위치 분석 기반
통합 클라이언트-서버 프로그램 컴파일 방법 구현

김가영 [1], 최광훈 [1], 창병모 [2]
전남대학교 전자컴퓨터공학과 [1] 숙명여자대학교 컴퓨터과학과 [2]
kirayu15@gmail.com, kwanghoon.choi@jnu.ac.kr, chang@sookmyung.ac.kr

An Implementation of Compilation Based On Location Analysis for Unified Client-Server Programs

Gayoung Kim[1], Kwanghoon Choi[1] , Byeong-Mo Chang[2]
Dept. Electronics and Computer Engineering, Channam National University[1]
Dept. of Computer Science, Sookmyung Women's University[2]

요 약

본 논문은 원격 함수 호출 위치를 분석하여 통합 RPC 프로그램을 클라이언트-서버 프로그램으로 분리 컴파일하는 방법을 구현한 내용을 소개한다. 웹 서비스는 가장 널리 사용되고 있는 플랫폼이지만, 이 서비스를 제공하기 위해 다양한 언어를 사용해야 한다는 단점이 있다. 이 문제점을 해결하기 위해 통합 프로그래밍 언어들이 제시되었다. 그 중 RPC 계산법은 위치를 지정하는 확장된 람다 계산법을 도입하였지만, 위치에 대한 유추 과정이 없어 컴파일이 매우 복잡하다. 본 논문은 이를 단순화시킨 이론을 직접 구현함으로써 이론을 뒷받침하며, RPC 프로그램을 클라이언트-서버 프로그램으로 자동 분리 컴파일 하는 과정을 설명한다. 또한 Http 통신을 이용한 런타임 시스템으로 동작을 확인한다.

1. 서 론

WWW(World Wide Web)는 인터넷에 연결된 환경이라면 어느 누구나 접근할 수 있는 공간으로 가장 널리 사용되고 있는 플랫폼이다. 이 플랫폼에 파일 전송, 원격 접속, 와이즈(WAIS) 등의 다양한 서비스를 제공하는 것을 웹 서비스라 말한다. 이 서비스를 구축하기 위해서, 개발자는 HTML로 웹 페이지의 구조를 작성하고, CSS로 HTML 페이지를 형식화 해주며, JavaScript로 클라이언트와의 상호작용을 위한 동적 기능을 추가한다. 또한 다양한 데이터를 서버에 저장하기 위해 SQL과 같은 언어를 사용해야 하며, 데이터를 구조화시켜 전달하기 위해 JSON을 이용한다. 이처럼 다양한 프로그래밍 언어를 사용한다는 것은 웹 프로그래밍을 어렵게 하며, 테스팅을 복잡하게 만든다.

이러한 문제점을 개선하기 위해 Cooper와 Wadler는 RPC 계산법(calculus) [1]을 제안하였다. 각 함수에 실행할 위치를 지정하도록 람다 계산법(lambda calculus)을 확장한 것이다. 프로그래머는 RPC 계산법으로 통합된 단일 프로그램으로 서버와 클라이언트 측의 코드를 작성할 수 있다. 이 단일 프로그램은 RPC 계산법 컴파일러를 통해 자동으로 클라이언트 코드와 서버 코드로 분리된다.

[그림 1]은 RPC 계산법을 사용한 예제 프로그램이다. 이 프로그램은 클라이언트에서 실행할 함수 getCreds와 서버에서 실행할 함수 auth를 포함한다. getCreds 함수 정의(lam)에 ^c를 붙이고, auth 함수 정의에 ^s를 붙여 각 함수에 실행할 위치를 지정한다. 이 예제 프로그램을 클라이언트에서 실행하면 먼저 12번째 줄의 서버 함수 auth를 호출한다. 서버에서 6번째 줄의 클라이언트 함수 getCreds를 호출하여 사용자 아이디와 패스워드를 입력 받는다. getCreds 함수의 형식 인자 prompt는 실인자 "Enter name, pwd: " 문자열을 전달받아 print 함수를 통해 사용자에게 이 문자열을 보여주고, read 함수로 사용자로부터 입력을 받아 서버측 지역변수 creds로 반환한다. 7번째 줄 if 문에서 creds가 문자열 "ezra:opensesame"와 비교하여 두 문자열이 같다면 "the secret document"를, 같지 않다면 "Access denied"를 클라이언트에 반환한다.

```
01: getCreds = lam^c prompt.
02:   let y = print prompt in
03:     read
04:
05: auth = lam^s x.
06: let creds =
      getCreds "Enter name, pwd: " in
07:   if creds == "ezra:opensesame" then
08:       "the secret document"
09:   else
10:       "Access denied"
```

```
11:
12: main = auth ()
```

