장에서는 내포 결과 전달 방식을 사용하는 웹 서비스들에 대한 효율적인 호출 실행을 위한 비용 기반 최적화 문제를 정의하고, 4장에서 A* 탐색 방법 및 그리디 방식에 기반한 최적화 알고리즘을 제안한다. 5장에서는 실험을 통해 제안한 알고리즘에 대한 성능을 평가하고, 6장에서 결론을 맺고 향후 연구 방향을 제시한다.

2. 관련 연구

웹 서비스의 활용은 그것의 기능 및 인터페이스를 응용 프로그램 수준에서 인식 가능하도록 정의하는 웹 서비스 기술 언어(Web Service Description Language: WSDL), 그것을 외부에 공개하고 검색하기 위한 저장 시스템(Universal Description, Discovery, and Integration: UDDI), 그리고 다른 웹 서비스와의 통신을 위한 객체 접근 프로토콜(Simple Object Access Protocol: SOAP) 등을 통해 이루어 진다[10]. 최근에는 웹 서비스들의 조합, 서비스 품질 향상, 의미론적 서비스 기술 및 탐색 등에 대한 연구들이 활발히 이루어 지고 있다[11,12].

최근 학계 및 산업계에서는 내포 XML 문서 및 이를 활용한 응용 시스템들이 제안되었다[2-4]. 특히 Active XML(AXML)[13]은 피어-투-피어(peer-to-peer: P2P) 시스템 환경에서 XML 및 웹 서비스 기술을 이용하여 분산 데이타 관리, 데이타 교환, 검색 및 통합을 위한 프레임워크(framework)를 제공한다. 각 피어 시스템은 AXML 문서들에 대한 데이타베이스를 관리하고 AXML 웹 서비스를 통해 이에 대한 접근을 허용한다. 즉, AXML 웹 서비스는 AXML 문서에 대한 매개 변수화된 XML 질의 프로시저로 정의된다. AXML 문서는 다른 피어 시스템에서 제공하는 웹 서비스에 대한 호출을 포함(embedding)할 수 있으며, 이러한 서비스 호출을 통해 원격 시스템에 존재하는 데이타를 검색 및 추출하고 지역적인 정적 XML 데이타와 통합할 수 있다.

[5]에서는 웹 서비스들 사이에 매개변수 및 결과로 내포 XML 문서를 교환하기 위해 기정의된 내포 XML 스키마에 부합하는 내포 데이타를 생성하는 데이타 변환 기법을 제안하였다. [14]에서는 내포 XML 문서들에 대한 질의 처리 시 관련있는 웹 서비스들 만을 선택적으로 호출하여 효율적으로 처리하기 위한 방법을 제안하였다.

[9]에서는 AXML 플랫폼 상에서 내포 XML 문서를 효율적으로 실체화 하기 위한 휴리스틱 기법을 제안하였다. 이 논문에서는 피어 시스템들이 다른 피어 시스템으로부터 임의의 서비스 호출을 전달받아 실행할 수 있는 범용 질의 서비스(generic query service)를 제공한다고 가정하고 있다. 이 논문은 이 범용 질의 서비스에 내포 데이타를 입력 매개변수로 사용하여 전달할 때 생성 가능한 다양한 실체화 방법들을 고려하여 이들에 대한 최적화 방안을 제안하였다. 그러나 본 논문에서 제안한 바와 같은 내포 결과의 전달을 통한 웹 서비스 호출의 위임 방법에 대해서는 고려하지 않았으며, 구체적인 최적화 알고리즘이나 그것의 효과에 대한 실험 결과도 제시되지 않았다.

3. 문제 정의

본 장에서는 내포 결과를 반환 및 처리 가능한 복합 웹 서비스들에 대한 최소 비용 호출 실행 계획에 대해 정형적으로 기술한다. 본 논문에서는 웹 서비스들이 연산의 결과 값으로 내포 XML 데이타를 반환할 수 있고 웹 서비스 호출자는 이러한 내포 결과를 처리할 수 있다고 가정한다. 또 호출 실행 계획의 전역적인 최적화를 위해 복합 웹 서비스들의 접근 정보 및 요소 웹 서비스들에 관한 정보가 주어진다고 가정한다.[1)]

본 논문에서는 웹 서비스 호출 실행 과정에 대한 다음과 같은 비용 모델을 이용한다. 일반적으로 특정 클라이언트가 특정 웹 서비스를 호출할 때, 다음과 같은 작업들이 순서대로 수행된다.

(1) 웹 서비스 호출 초기화: 클라이언트에서 웹 서비스 호출 실행을 위해 필요한 초기화 작업을 수행한다.
(2) 호출 메시지 생성: 웹 서비스 호출을 위해 필요한 서비스 주소, 연산 이름, 매개변수 목록 등을 XML 문서 형태로 인코딩 및 패킹(packing)한 SOAP 메시지를 생성한다.
(3) 호출 메시지 전송: 네트워크를 통해 웹 서비스 호출 메시지를 웹 서비스 실행 서버로 전송한다.
(4) 호출 메시지 해석(parsing): 웹 서비스 실행 서버에서 호출 SOAP 메시지를 분해(unpacking)하고 해석한다.
(5) 웹 서비스 실행: 웹 서비스 실행 서버에서 호출된 웹 서비스를 실행한다.
(6) 웹 서비스 결과 메시지 생성: 웹 서비스 실행 서버에서 웹 서비스 실행 결과를 포함한 SOAP 메시지를 생성한다.
(7) 결과 메시지 전송: 네트워크를 통해 웹 서비스 결과 메시지를 클라이언트로 전송한다.
(8) 결과 메시지 해석: 클라이언트에서 결과 SOAP 메시지를 분해 및 해석한다.

1) 예를 들면, 이러한 정보는 웹 서비스 제공자에 의해 웹 서비스 기술 정보에 포함되어 UDDI 저장소 등을 통해 공개될 수 있다.

(9) 웹 서비스 결과 결합(merging): 클라이언트에서 타 웹 서비스로부터의 내포 결과에 포함된 웹 서비스를 실행한 경우, 이 실행 결과를 타 웹 서비스의 실행 결과와 결합한다.

따라서 특정 웹 서비스 호출에 따른 실행 비용은 위에서 기술된 각 단계별 비용 요소들의 합으로 구성된다. 이들 중 (1), (2), (8), (9) 단계는 클라이언트 측 비용, (4)~(6) 단계는 서버 측 비용, 그리고 (3), (7) 단계는 클라이언트와 서버 사이의 통신 비용에 해당한다. 이러한 비용 요소는 처리에 소요되는 응답 시간(response time)으로 표현될 수 있다.

