ISSN 2586-4599

2025 한국소프트웨어종합학술대회 논문집

2025년 12월 16일(화)~19일(금)
여수엑스포컨벤션센터

한국정보과학회
Korean Institute of Information Scientists and Engineers

프로그래밍언어

학부생논문

멀티티어 액터 프로그래밍 언어의 설계 및 구현

이현진[01] 최광훈[1]

[1]전남대학교 인공지능융합학과

hyeonjin@jnu.ac.kr, kwanghoon.choi@jnu.ac.kr

Design and Implementation of a Multitier Actor Programming Language

Hyeonjin Lee[01] Kwanghoon Choi[1]

[1]Dept. of Artificial Intelligence Convergence, Chonnam National University

hyeonjin@jnu.ac.kr, kwanghoon.choi@jnu.ac.kr

요 약

본 연구는 분산 시스템의 복잡성과 통신 정합성 문제를 해결하기 위해, 액터 모델과 RPC 계산법을 통합한 멀티티어 액터 언어(MAPL)을 제안한다. MAPL은 원격 상호작용을 RPC로 추상화하여 통신 정합성을 보장하고 분산 연산을 로컬 연산과 동일하게 취급한다. 이는 분산 프로그램의 순차적 흐름 기술을 가능하게 함으로써, 물리적으로 분산된 로직에 대한 전역적 모듈성을 제공한다. 본 언어는 Haskell 및 Cloud Haskell 기반으로 구현되었으며, ZKP 인증 프로토콜과 멀티티어 벤치마크를 통해 표현력을 확인하였다.

1. 서 론[1]

분산 프로그래밍은 다수의 컴퓨터에서 작업을 나눠 처리하는 현대 시스템의 핵심 기술로, 대표적으로 액터 모델 [1,2]이 널리 사용된다. 액터 모델은 독립된 메모리와 비동기 메시지를 이용한 높은 동시성을 제공하지만 두 가지 한계를 갖는다. 첫째, 메시지 송·수신 짝이 맞는지 보장하지 않아 통신 정합성이 깨지기 쉽다. 둘째, 로직이 여러 액터에 분산되어 전체 시스템의 동작을 전역적 관점에서 이해하거나 모듈화하기 어렵다. 한편, 멀티티어 프로그래밍[3-7]은 분산 로직을 하나의 전역 구조로 기술하는 접근법이다. 특히 RPC 계산법 [3-5]은 함수에 실행 위치를 붙여 원격 함수 호출을 지원하며, 호출-응답 구조를 통해 통신 정합성을 보장한다.

```
let s = proc a ( f )
          proc b ( g )
            proc c ( x )
              ( f x )  ( g x ) in

let k = proc a ( y )
          proc b ( z )  y in

print ( s k k 1 )
```

그림 1 분산된 위치의 S K 조합자 – RPC 계산법

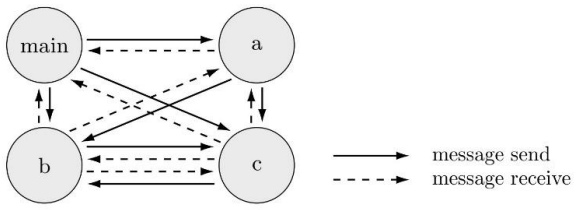

그림 2 분산된 위치의 S K 조합자 – 순수 액터 기반

*본 연구는 과학기술정보통신부 및 정보통신기획평가원의 인공지능융합혁신인재양성사업 연구 결과로 수행되었음(IITP-2023-RS-2023-00256629)

*이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(RS-2025-24523420)

*본 연구는 한국인터넷진흥원(KISA)-정보보안 특성화대학 지원사업의 지원을 받아 수행된 연구임

또한 여러 위치의 코드를 계층 간 모듈로 구성할 수 있어 분산된 기능을 단일 구조로 조직화할 수 있다. 그러나 RPC 계산법은 위치의 동적 생성을 지원하지 않기 때문에 액터 기반 시스템의 유연한 구조를 그대로 표현하기에는 한계가 있다.

그림 1의 RPC 계산법 예제는 서로 다른 위치에 배치된 연산(S, K 조합자)을 단일 프로그램 안에서 정의하며 proc loc (x) exp 는 지정된 위치 loc에서 표현식 exp를 평가하는 위치 지정 함수로 원격 함수 호출을 지원한다. 예를 들어 f는 위치 a에서, g는 위치 b에서 실행되지만, (f x) (g x) 와 같이 위치 c에서 참조될 수 있다. 반면 동일한 계산을 순수 액터 모델로 작성하면 그림 2처럼 복잡한 메시지 흐름을 직접 구성해야 한다.[i] 이는 RPC 계산법이 분산 상호작용을 더 간결하게 추상화함을 보여준다.