그림 1 RPC 프로그램 예제

Choi와 Chang [2]은 함수 호출에서 위치 정보를 타입으로 파악하여 기존 RPC 계산법의 복잡한 구현 방법을 단순화 시킨 컴파일 방법을 제안하였다. 이 타입 기반 컴파일 방법이 정확함을 증명하였지만 본 논문에서 실제로 구현해보임으로써 그 증명을 뒷받침하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서 통합 클라이언트-서버 프로그래밍 방법과 관련된 연구를 소개한다. 3장에서는 타입 기반 위치 정보를 활용한 RPC 프로그램의 컴파일 방법에 대한 상세 내용을 설명하고, 4장에서 결론을 맺는다.

2. 관련연구

통합 프로그래밍 언어에 대한 기존 연구는 다음과 같다. RPC 계산법[1]은 람다 계산법에 위치 정보를 추가함으로써 통합된 단일 프로그램으로 서버와 클라이언트 코드를 작성할 수 있다. 이를 기반으로 만들어진 언어가 Links[3]로, 데이터베이스 쿼리 최적화, 연속성, 메시지 전달 및 동시성, XML 프로그래밍을 통합한 함수형 통합 웹 프로그래밍 언어이다.

Ur/Web[4], Hop[5]은 통합 웹 프로그래밍 언어로, 클라이언트에서 서버 함수를 호출할 수 있지만, 서버에서 클라이언트 함수를 호출할 수 없는 비 대칭형 통신만 제공한다. 서버에서 클라이언트의 함수를 호출하기 위해서는 외부 라이브러리를 이용해야 하기 때문에, 해당 언어에서 이와 관련된 구문(Syntax)을 제공하지 않는다.

Cooper와 Wadler의 연구[1,3]에서는 서버 상태를 유지하지 않기(Stateless) 때문에, 반복적으로 클라이언트와 서버 간의 호출이 있을 경우 통신상의 오버헤드가 발생할 수 있다. 데이터베이스 쿼리와 같이 Stateless 프로그램에서 인코딩하기 어려운 프로그램도 존재하기에 Choi와 Chang[2]은 이를 해결할 방법으로 Stateful 람다식과 위치 정보를 파악하여 RPC 계산법의 구현을 단순화시킬 컴파일 방법을 제안하였다. 또한, 다른 언어에서는 클라이언트에서 서버의 함수를 호출하는 비 대칭형 통신을 지원하지만, Choi와 Chang의 연구에서는 클라이언트에서 서버를 호출할 수 있고 서버에서 클라이언트 함수를 호출할 수 있는 대칭형 통신을 지원한다.

본 논문에서는 Choi와 Chang의 연구에서 제안한 이론을 뒷받침하기 위해 Parser, 위치 유추 알고리즘, 컴파일러를 구현했으며, HTTP 기반의 런타임 시스템을 구현하여 프로그램이 실제로 동작함을 확인한다.