웹 서비스의 중복(replication) 실행을 고려하지 않으면, 특정 웹 서비스가 실행되는 위치, 즉, 웹 서비스 제공자는 그 웹 서비스를 호출하는 클라이언트에 독립적이며, 따라서 내포 결과 전달을 통해 여러 피어 시스템에서 웹 서비스를 호출 가능한 경우에도 서버 측 처리 비용은 동일하다. 그러나 클라이언트 측 처리 비용과 클라이언트와 서버 사이의 통신 비용은 웹 서비스를 호출하는 클라이언트 시스템에 따라 달라진다. 따라서 본 논문에서는 서버 측 실행 비용인 (4)~(6) 단계의 비용을 제외한 나머지 단계들의 비용의 합을 특정 클라이언트로부터의 특정 웹 서비스에 대한 호출 실행 비용으로 정의한다.

웹 서비스 호출 실행의 비용 기반 최적화를 위해서는 상기 비용 요소들에 대한 예측이 필요하다. 클라이언트 측 비용은 실시간 모니터링 등을 통해 각 시스템의 성능 및 부하량을 지속적으로 측정함으로써 예측될 수 있다[9,15]. 두 피어 시스템 사이의 통신 비용도 네트워크 상의 트래픽과 통신 지연 시간 측정을 통해 예측된다고 가정한다.

본 논문에서는 기술의 단순함을 위해 요청 및 응답(request and response) 방식의 웹 서비스 연동만을 고려한다. 또 임의의 두 웹 서비스 사이에 하나의 호출 경로만이 존재하고 일련의 웹 서비스들 사이에 순환(cycle)적인 호출 관계가 존재하지 않는다고 가정한다.[2] 이와 같은 가정 하에, 웹 서비스들 사이에 정의된 계층적인 호출 관계는 하나의 클라이언트를 루트 노드(root node)로 갖는 유향 트리(tree)로 표현될 수 있다.

정의 1. (웹 서비스 호출 정의 트리) $DT = (V_d, A_d, W_d)$ 는 각 웹 서비스 정의에 따른 웹 서비스들 사이의 호출 관계를 나타낸 가중 유향 트리(weighted directed tree)로서, V_d는 각 웹 서비스를 나타내는 유한개의 노드(vertice)들의 집합, $A_d \subseteq V_d \times V_d$는 두 웹 서비스 사이의 호출 관계를 나타내는 아크(arc) 들의 집합, $W_d: A_d \rightarrow Z^+$는 각 아크에 대한 웹 서비스 호출 실행 비용을 정의하는 가중 함수이다. □

정의 2. (호출 정의 경로) 웹 서비스 호출 정의 트리 DT 내의 임의의 서로 다른 두 노드 v, w에 대해, v에서 w까지의 연결 경로에 포함된 노드들을 순서대로 나열한 리스트(v, s_1, s_2, …, s_n, w)를 v에서 w까지의 호출 정의 경로라 정의하고 $P_d(v, w)$로 나타낸다. □

웹 서비스 호출 정의 트리의 높이는 웹 서비스들 사이의 연속적인 호출 관계의 최대 길이를 의미하고, 각 노드의 외향 차수는 그 노드에 대응되는 복합 웹 서비스를 구성하는 요소 웹 서비스들의 개수를 나타낸다.

웹 서비스들 사이의 내포 결과 전달을 고려할 때, 임의의 서로 다른 두 웹 서비스 v, w에 대해 호출 정의 경로 $P_d(v, w) = (v, s_1, s_2, \cdots, s_n, w)$ ($n \geq 1$)가 존재하면 v가 w를 호출하는 웹 서비스 실행 사례가 존재할 수 있다. 왜냐 하면, w에 대한 호출을 포함한 내포 결과가 웹 서비스 s_n으로부터 v에게 직간접적으로 전달되면 v가 w를 호출할 수 있기 때문이다. 따라서 웹 서비스 호출 정의 트리 $DT=(V_d, A_d, W_d)$가 주어졌을 때, A_d에 대해 이행적 폐쇄(transitive closure)를 수행한 결과인 A_d^*를 아크 집합으로 갖고 W_d를 포함하는, A_d^*에 대한 가중 함수 $W_d^*: A_d^* \rightarrow Z^+$를 갖는 그래프 $DT^*=(V_d, A_d^*, W_d^*)$는 웹 서비스들 사이에 내포 결과를 전달함으로써 가능한 모든 웹 서비스 호출 관계를 나타낸다(그림 2(a) 및 2(b) 참조).

정의 3. (웹 서비스 호출 실행 트리) $ET=(V_e, A_e, W_e: A_e \rightarrow Z^+)$는 주어진 웹 서비스 호출 정의 트리 $DT=(V_d, A_d, W_d)$에 대해, 내포 결과 전달을 이용한 적법한 호출 실행 사례(instance)를 나타내는 가중 유향 트리로서, $V_e = V_d$, $A_e \subseteq A_d^*$, $W_e \subseteq W_d^*$를 만족한다. 이 때, A_e는 두 웹 서비스 사이의 호출 실행 관계를 나타내는 아크들의 집합이다. □

정의 4. (호출 실행 경로) 웹 서비스 호출 실행 트리 ET 내의 임의의 서로 다른 두 노드 v, w에 대해, v에서 w까지의 연결 경로에 포함된 노드들을 순서대로 나열한 리스트(v, s_1, s_2, …, s_n, w)를 v에서 w까지의 호출 실행 경로라 정의하고 $P_e(v, w)$로 나타낸다. □

주어진 DT에 대한 적법한 실행 사례에서는 각 웹 서비스가 하나의 호출자에 의해 한 번 호출되고 그 실행 결과는 그 호출자로 반환되어야 한다. 즉, ET에서 임의의 노드 s_k에 대해 $(s_i, s_k) \in A_e$와 $(s_j, s_k) \in A_e$를 동시에 만족시키는 서로 다른 두 노드 s_i, s_j가 존재할 수 없다. 따라서 주어진 웹 서비스들에 대한 하나의 실행 사례를 나타내는 호출 실행 트리 ET는 DT^*에 대한 유향 신장

2) 그러나 본 논문에서 제안하는 방법은 보다 일반적인 웹 서비스 환경을 위해 확장될 수 있다. 가령, 두 웹 서비스 사이에 다수의 호출 경로들이 존재할 경우 웹 서비스 호출 정의 트리와 호출 실행 트리의 아크 및 가중 함수는 각각 멀티셋(multi-set)으로 정의된다.

트리(spanning tree)를 구성한다(그림 2(c) 참조). 그러나 모든 유향 신장 트리가 DT에 대한 적법한 실행 사례를 의미하는 것은 아니다.

일반적으로 웹 서비스 호출 실행 트리는 다음과 같은 특성을 갖는다.

보조정리 1. 웹 서비스 호출 정의 트리 DT 내의 두 웹 서비스 v, w에 대해 호출 정의 경로 $P_d(v, w)$가 존재하면, DT에 대한 모든 호출 실행 트리 ET에 호출 실행 경로 $P_e(w, v)$가 존재할 수 없다.