따라서 본 연구는 액터 모델과 RPC 계산법을 통합하여 분산 프로그램을 전역적 관점으로 기술하며 원격 함수 호출 기반의 통신 정합성을 제공하는 멀티티어 액터 언어(MAPL)를 제안한다.

2. 멀티티어 액터 프로그래밍 언어 (MAPL)

MAPL의 핵심 설계 원리는 액터를 RPC 계산법의 위치(location)와 동일시하는 것이다. 액터는 생성 시 고유한 액터 ID를 부여받으며, 이는 해당 액터의 기본 실행 장소로 사용된다. 위치 지정 함수는 이러한 액터 ID를 이용해 실행 지점을 명시하고, 위치 주석이 생략되면 함수가 속한 렉시컬 스코프의 액터가 실행 위치가 된다.

MAPL은 THREADS 언어 [8]와 Agha의 액터 계산법(new, send, ready)을 통합한 기반 위에, 위치 지정 함수, 위치 정보를 포함한 확장된 환경, 원격 변수 접근과 원격 함수 생성·호출 기능을 추가해 멀티티어 실행 모델을 구성한다. 런타임에서는 확장된 환경의 위치 정보를 바탕으로 원격 호출에 필요한 메시지 교환을 자동으로 수행하여, 분산 연산을 로컬 연산과 동일한 형태로 표현할 수 있다.

연산 의미론은 멀티티어 액터 계산법으로 정의되었으며 순수 Haskell 인터프리터로 구현된 후 실제 분산 환경 지원을 위해 Cloud Haskell [ii] 기반으로 이식되었다.[i] Cloud Haskell은 노드(물리적인 실행 단위)와 액터(논리적 실행 단위)를 분리해 관리하며, 지정된 역할(role)에 따라 액터를 적절한 노드에 배치한다.

이를 바탕으로 proc @a (x) exp 형태의 매개변수화된 위치(parameterized location) 구문을 제공하여 역할을 가진 노드가 온라인 상태가 될 때 자동으로 해당 위치에 클로저를 생성한다. 이는 채팅 시스템과 같이 동적 액터 환경을 지원하며 기존 RPC 계산법이 갖는 정적 위치 가정의 한계를 해결한다.

3. 사례 연구

사례 연구로 ZKP(Zero Knowledge Proof) 기반 인증 프로토콜을 구현하였다.[i] 프로토콜은 초기화-챌린지-응답-검증의 네 단계로 구성되며, 그림 3과 그림 4는 Prover와 Verifier의 통신 구조와 MAPL 코드 구조를 보여준다.

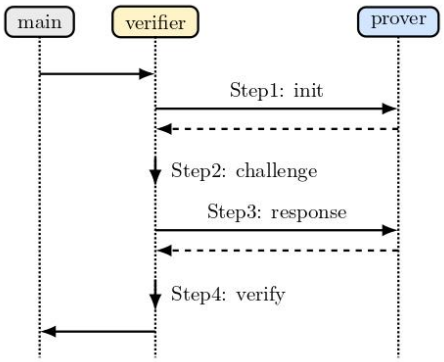

그림 3 핵심 통신 과정

```
proc(main)
  let p = 23 in        // verifier와 prover가 공유
  let g = 5 in
  let commit = ... in  // prover의 공개키
  ...
  let prover   = new ( ... ) in   // prover 액터
  let verifier = new ( ... ) in   // verifier 액터
  ...

  let init = proc prover ()
    let t = powMod g r p ...

  let challenge = proc verifier ()
    ...

  let verify = proc verifier (s, t, c)
    let left = powMod g s p ...

let login = proc verifier ()
  let (response, t) = init () in   // Step 1
  let c = challenge () in          // Step 2
  let s = response (c) in          // Step 3
  verify (s, t, c)                 // Step 4

in  login ()
```

그림 4 ZKP 인증 로그인 구현 코드

MAPL에서는 두 역할의 로직을 위치 지정 함수를 통해

기존 액터 모델에서 파편화되기 쉬운 상호작용 프로토콜을 효과적으로 모듈화한다. 또한 함수 호출을 통해 통신 정합성이 자연스럽게 확보되어, 개발자는 통신 로직보다 프로토콜의 논리 구현에 집중할 수 있다. 아울러 네 가지 멀티티어 벤치마크 (Chat·Token Ring·Master-Worker·HO-Book-Seller) 모두 성공적으로 구현되고 실행되었다.[i] 이는 MAPL이 멀티티어 프로그래밍의 표현력을 손실 없이 통합함을 보여준다.