3. 타입 기반 위치 정보를 활용한 RPC 프로그램 컴파일 방법

본 논문에서 제안하는 타입 기반 RPC 프로그램 컴파일 방법을 설명한다. [그림 2]는 RPC 프로그램이 클라이언트-서버 프로그램으로 분리되는 과정을 도식화한 것이다. 첫 번째 단계에서는 프로그래머가 작성한 RPC 프로그램의 타입 유추(type inference)를 통해 함수를 실행해야하는 위치를 결정한다. 두 번째 단계에서는 타입 정보를 이용하여 RPC 프로그램에 나타난 각 함수 호출이 지역인지 원격인지를 판단하여 컴파일 변환한다. 세 번째 단계에서는 클라이언트 프로그램과 서버 프로그램으로 분리한다.

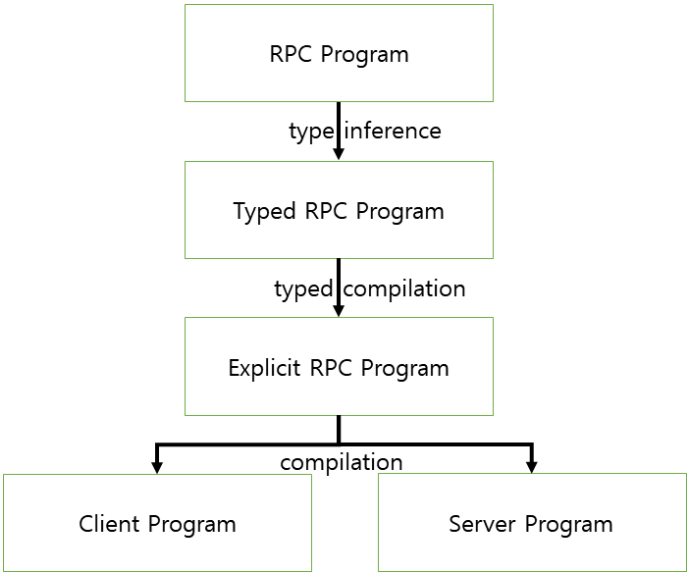

그림 2 클라이언트-서버 프로그램으로 분리하는 과정

앞으로 1장에서 제시한 [그림 1]의 RPC 프로그램 예제를 이용해 설명한다.

3.1 타입 유추(Type Inference) 과정

첫 번째 단계에서는 프로그래머가 작성한 RPC 프로그램의 타입 유추(type inference)를 통해 함수를 실행해야하는 위치를 결정한다. 원격 함수 호출과 지역 함수 호출을 구별하기 위해 이 과정이 필요하다.

[2] 논문에서 위치 정보를 갖는 함수 타입을 다음과 같이 제안하였다.

t1-loc->t2

t1은 함수의 인자 타입이고, t2는 함수의 결과 타입이다. 그리고 loc는 c 또는 s이고, c는 클라이언트, s는 서버를 가리킨다. 이 타입을 갖는 함수는 해당하는 위치(loc)에서 실행됨을 의미한다.

예를 들어, [그림 1]의 1번째 줄의 getCreds는 문자열을 인자로 받고 문자열을 반환하는 함수이고, 'lam^c'로 작성하여 클라이언트에서 실행하도록

정의되었기 때문에 String-c->String의 함수 타입을 갖는다. 5번째 줄의 서버 함수 auth는 void 인자를 받아 문자열을 반환하기 때문에 Void-s->String의 함수 타입을 갖는다.

이와 같은 방식으로 타입 유추 과정을 거치면 [그림 3]와 같은 결과를 얻을 수 있다. 이 과정에서 유추한 타입을 볼드체(Bold)로 구분하여 표시하였다. 각 변수의 타입은 변수 선언에서 ':' 다음에 온다. 예를 들어, 'getCreds: String-c->String = …'는 getCreds 함수의 타입이다. '^c^' 또는 '^s^'는 호출할 함수가 실행될 위치가 클라이언트 또는 서버임을 나타낸다. 예를 들어, 서버에서 클라이언트 함수를 호출하는 코드 'getCreds ^c^ "Enter name, pwd: " '의 ^c^는 getCreds 함수가 실행되어야 할 위치가 클라이언트임을 나타낸다. 클라이언트에서 서버 함수를 호출하는 코드 'auth ^s^ ()'의 ^s^는 auth 함수가 서버에서 실행되어야 함을 나타낸다.