증명: $P_e(w, v)=(s_1, s_2, \cdots, s_n)$가 ET 내에 존재한다고 가정하자. 호출 실행 트리의 정의에 의해 $A_e \subseteq A_d^*$이므로 $P_e(w, v)$에 속한 모든 아크 (s_i, s_{i+1})에 대해 호출 정의 경로 $P_d(s_i, s_{i+1})$이 존재한다 ($1 \le I \le n-1$). 따라서, $P_d(w, v)$이 존재해야 하나, 이는 $P_d(v, w)$가 DT에 존재한다는 가정에 위배된다. □

상기 보조정리 1은 일련의 웹 서비스들이 반드시 DT 내의 호출 정의 경로에 나타난 순서에 따라 호출되어야 함을 의미한다. 그러나 $P_d(v, w)$가 존재할 때 내포 결과 전달을 통해 v가 w보다 먼저 복귀하는 것이 가능하다.

보조정리 2. 웹 서비스 호출 정의 트리 DT 내의 두 웹 서비스 v, w에 대해 호출 정의 경로 $P_d(v, w) = (v, s_1, s_2, \cdots, s_n, w)$ ($n \ge 1$)가 존재한다고 할 때, DT에 대한 호출 실행 트리 ET에 아크 (v, w)가 존재한다(즉, 웹 서비스 v가 웹 서비스 w를 호출한다)고 가정하면, 모든 웹 서비스 s_i ($1 \le i \le n$)에 대한 호출 실행 경로 $P_e(v, s_i)$가 ET 내에 존재한다.

증명: 호출 정의 경로 $P_d(v, w)$의 길이에 대한 수학적 귀납법으로 증명할 수 있다.

(i) $n = 1$, 즉, $|P_d(v, w)|=3$일 때, v가 w를 호출하기 위해서는 v가 s_1을 호출하고 w에 대한 호출 정보를 전달받아야 하므로 성립한다.

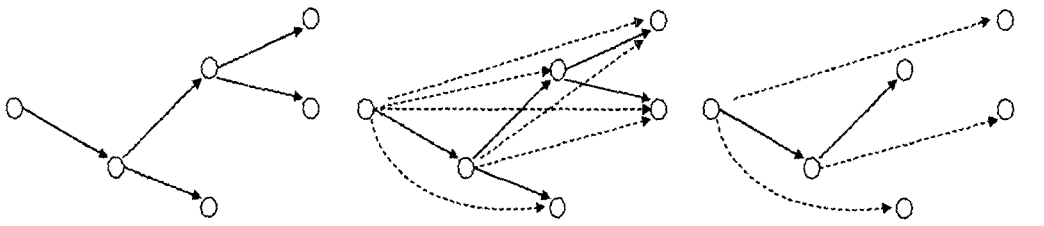
(ii) $n \le k$ ($k \ge 1$), 즉, $|P_d(v, w)| \le k + 2$일 때 성립 가정한다.

(iii) $n = k + 1$, 즉, $|P_d(v, w)| = k + 3$일 때, $P_d(v, w) = (v, s_1, s_2, \cdots, s_n, w)$에서 v가 w를 호출하기 위해서는 s_n으로부터 w 호출 정보가 v에게 직간접적으로 전달되어야 한다. 즉, 호출 실행 경로 $P_e(v, s_n) = (r_1, r_2, \cdots, r_m)$ (단, $r_1 = v$, $r_m = s_n$, $m \ge 3$, $r_i \in P_d(v, w)$ for all $1 \le i \le m$ 을 만족)이 존재해야 한다. 따라서 v를 제외한, $P_e(v, s_n)$에 포함된 모든 노드 r_i ($1 < i \le m$)에 대해 $P_e(v, r_i)$가 존재한다. 한편, $P_e(v, s_n)$에 포함된 임의의 두 이웃 노드 r_i, r_{i+1} ($1 \le i < m$)에 대해 $|P_d(r_i, r_{i+1})| \le k+2$이므로 상기 가정에 의해 $P_d(r_i, r_{i+1})$에 포함된 모든 노드 s (단, $s \ne r_j$ 이고 $s \ne r_{j+1}$)에 대해 호출 실행 경로 $P_e(r_i, s)$가 존재한다. 따라서 v를 제외한, $P_d(v, s_n)$에 포함된 모든 노드 s_i에 대해 $P_e(v, s_i)$가 존재한다. □

상기 특성들로부터 다음과 같은 정리가 성립한다(그림 3 참조).

정리 1. 웹 서비스 호출 정의 트리 DT 내의 두 웹 서비스 v, w에 대해 호출 정의 경로 $P_d(v, w) = (s_1, s_2, \cdots, s_n)$ ($s_1=v$, $s_n=w$, $n \ge 4$) 가 존재할 때, DT에 대한 모든 호출 실행 트리 $ET = (V_e, A_e, W_e)$ 에서 두 아크 (s_i, s_k)와 (s_j, s_m)이 동시에 존재할 수 없다. (단, i, j, k, m은 $1 \le i < j < k < m \le n$를 만족하는 자연수)

증명: DT에 대한 임의의 호출 실행 트리 $ET = (V_e, A_e, W_e)$에서 $(s_j, s_m) \in A_e$이고 또한 $(s_i, s_k) \in A_e$이라 가정하자. $j < k < m$ 이므로 보조정리 2에 의해 호출 실행 경로 $P_e(s_j, s_k)$가 존재한다. 그러나 보조정리 1에 의해 $P_e(s_j, s_i) \notin ET$ 이므로 s_i는 $P_e(s_j, s_k)$에 속할 수 없



(a) 웹 서비스 호출 정의 트리 DT (b) DT^* (c) 내포 결과 전달에 의한 실행 트리 ET

그림 2 웹 서비스 호출 정의 트리 및 호출 실행 트리

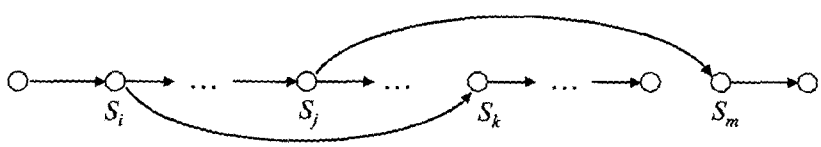


그림 3 내포 결과를 전달하는 웹 서비스들에 대해 적법하지 않은 실행 사례

다. 따라서 s_k는 s_i 외의 다른 웹 서비스에 의해서도 호출되어야 하나, 이러한 중복 호출 관계는 적법한 웹 서비스 실행 사례가 될 수 없다. 따라서 $(s_j, s_m) \in A_e$일 경우 아크 (s_i, s_k)가 A_e에 존재할 수 없다. 또한 대우 관계에 의해 $(s_i, s_k) \in A_e$이면 $(s_j, s_m) \notin A_e$가 성립한다. 따라서, 두 아크는 A_e 내에 동시에 존재할 수 없다. □

상기 정리 1에 의해, 내포 결과를 전달하는 웹 서비스들의 효율적인 실행을 위한 최적화 문제를 다음과 같이 정의할 수 있다.