4. 관련 연구

분산 시스템에서 통신 일관성을 보장하기 위한 다양한 접근 방식이 있다. ScalaLoci [6,7]는 퓨처(futures)를 이용한 원격 블록 실행과 배치 타입을 통해 이를 해결한다. 안무 프로그래밍 [9-13]은 send 및 receive를 직접 사용하는 대신 하나의 추상화된 통신 구문을 사용하며, 이는 컴파일 시점에 각 노드를 위한 로컬 프로그램으로 프로젝션된다. RPC 계산법 [3-5]은 위치 지정 함수와 다형적 위치 타입을 핵심 기능으로 제공하며 세션 타입 [14]은 명시적인 send 및 receive 구문을 사용하되, 통신 순서와 프로토콜을 정확히 반영하는 고급 타입 시스템을 통해 정합성을 보장하는 방식이다.

5. 결론 및 향후 연구

본 연구는 액터를 실행 위치로 간주하여 액터 모델의 동적 생성 능력과 RPC 계산법의 안정적 통신 구조를 결합한 MAPL을 제안하였다. MAPL은 원격 상호작용을 로컬 연산처럼 추상화하여 복잡한 통신 코드를 제거하고, 이를 통해 개발자는 분산 시스템을 마치 순차 프로그램처럼 직관적으로 기술할 수 있다. 결과적으로 본 언어는 통신 정합성을 구조적으로 보장할 뿐만 아니라, 물리적으로 분산된 로직을 논리적으로 통합하여 전역적 모듈성을 제공한다.

향후 연구로는 비동기 메시지 상호작용의 정합성을 보장하기 위한 Mailbox-oriented 타입 시스템과, 위치 간 정보 흐름을 제어하는 Information-flow 타입 시스템을 도입하여 MAPL의 안전성을 강화할 계획이다. 이를 통해 일반 분산 애플리케이션과 보안 프로토콜을 일관된 방식으로 표현할 수 있는 언어로 확장하고자 한다.

참고 문헌

[1] Gul Agha, Actors: a model of concurrent computation in distributed systems, MIT press, 1986

[2] Hans Svensson, Lars-Åke Fredlund, and Clara Benac Earle, A unified semantics for future Erlang, Proceedings of the 9th ACM SIGPLAN Workshop on Erlang (Erlang '10), 23-32, 2010

[3] Kwanghoon Choi and Byeong-Mo Chang, A theory of RPC calculi for client-server model, Journal of Functional Programming, 29, e5, 2019

[4] Kwanghoon Choi, James Cheney, Simon Fowler, and Sam Lindley, A polymorphic RPC calculus, Science of Computer Programming, 197, 102499, 2020

[5] Kwanghoon Choi, James Cheney, Sam Lindley, and Bob Reynders, A Typed Slicing Compilation of the Polymorphic RPC calculus, Proceedings of the 23rd International Symposium on Principles and Practice of Declarative Programming (PPDP '21), Article 11, 2021

[6] Pascal Weisenburger, Mirko Köhler, and Guido Salvaneschi, Distributed System Development with ScalaLoci, Proc. ACM Program. Lang., 2, OOPSLA, Article 129, 2018

[7] Pascal Weisenburger, Johannes Wirth, and Guido Salvaneschi, A Survey of Multitier Programming, 53, 4, Article 81, 2020

[8] Daniel P. Friedman and Mitchell Wand, Essentials of Programming Languages (third ed.), The MIT Press, Cambridge, Massachusetts, 2008

[9] Marco Carbone and Fabrizio Montesi, Deadlock-freedom-by-design: multiparty asynchronous global programming, SIGPLAN Not., 48, 1, 263-274, 2013

[10] Luís Cruz-Filipe, Eva Graversen, Lovro Lugović, Fabrizio Montesi, and Marco Peressotti, Functional Choreographic Programming, Theoretical Aspects of Computing – ICTAC 2022, 212-237, 2022

[11] Andrew K. Hirsch and Deepak Garg, Pirouette: higher-order typed functional choreographies, Proc. ACM Program. Lang., 6, POPL, Article 23, 2022

[12] Gan Shen, Shun Kashiwa, and Lindsey Kuper, HasChor: Functional Choreographic Programming for All (Functional Pearl), Proc. ACM Program. Lang., 7, ICFP, Article 207, 2023

[13] Lovro Lugovic and Fabrizio Montesi, Real-World Choreographic Programming: Full-Duplex Asynchrony and Interoperability, Art Sci. Eng. Program., 8, 2, 2024

[14] Ancona et al. Behavioral Types in Programming Languages. 2016

[i] https://github.com/chaendaya/ksc2025

[ii] distributed-process 0.7.8, network-transport-tcp 0.8.0