프로그래머가 작성한 RPC 프로그램에서는 함수 정의에만 위치를 지정할 뿐, 함수 호출 코드에 위치 정보를 지정하지는 않는다. 타입 유추 과정에서 이 정보를 자동으로 찾아낸다.

```
getCreds: String-c->String =
  lam^c prompt: String.
    let y: Unit = print ^c^ prompt in
      read
auth: Void-s->String =
  lam^s x: Unit.
    let creds: String =
      getCreds ^c^ "Enter name, pwd: " in
      if creds == "ezra:opensesame" then
        "the secret document"
      else
        "Access denied"

main: String = auth ^s^ ()
```

그림 3 Typed RPC Program

3.2 타입 기반 위치 정보를 활용한 컴파일 과정

두 번째 단계에서는 타입 정보를 이용하여 RPC 프로그램에 나타난 각 함수 호출 'f arg'가 지역인지 원격인지를 판단하여 컴파일 변환한다. 구체적으로, 클라이언트에서 클라이언트 함수를 호출하면 지역 함수 호출로 판단하고, 서버 함수를 호출하면 원격 함수 호출로 판단한다. 서버에서 클라이언트 함수를 호출하면 원격 함수 호출로 판단하고, 서버 함수를 호출하면 지역 함수 호출로 판단한다. 이 내용을 다음의 [표 1]로 요약할 수 있다.

표 1 위치에 따른 함수 호출 구분

	클라이언트 함수	서버 함수
클라이언트에서 호출	(1) 지역 함수 호출	(2) 원격 함수 호출
서버에서 호출	(3) 원격 함수 호출	(4) 지역 함수 호출

[표 1]로 요약된 함수 호출 종류로 판단 내린 다음, 컴파일 변환을 다음과 같이 한다. (2)번의 경우, 'f arg' 형태의 함수 호출을 'Req f arg' 형태의 원격 함수 호출로 변환한다. (3)번의 경우, 'f arg' 형태의 함수 호출을 'Call f arg' 형태의 원격 함수 호출로 변환한다. 나머지 (1)번과 (4)의 경우 'f arg' 함수 호출의 형태를 유지한다. 위에서 설명한 컴파일 변환을 [표 2]와 같이 요약할 수 있다.

표 2 함수 호출 방법에 따른 컴파일 변환 방법

컴파일 변환	컴파일 전	컴파일 후
(2)	f arg	Req f arg
(3)	f arg	Call f arg
(1), (4)	f arg	f arg

예를 들어, [그림 3]의 'auth ^s^ ()'가 (2)에 해당하는 컴파일 변환을 적용한다. 적용한 결과는 아래와 같다.

컴파일 전	컴파일 후
auth ^s^ ()	Req auth ()

'getCreds ^c^ "Enter name, pwd: "'는 (3)에 해당하는 컴파일 변환을 적용할 수 있다. 적용한 결과는 아래와 같다.

컴파일 전	컴파일 후
getCreds ^c^ "Enter name, pwd: "	Call getCreds "Enter name, pwd: "

'print ^c^ prompt'는 (1)에 해당하는 지역함수 호출이다. 따라서 컴파일 변환에서 그대로 둔다.

컴파일 전	컴파일 후
print ^c^ prompt	print prompt

[그림 3]에서는 (4)에 해당하는 경우는 없지만, 위의 (1)번의 경우와 같이 그대로 두는 컴파일 변환을 적용한다.