정의 5. (최적 웹 서비스 호출 실행 계획 생성 문제) 웹 서비스들의 집합에 대한 웹 서비스 호출 정의 트리 $DT = (V_d, A_d, W_d)$가 주어졌을 때, 다음과 같은 집합 Π를 고려한다.

$\Pi = \{ T \mid T$는 DT^*에 대한 유향 신장 트리로서, DT 내의 임의의 호출 정의 경로 $P_d(s_i, s_m) = (s_i, s_{i+1}, \cdots, s_m)$ 에 대해, $i < j < k < m$을 만족하는 두 아크 (s_i, s_k)와 (s_j, s_m)를 동시에 포함하지 않음$\}$

이 때, Π에 속한 각 트리를 DT에 대한 호출 실행 계획(invocation plan)이라 한다. 또 임의의 가중 유향 트리 $T=(V, A, W)$에 대해 $Weight(T) = \sum_{a \in A} W(a)$ 라 할 때, $Weight(T_0) = \underset{T \in \Pi}{Min}\{Weight(T)\}$를 만족하는 호출 실행 계획 $T_0 \in \Pi$를 구하는 문제를 최적 웹 서비스 호출 실행 계획 생성 문제라 정의한다. □

4. 최적화 방안

본 장에서는 3장에서 기술한, 내포 결과 전달을 고려한 웹 서비스 호출 실행 계획에 대한 최적화 방안에 대해 기술한다. 먼저 가능한 모든 호출 실행 계획들을 조사하는 전역 탐색 방법의 문제점을 살펴보고, A* 탐색 방법에 기반하여 최적 해를 찾는 방법과 빠른 시간 내에 최적 해에 가까운 효율적인 해를 구할 수 있는 그리디 방법을 제안한다.

4.1 전역 탐색

상기 문제에 대한 가장 직접적인 해결 방안은 가능한 모든 호출 실행 계획들을 포함한 해 공간(solution space)을 전역적으로 탐색(exhaustive search)하는 것이다. 즉, 정의 5에 정의된 집합 Π에 속한 모든 호출 실행 트리들을 나열(enumeration)하고 이들 중 최소 실행 비용을 가진 것을 선택한다. 각 호출 실행 트리에서, 루트 노드 r에서부터 임의의 노드까지의 호출 실행 경로는 유일해야 한다. 즉, 서로 다른 두 노드 v, w에 대해, $P_d(r, v) = (s_1, s_2, \cdots, s_l, t_1, t_2, \cdots, t_m)$, $P_d(r, w) = (s_1, s_2, \cdots, s_l, u_1, u_2, \cdots, u_n)$이라 할 때, 중복되는 경로 $(s_1, s_2, \cdots, s_l)$에 속한 각 노드에 대한 호출 계획이 유일해야 하므로 두 경로 $P_d(r, v)$와 $P_d(r, w)$에 대해 개별적으로 최적 호출 실행 경로를 구하는 것은 전체 호출 정의 트리에 대한 올바른 실행 계획을 도출할 수 없다. 따라서 적법한 모든 호출 실행 계획들을 체계적으로 생성하기 위한 방법이 필요하다.

웹 서비스 호출 정의 트리 DT에 속한 노드들 중 깊이가 k 이하인 노드들 만을 포함하는 호출 실행 트리들의 집합을 $E(k)$라 하면, $E(k+1)$은 $E(k)$로부터 다음과 같은 방법으로 생성될 수 있다. $E(k)$에 속한 각 호출 실행 트리 $ET_{k\ i}$에 대해, 깊이가 k인 노드(즉, 웹 서비스)들에 대한 DT 내의 자식 노드(즉, 요소 웹 서비스)들의 집합 N을 구한다. 그리고 내포 결과 전달 방식을 고려하여, N에 속한 각 노드에 대해 그것을 호출할 수 있는 웹 서비스 노드를 $ET_{k\ i}$ 내에서 선택하고 각 노드 쌍에 대한 아크를 추가한다. 이러한 방법으로 $E(k)$에 속한 호출 실행 트리들로부터 $E(k+1)$에 속하는 모든 호출 실행 트리들을 생성할 수 있다.

이와 같은 방법을 통해 주어진 웹 서비스 호출 정의 트리 DT에 대해 가능한 모든 호출 실행 트리를 생성할 수 있다. 복잡도 분석을 위해, DT가 모든 내부 노드들의 외향 차수(out-degree)가 f로 동일하고 모든 단말 노드들의 깊이가 h로 동일한 완전 트리(perfect tree)라고 가정하면, 가능한 호출 실행 트리의 총 개수는 다음과 같다.

$$|E(h)| = \sum_{k_1,k_2,\ldots,k_{h-1}} \binom{f^{h-1}}{k_1, k_2, \ldots, k_{h-1}} \left(2^f\right)^{k_1} \cdot \left(3^f\right)^{k_2} \cdots \left(h^f\right)^{k_{h-1}}$$

$$= \left(2^f + 3^f + \cdots + h^f\right)^{f^{h-1}} = \left(\sum_{i=2}^{h} i^f\right)^{f^{h-1}}$$

where $k_1 + k_2 + \ldots + k_{h-1} = f^{h-1}$ and $h \geq 2$

위의 결과에 의하면, 내부 노드의 외향 차수나 트리의 깊이가 증가함에 따라 고려해야 할 호출 실행 트리의 수가 급격히 증가하므로 실제적으로 전역 탐색을 수행하는 것이 불가능해 진다. 예를 들어, f = 3, h = 4인 호출 정의 트리의 경우, 가능한 호출 실행 트리의 개수는 99^{27} = 7.6e+53개나 된다. 한편, DT에 속한 노드, 즉, 웹 서비스들의 개수를 n이라 하면 전역 탐색의 시간 복잡도는 $O(h^{2f^h}) = O(\log^{2n} n)$ 이 된다.

4.2 A* 탐색 알고리즘

휴리스틱(heuristic) 탐색 방법의 일종인 A* 알고리즘은 상태 공간의 일부를 탐색에서 제외시킴으로써 모든 상태들을 조사하지 않고 효율적으로 최적 해(optimal solution)를 찾을 수 있다[16]. 본 장에서는 주어진 웹 서비스 호출 정의 트리 DT에 대해 최소의 비용을 갖는 호출 실행 트리를 찾을 수 있는 효율적인 A* 알고리즘

을 제시한다.