[그림 3]의 예제를 입력 받아 두 번째 단계 컴파일 변환 과정을 거치면 [그림 4]와 같은 결과를 얻을 수 있다. 컴파일 변환 과정 전후의 가장 큰 차이점은 원격 함수 호출을 Req와 Call로 명시적으로 표시한 것이다. Req로 표시된 원격 함수 호출은 클라이언트에서 서버 함수를 호출하고, Call로 표시된 원격 함수 호출은 서버에서 클라이언트 함수를 호출하는 것을 구분한 것이다.

```
getCreds = lam^c prompt.
```

```
  let y = print prompt in
    read

auth = lam^s x.
  let creds =
    (Call getCreds "Enter name, pwd: ") in
    if creds == "ezra:opensesame" then
      "the secret document"
    else
      "Access denied"

main = (Req auth ())
```

그림 4 RPC Program with Req and Call

3.3 클라이언트-서버 프로그램으로 분리

세 번째 단계에서는 클라이언트 프로그램과 서버 프로그램으로 분리한다. 컴파일 단계를 거친 프로그램은 여전히 하나의 프로그램이기 때문에, 클라이언트-서버 프로그램으로 분리해야 한다.

프로그램 내에 존재하는 함수가 정의된 위치에 따라 분리된다. 즉, 클라이언트 위치로 지정된 함수는 클라이언트 측의 함수 리스트에, 서버 위치로 지정된 함수는 서버 측의 함수 리스트로 분리된다.

[그림 4]의 예제 프로그램을 입력 받아 앞에서 설명한 분리 컴파일 과정이 끝나면 [그림 5]와 같은 클라이언트 함수 리스트와 서버 함수 리스트를 출력한다. 클라이언트 함수 리스트에서 메인 프로그램은 main이다. [그림 5]의 예제에서 함수를 표현하는 클로저를 사용한다. 클로저(Closure), 'Clo f {fv1, fv2, …, fvn}'는 함수 코드 f와 환경(Environment)이다. 환경은 함수 코드에서 참조하는 자유 변수(free variable), fv1, fv2, …, fvn의 값을 저장한 자료구조이다.

```
Client function list
main = (Req auth ())
getCreds = (Clo _gf1 {read, print})
_gf1 = {read, print} lam^c prompt.
  let y = print prompt in
    read
_gf2 = {getCreds} lam^c z1.
  let y1 = getCreds z1 in Ret y1

Server function list
auth = (Clo _gf3 {getCreds})
_gf3 = {getCreds} lam^s x.
  let creds = (Call (Clo _gf2 {getCreds})
                    "Enter name, pwd: ")
  in
    if creds == "ezra:opensesame" then
      "the secret document"
    else
      "Access denied"
```

그림 5 Stateful Client-Server Program

3.4 클라이언트-서버 통신 구현 방법

Http 프로토콜을 기반으로 클라이언트 프로그램과 서버 프로그램 간의 원격 함수 호출을 지원하는 런타임 시스템을 구현하였다. Req 원격 함수 호출은 Http 프로토콜의 요청(Request)으로 구현하고, Call 원격 함수 호출은 Http 프로토콜의 응답(Response)로 구현하였다. 지역 함수 호출은 통신 방법과 무관하게 일반 함수 호출로 구현하였다.

4. 결 론

이 논문에서 원격 함수 호출 위치를 분석하여 통합 RPC 프로그램을 클라이언트-서버 프로그램으로 자동 분리 컴파일하는 방법을 구현하였다. 분리된 클라이언트 프로그램과 서버 프로그램 간의 통신을 지원하기 위한 Http 프로토콜 기반의 런타임 시스템을 구현하였으며, 분리 컴파일 된 프로그램이 잘 동작함을 확인했다.

5. 참고문헌

[1] E. Cooper, P. Wadler, "The RPC Calculus," Proceedings of the 11th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 2009.

[2] K. Choi, B.M. Chang, "A Theory of RPC Calculi for Client-Server Model," To Appear in Journal of Functional Programming, Cambridge University Press, 2019.

[3] E. Cooper, S. Lindley, P. Wadler, J. Yallop, "Links: Web Programming Without Tiers," International Symposium on Formal Methods for Components and Objects(FMCO '06), November 2006.

[4] A. Chlipala, "Ur/Web: A Simple Model for Programming the Web," Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Language, January 2015.

[5] M. Serrano, G. Berry, "Multitier Programming in Hop," Communications of the ACM, Vol. 55, No. 8, pp. 53-59, 2012.