본 알고리즘에서 상태 공간(state space)을 이루는 상태 s는 DT에 속한 노드들 중 깊이가 d_s보다 작거나 같은 노드들에 대한 호출 실행 트리 ET_s를 나타낸다. 그림 4에 기술된 바와 같이, 본 알고리즘은 DT의 루트 노드만으로 이루어 진 실행 트리에 대한 시작 상태로부터 시작하여 새로운 상태들을 생성해 나간다. $OPEN$은 이미 생성된 상태들 중 확장(expansion)을 위한 후보 상태들의 집합이다. A* 알고리즘은 매 단계마다 $OPEN$에 속한 각 상태 s에 대해 그것으로부터 도달 가능한 목적 상태(goal state)의 탐색 비용 $f^*(s)$을 휴리스틱을 통해 추정한 값 $f(s)$를 계산하고 최소 값을 가진 것을 선택하여 그것의 후속 상태들을 생성한다. 이 때, 상태 s로부터 도달 가능한 목적 상태란 DT에 대한 완전한 호출 실행 트리이면서 ET_s를 부-그래프로 포함하는 것들 중 최소 비용을 가진 트리와 연관된 상태를 말한다. 확장 후보로 선택된 상태 s의 후속 상태(successor)를 생성하는 방법은ET_s를 부-그래프로 포함하고 DT에서 깊이가 $d_s + 1$인 노드들과 그것들에 대한 호출 아크를 추가로 포함하는 모든 가능한 호출 실행 트리 ET_n에 대해 새로운 상태 s_{succ}를 하나씩 생성한다(그림 4의 16~20행 및 그림 5(a) 참조).

주어진 문제의 초기 상태로부터 상태 s까지의 탐색 비용을 $g^*(s)$, 상태 s로부터 어떤 목적 상태까지의 탐색 비용을 $h^*(s)$, 그리고 $h^*(s)$에 대한 추정 값을 $h(s)$라 할 때, $f^*(s)$와 $f(s)$는 다음과 같이 표현될 수 있다.

$$f^*(s) = g^*(s) + h^*(s),\ f(s) = g^*(s) + h(s) \quad (1)$$

일반적으로 A* 알고리즘이 항상 최적 해를 찾기 위해서는 모든 상태 s에 대해 추정 값 $h(s)$가 실제 탐색 비용 $h^*(s)$ 보다 항상 작거나 같음을 보장해야 한다 [16]. 이러한 탐색 비용에 대한 낙관적인 추정을 위해 본 알고리즘에서는 각 상태에 대해 다음과 같은 보조 트리를 이용한다.

정의 6. (탐색 비용 추정을 위한 보조 트리) 주어진 $DT = (V_d, A_d, W_d)$의 루트 노드를 r이라 하고 상태 s의 호출 실행 트리를 $ET_s = (V_s, A_s, W_s)$라 할 때, $HT_s = (V, A, W)$, 단, $V = V_d$, $A = A_s \cup \{ (u, w) \mid w \in V_d - V_s$, ET_s의 단말 노드이면서 $P_d(r, w)$에 속한

```
1  A* Search Algorithm
2  Input: web services call definition tree DT = (Vd, Ad, Wd)
3  Output: the optimal invocation tree for DT
4  Begin
5     Let the start state s0 = (({r}, ∅, ∅), 0).
6     Let OPEN be a priority queue storing states in non-decreasing order of f() value
      defined by the equation (2).
7     OPEN := { s0 };
8     Loop
9        Select from OPEN a state s having the smallest value of f(s).
10       OPEN := OPEN – { s };
11       Let the selected state s = (ETs, ds) where ETs=(Vs, As, Ws).
12       if Vs = Vd then
13          return ETs;
14       end if;
15       N := { w | w ∈ Vd –Vs and its depth in DT is ds + 1 };
16       for each possible invocation tree ETn = (Vn, An, Wn), where
            Vn = Vs ∪ N, An = As ∪ { (u, w) | for each w∈N, u is a node in Pe(r, v) in ETs
            where v is a node s.t. (v, w)∈Ad }, and Wn:An→Z+ satisfying Ws ⊆ Wn ⊆ Wd*
17       do
18          Generate a successor state of s, ssucc=(ETn, ds+1).
19          OPEN := OPEN ∪ { ssucc };
20       end for;
21    end loop;
22 end
```

그림 4 A* 탐색 알고리즘

노드를 v라 할 때, $P_e(r, v) \cdot P_d(v, w)$ (단, v는 한 번만 포함하고, w는 제외시킴) 에 속한 모든 노드 u에 대해 $W_d^*((u, w))$의 값이 최소인 u }, 그리고 $W:A \rightarrow Z^+$는 $W_s \subseteq W \subseteq W_d^*$를 만족한다. □

위의 정의에 의하면 HT_s는 DT에 속한 모든 노드를 포함하는 신장 트리로서, ET_s를 부-그래프로 포함하고, DT의 노드들 중 ET_s에 속하지 않은 각 노드에 대해 개별적으로 최소 호출 비용을 갖는 호출자 노드를 ET_s에서 선택하여 아크를 추가함으로써 생성된다(그림 5(b) 참조). 따라서 정리 1을 만족하는, 즉, DT에 대한 적법한 웹 서비스 호출 실행 트리가 아닐 수 있다. 그러나 s로부터 생성 가능한, DT에 대한 모든 호출 실행 트리 ET에 대해 $Weight(HT_s) \le Weight(ET) = f^*(s)$이 성립한다. 따라서,

$$f(s) = Weight(HT_s) = Weight(ET_s) + \sum_{w \in V_d - V_s} \underset{u \in P_e(r,v) \cdot P_d(v,w)}{Min} W_d^*((u,w)),$$

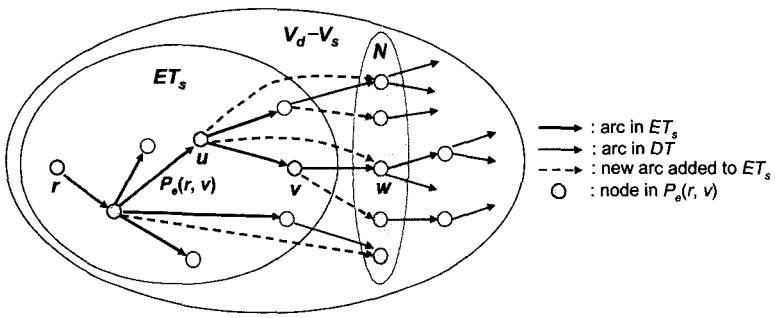
(단, v는 ET_s의 단말 노드이면서 $P_d(r,w)$에 속한 노드임) (2)

라 정의하면 모든 상태 s에 대해 $f(s) \le f^*(s)$이고, 수식 (1)로부터 $h(s) = f(s) - g^*(s)$, $h^*(s) = f^*(s) - g^*(s)$이므로 $h(s) \le h^*(s)$가 성립한다. 따라서 본 알고리즘은 수행 종료 시 항상 최적 해, 즉, 최소 호출 실행 비용을 갖는 호출 실행 계획을 생성함을 보장한다.

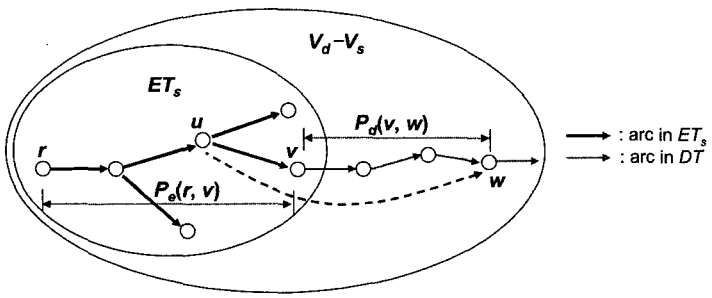
4.3 그리디 알고리즘

4.2절에서 기술된 A* 탐색 알고리즘은 최적 해를 찾을 수는 있으나 웹 서비스들의 수가 증가할수록 알고리즘의 성능이 크게 저하되는 문제점이 있다. 본 절에서는 보다 빠른 시간 내에 효율적인 웹 서비스 호출 실행 계획을 생성할 수 있는 그리디(greedy) 방식의 알고리즘을 제안한다.

그림 6의 의사코드에 기술된 바와 같이 본 알고리즘은 주어진 웹 서비스 호출 정의 트리 $DT = (V_d, A_d, W_d)$에 대해 넓이-우선 탐색(breadth-first traversal)을 수행하면서 하나의 완전한 호출 실행 트리를 생성한다. 즉, DT 내의 각 노드 w에 대해 그것을 호출할 호출자 노드를 선택하여 현재 생성 중인 실행 트리에 새로운 노드 및 아크를 추가한다. 이 때, w에 대한 호출자로 선택 가능한 노드는 루트 노드 r에서 w의 부모 노드 v까지의 호출 실행 경로에 포함된 조상 노드들이다. 본 알고리즘에서는 이들 중 w에 대한 호출 실행 비용뿐만 아니라 w의 모든 자손 노드들에 대한 호출 비용도 고려하여 호출자를 선택한다. 즉, DT 내에서 노드 w의 자손 노드들의 집합을 $Desc(w)$라 할 때, w를 호출 가능한 모든 웹 서비스 u에 대해 $W_d^*((u,w)) + \sum_{x \in Desc(w)} \underset{t \in P_e(r,u)}{Min} W_d^*((t,x))$



(a) 상태 s의 후속 상태 s_{succ} 생성



(b) 상태 s에서의 탐색 비용 추정을 위한 보조 트리의 생성

그림 5 A* 알고리즘 실행 방식

```
1  Greedy Algorithm
2  Input: web services call definition tree DT = (V_d, A_d, W_d)
3  Output: an invocation tree for DT
4  begin
5     Let Q be a queue to store nodes in DT. Q := ∅;
6     Let ET = (V_e, A_e, W_e) be the result invocation tree. ET := ({r}, ∅, ∅);
7     Let H be a hash table to store minimum weight values for descendent nodes.
8     while Q ≠ ∅ do
9        v := Dequeue(Q);
10       Let (s_1, s_2, ... , s_n) be the sequence of nodes in P_e(r, v) in ET.
11       for each child node w of v in DT do
            /* find u such that u = Min_{u'∈P_e(r,v)} ( W_d^*((u', w)) + Σ_{x∈Desc(w)} Min_{t∈P_e(r,u')} W_d^*((t, x)) )  */
12          Initialize H by H(x) := MAX_VAL for all x in Desc(w);
13          S_2 := MAX_VAL;
14          for each s_i from s_1 to s_n do
15             S_1 := 0;
16             for all x in Desc(w) do
17                H(x) := min{ H(x), W_d^*((s_i, x))};
18                S_1 := S_1 + H(x);
19             end for;
20             if W_d^*((s_i, w)) + S_1 < S_2 then
21                S_2 := W_d^*((s_i, w)) + S_1; u := s_i;
22             end if;
23          end for;
24          V_e := V_e ∪ {w}; A_e:= A_e ∪ {(u, w)}; W_e:= W_e ∪ {W_d^*((u, w))};
25          Enqueue(Q, w).
26       end for;
27    end while;
28    return ET;
29 end
```

그림 6 그리디 알고리즘

의 값이 최소인 것을 w의 호출자로 선택한다(그림 7 참조). 주목할 점은, 어떤 노드 u가 w를 호출할 경우 w의 자손 노드를 호출할 수 있는 노드는 $P_e(r, u)$에 속하는 노드들로 제한된다는 점이다. 따라서 호출 실행 경로 $P_e(r, v)$ 내의 노드 u의 위치에 따라 서로 다른 조상 노드 집합에 대해 각 자손 노드 x에 대한 최소 비용 호출자 t_0 및 그 호출 비용 $W_e^*((t_0, x))$을 찾아야 한다. $P_e(r, v)$의 길이를 l이라 할 때, 이러한 호출자 선택 과정의 시간 복잡도는 $O(|Desc(w)| \cdot l^2)$ 이다. 이 때, $O(|Desc(w)|)$의 부가적인 메모리 공간을 사용하여 시간 복잡도를 $O(|Desc(w)| \cdot l)$로 향상시킬 수 있다. 즉, 그림 6의 12~23행에 나타난 바와 같이, w의 호출자로서 $P_e(r, v)$에 포함된 각 노드 u를 루트 노드에서부터 깊이가 증가하는 순서로 고려하고, 각 단계에서 w의 각 자손 노드 x에 대해 $P_e(r, u)$에 포함된 조상 노드들로부터

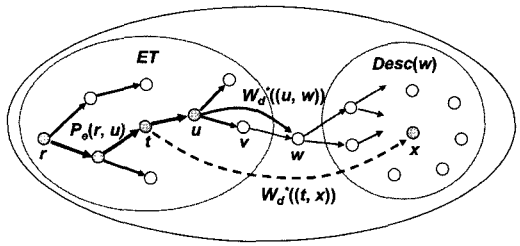


그림 7 그리디 알고리즘에서 w의 호출자 선택 방법

의 최소 호출 비용 $\underset{t \in P_e(r,u)}{Min} W_d^*((t, x))$ 을 참조 테이블에 저장하고 이를 그 다음 단계에서 이용하면 많은 비교 연산들을 피할 수 있다. DT를 모든 내부 노드들의 외향 차수가 f로 동일하고 모든 단말 노드들의 깊이가 h로 동일한 완전 트리라고 가정하고 DT에 포함된 웹 서비

스 노드의 개수를 n이라 하면, 모든 노드들에 대해 이와 같은 방식으로 호출자를 선택하는 본 그리디 알고리즘의 전체적인 시간 복잡도는 다음과 같다.

$$O(\sum_{i=1}^{h} |Desc(w)| \cdot i \cdot f^i) = O(\sum_{i=1}^{h} \frac{f^{h-i+1}-1}{f-1} \cdot i \cdot f^i)$$

$$= O(\frac{f}{f-1} n \log_f^2 n - 2n \log_f n)$$

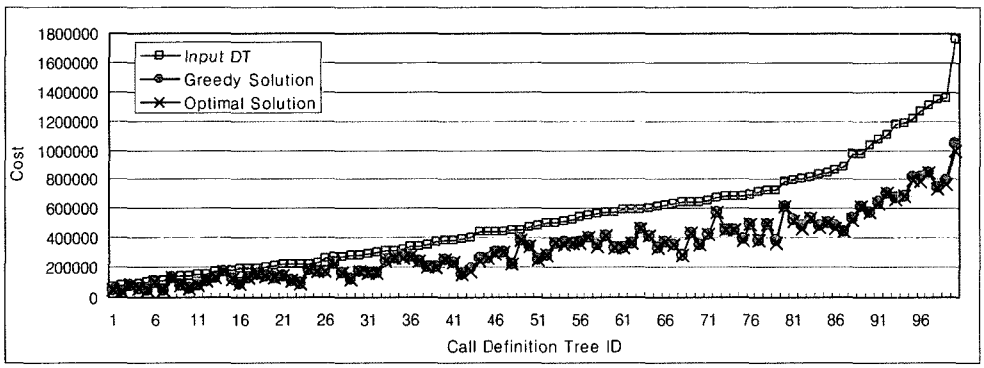
$$= O(n \log^2 n)$$

5. 성능 평가

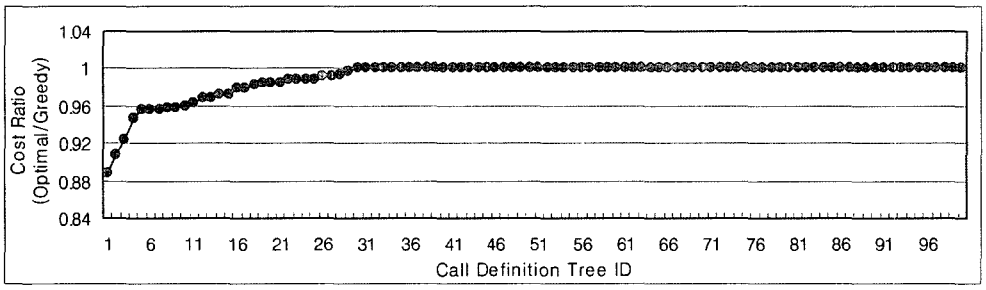
본 장에서는 4장에서 제안한 웹 서비스 호출 실행 계획에 대한 최적화 방법의 효과 및 성능을 실험을 통해 평가한다. 다양한 실험 데이타에 대해 4.3절에서 제안한 그리디 알고리즘의 효과, 즉, 그것에 의해 생성된 호출 실행 계획의 품질을 A* 알고리즘에 의해 생성된 최적 실행 계획 및 내포 결과를 이용하지 않는 호출 실행 계획과 비교 분석한다. 또 그리디 알고리즘과 A* 알고리즘의 최적화 수행 시간을 각각 측정함으로써 두 방법의 성능을 비교 평가한다.

본 실험을 위해 4장에서 제안한 A* 알고리즘과 그리디 알고리즘을 C++ 언어 및 표준 템플릿 라이브러리를 사용하여 구현하였다. 본 실험은 Intel Xeon 3.2GHz 프로세서 및 6GB 주 메모리, Red Hat Linux 9.0 운영체제를 탑재한 서버에서 수행하였다. 실험을 위한 웹 서비스 호출 정의 트리는 그것의 높이 및 내부 노드의 외향 차수를 매개변수로 하여 생성하였고, 웹 서비스에 대한 호출 비용은 1~65535 사이의 값을 임의로 할당하였다.

그림 8은 다양한 종류의 웹 서비스 호출 정의 트리들에 대한 최적화 수행 결과를 나타낸다. 이 실험에서는 3 또는 4의 높이와 0~3의 노드 외향 차수를 갖는 100개의 웹 서비스 호출 정의 트리를 생성하고 이들에 대해 A* 알고리즘과 그리디 알고리즘을 수행하여 생성된 호출 실행 트리들의 비용을 입력된 호출 정의 트리의 비용 값에 따라 정렬하였다. 실험 결과 A* 알고리즘으로부터 생성된 최적 해와 그리디 알고리즘에 의해 생성된 그리디 해는 평균적으로 입력 호출 정의 트리의 비용 대비 각각 61.7% 및 62.4%의 비용을 갖는 것으로 나타났다. 이는 내포 결과를 이용하여 웹 서비스 호출 작업에 따른 비용을 효율적으로 분산시키고 호출 주체를 최적화 함으로써 실행 비용을 크게 절약할 수 있음을 의미한다. 또, 본 실험에서 그리디 알고리즘은 전체 입력 트리의 71%에 대해 A* 알고리즘 실행 결과와 동일한 비용을 갖는 최적 해를 생성하였다. 최적 해가 아닌 경우에도 그리디 알고리즘에 의해 생성된 호출 실행 계획



(a) A* 알고리즘 및 그리디 알고리즘에 의해 생성된 호출 실행 계획의 비용



(b) 최적 해와 그리디 해의 비용의 비

그림 8 최적 해와 그리디 해의 품질 비교

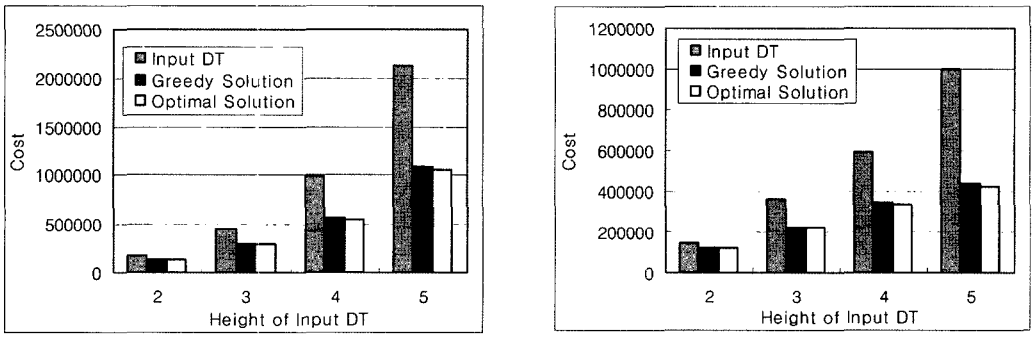
의 비용 효율성은 최적 해의 약 96.8% 수준으로, 최적 해에 매우 근접한 결과를 생성함을 알 수 있다(그림 8(b) 참조).

제안한 방법들에 의한 웹 서비스 호출 실행 비용의 절감 효과는 주어진 호출 정의 트리의 높이나 외향 차수가 증가함에 따라 더욱 향상되는 것으로 나타났다. 그림 9는 서로 다른 높이를 가진 호출 정의 트리들에 대한 최적화 실험 결과를 나타낸다. 그림 9(a)는 모든 내부 노드의 외향 차수가 2로 고정된 호출 정의 트리들에 대한 실험 결과이고, 9(b)는 내부 노드의 외향 차수가 0~3 사이의 임의의 값을 갖는 트리들에 대한 것이다. 각 실험 결과는 설정된 높이와 외향 차수를 갖는 10개의 서로 다른 호출 정의 트리들에 대한 최적화 결과의 평균 값을 의미한다. 그림 9(a)에 나타난 바와 같이, 그리디 알고리즘에 의해 생성된 호출 실행 계획의 평균 비용은 입력 호출 정의 트리의 높이가 2에서 5로 증가함에 따라 입력 트리의 평균 비용 대비 78.8%에서 51.2%로 감소하였다. 따라서 최적화에 따른 비용 절감 효과가 점점 더 증가함을 알 수 있다.

그림 10은 그림 8의 실험에서 A* 알고리즘과 그리디 알고리즘의 수행 시 소요된 CPU 처리 시간을 각각 측정하여 입력 호출 정의 트리 내의 노드, 즉, 웹 서비스의 개수에 따라 정렬한 결과를 나타낸다. A* 알고리즘은 웹 서비스의 수가 증가함에 따라 수행 시간이 매우 급격히 증가함을 알 수 있으며, 최대 93시간이 소요되었다. 반면, 그리디 알고리즘은 모든 실험 대상에 대해 10 msec 이하에 수행을 완료하여, 주어진 웹 서비스의 개수나 호출 관계의 복잡성에 대한 확장성이 우수함을 알 수 있다.

6. 결론 및 향후 연구 방향

본 논문에서는 계층적인 연동 관계가 존재하는 복합 웹 서비스들에 대한 호출을 내포 결과 전달을 통해 효율적으로 실행하기 위한 비용 기반 최적화 방안을 제안하였다. 다른 웹 서비스들에 대한 호출을 포함하는 내포 XML 데이타를 결과로 반환함으로써 실현 가능한 웹 서비스 호출 실행 계획과 이에 대한 최적화 문제를 정의하였다. 서로 다른 비용을 가지는 호출 실행 계획들 중 최소 비용을 갖는 것을 찾기 위한 전역 탐색 방법 및 그것의 문제점에 대해 분석하고 보다 효율적으로 실행될 수 있는 A* 탐색 방법을 제시하였다. 또, 빠른 시간 내에 효율적인 해를 찾을 수 있는 그리디 알고리즘



(a) fan-out = 2 (b) fan-out = 0~3

그림 9 입력 호출 정의 트리의 높이에 따른 최적화 결과의 비교

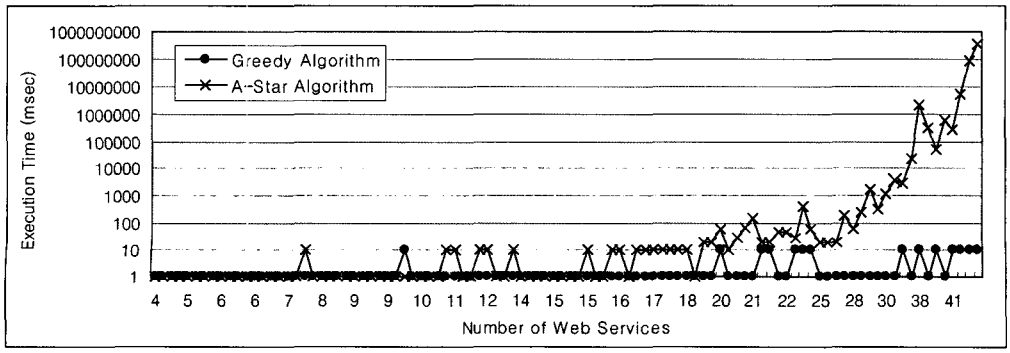


그림 10 A* 알고리즘과 그리디 알고리즘의 실행 시간 비교

을 제안하였다. 실험을 통한 평가 결과, 제안한 그리디 알고리즘은 빠른 시간 내에 최적 해에 가까운 효율적인 호출 실행 계획을 생성하며, 웹 서비스들의 개수나 호출 관계의 복잡성 증가에 대해 우수한 확장성을 갖는다. 이러한 최적화 방안을 이용함으로써 웹 서비스들의 호출 및 복귀에 따른 처리 비용을 내포 결과를 이용하여 피어 시스템들에 효율적으로 분산시키고 호출 주체를 최적화 하여 웹 서비스 시스템의 전체적인 수행 성능을 향상시킬 수 있다.

본 논문에서 제안한 최적화 방법은 모든 웹 서비스들에 대한 호출 정보 및 비용을 이용하여 전역적으로 수행된다. 개별 웹 서비스 시스템의 자율성을 좀 더 보장하기 위해서는 각 서버에서 지역적으로 수행될 수 있는 분산화된 최적화 방법에 대한 연구가 필요하다. 또 웹 서비스 호출 시 내포 XML 데이타를 매개변수로 전달하는 것을 고려한 호출 실행 계획의 생성 및 최적화에 관한 연구를 수행할 계획이다.

참 고 문 헌

[1] Thomas Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall, 2005.

[2] The Active XML Homepage. http://activexml.net/

[3] Macromedia Coldfusion MX. http://www.macromedia.com/

[4] Apache Jelly: Executable XML. http://jakarta.apache.org/commons/jelly/

[5] T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, and F. Dang Ngoc, Exchanging Intensional XML Data, In *Proc. of ACM SIGMOD*, 2003.

[6] N. Jennings and M. Wooldridge, Software Agents, *IEE Review*, 42(1), pp.17-20, Jan. 1996.

[7] Dan Davis and Manish Parashar, Latency Performance of SOAP Implementations, In *Proc. of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp.407-412, May 2002.

[8] C. Kohlhoff and R. Steele, Evaluating SOAP for High Performance Business Applications: Real-Time Trading Systems, In *Proc. of WWW'03*, 2003.

[9] N.Ruberg, G.Ruberg, and I. Manolescu, Towards cost-based optimization for data-intensive Web service computations, In *Proc. of Brazilian Symposium on Databases*, Oct. 2004.

[10] A. Tsalgatidou and T. Pilioura, An Overview of Standards and Related Technology in Web services, *Distributed and Parallel Databases*, 12(2), pp.135-162, Sept. 2002.

[11] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana, The Next Step in Web Services, *Communications of the ACM*, 46(10), pp.29-34, Oct. 2003.

[12] B. Srivastava and J. Koehler, Web Service Composition: Current Solutions and Open Problems, In *Proc. of* Int'l Workshop on Planning for Web Services, pp.28-35, 2003.

[13] Serge Abiteboul, Omar Benjelloun, Tova Milo, Ioana Manolescu, and Roger Weber, Active XML: A Data-Centric Perspective on Web Services, Technical Report, No. 381, GEMO, INRIA Futurs, 2004.

[14] S. Abiteboul, O. Benjelloun, B. Cautis, I. Manolescu, T. Milo, and N. Preda, Lazy Query Evaluation for Active XML, In *Proc. of ACM SIGMOD*, June 2004.

[15] Y.Liu, Anne H.H.Ngu, and L.Zeng, QoS Computation and Polishing in Dynamic Web Service Selection, In *Proc. of WWW2004*, 2004.

[16] N. J. Nilsson, *Artificial Intelligence: A New Synthesis*, Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1998.

박 창 섭
정보과학회논문지 : 데이타베이스
제 33 권 제 1 호 참조