Journal of KIIT. Vol. 17, No. 7, pp. 31-45, Jul. 31, 2019. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2019.17.7.31

Rete 알고리즘의 병렬 및 분산 처리에 관한 기존 연구 분석

김 재 훈*

An Analysis of Existing Studies on Parallel and Distributed Processing of the Rete Algorithm

Jaehoon Kim*

이 논문은 2018년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임.
(No. 2016R1D1A1B03930564).

요 약

현재 지능적 서비스의 핵심 기술은 딥러닝 즉 신경망, 그리고 GPU 병렬 컴퓨팅 및 빅 데이터와 같은 병렬 분산 처리 기술이다. 하지만 미래의 전 세계적으로 공유된 온톨로지를 통한 지능적 서비스 및 지식 공유 서비스에서는 지식의 표현 및 추론을 위하여 신경망보다 더 나은 방법이 있다. 그것은 시맨틱 웹의 표준 규칙 언어인 RIF 혹은 SWRL의 IF-THEN의 지식 표현이며, 이러한 규칙을 rete 알고리즘을 이용하여 효율적으로 추론할 수 있다. 하지만 단일 컴퓨터에서 동작하는 rete 알고리즘의 처리 규칙 수가 100,000개가 될 경우 그 성능이 수 십 분으로 매우 안 좋아지며, 분명한 한계가 존재한다. 따라서 본 논문에서는 rete 알고리즘의 병렬 및 분산 처리에 대한 과거로부터 현재까지의 연구 내용을 정리 분석하며, 이를 통해 효율적인 rete 알고리즘의 구현을 위해 어떤 측면들이 고려되어야 하는지를 살펴본다.

Abstract

The core technologies for intelligent services today are deep learning, that is neural networks, and parallel and distributed processing technologies such as GPU parallel computing and big data. However, for intelligent services and knowledge sharing services through globally shared ontologies in the future, there is a technology that is better than the neural networks for representing and reasoning knowledge. It is a knowledge representation of IF-THEN in RIF or SWRL, which is the standard rule language of the Semantic Web, and can be inferred efficiently using the rete algorithm. However, when the number of rules processed by the rete algorithm running on a single computer is 100,000, its performance becomes very poor with several tens of minutes, and there is an obvious limitation. Therefore, in this paper, we analyze the past and current studies on parallel and distributed processing of rete algorithm, and examine what aspects should be considered to implement an efficient rete algorithm.

Keywords
rete algorithm, parallel, distributed, Hadoop, spark, GPU

* 서일대학교 정보통신공학과
- ORCID: https://orcid.org/0000-0002-0005-4506

· Received: Apr. 25, 2019, Revised: Jul. 15, 2019 , Accepted: Jul. 18, 2019
· Corresponding Author: Jaehoon Kim
Dept. of Information & Communication Engineering, Seoil University, 28, Yongmasan-ro 90 gil, Jungnang-gu, Seoul, 02192, Korea
Tel.: +82-2-490-7258, Email: jhkimygk@gmail.com

Ⅰ. 서 론

과거 생성 시스템(Production system) 혹은 규칙 기반 전문가 시스템(Rule-based expert system)에서 주로 사용되는 rete 알고리즘[1]이 최근 다시 관심을 받고 있다. 그 이유는 대량의 온톨로지 및 규칙을 위한 효율적 추론 방법이 되며[2], 또한 모든 분야에서의 지능적 서비스(Intelligent services)라는 현재 IT 패러다임과 관련이 있다고 할 수 있다.

현재 지능적 서비스의 핵심 기술은 딥러닝 즉 신경망, 그리고 GPU 병렬 컴퓨팅 및 빅 데이터와 같은 병렬 분산 처리 기술이다. 즉, 과거에는 상상 못했던 모든 상황의 데이터를 빅 데이터 기술로 수집하고, 이를 GPU 병렬 컴퓨팅 및 빅 데이터 기술을 이용하여 빠른 시간 안에 최적의 신경망으로 학습하는 것이다.

하지만 미래의 전 세계적으로 공유된 온톨로지(Globally shared ontologies)를 통한 지능적 서비스 및 지식 공유 서비스[3][4]에서는 지식의 표현 및 추론(Reasoning)을 위하여 신경망보다 더 나은 방법이 있다. 그것은 시맨틱 웹의 표준 규칙 언어인 RIF[5] 혹은 SWRL[6]의 IF-THEN의 지식 표현이며, 이러한 규칙을 rete 추론 알고리즘을 이용하여 효율적으로 추론할 수 있다.

최근 대량의 규칙 및 온톨로지 데이터를 추론하기 위하여 Hadoop 혹은 Spark와 같은 빅 데이터 기술을 활용해 보고자 하는 몇몇의 연구들[7]-[12]이 있다. 이러한 연구들을 분석해 보면 모든 연구들이 공통적으로 rete 알고리즘의 향상된 추론 성능을 제시하는 것을 확인할 수 있다. 따라서 본 논문에서는 과거 rete 알고리즘의 병렬 및 분산 처리로부터 현재의 빅 데이터 기술을 이용한 병렬 및 분산 처리까지 각 방법들이 어떤 측면에서 발전해 왔는지를 살펴보며, 대량의 규칙 및 온톨로지 데이터의 추론을 위하여 어떤 측면들이 고려되어야 하는지를 살펴본다.

이러한 rete 알고리즘에 대한 최근까지의 연구 내용에 대한 정리는 rete 알고리즘의 병렬 및 분산 처리 기술 개발에 관심 있는 연구자들에게 다른 시각에서의 참고자료로 활용될 수 있을 것이다.

1.1 몇 가지 전제

본 논문은 이전 연구[3][4]에서 소개된 RIF 규칙을 이용한 지식 공유 서비스에서의 규칙 추론 성능 향상과 관련된 연구이다. 우리는 그러한 서비스를 SRS(Shared Rules Service)라고 하였다.

그림 1은 SRS의 개념을 보여준다[3]. SRS에서는 사용자들이 공유된 온톨로지를 이용하여 자신들만의 지식을 RIF 규칙으로 표현하고, 이를 웹에 공유하여 다른 사용자들이 자신이 모르는 유용한 정보를 추론 하도록 하는 것이다.

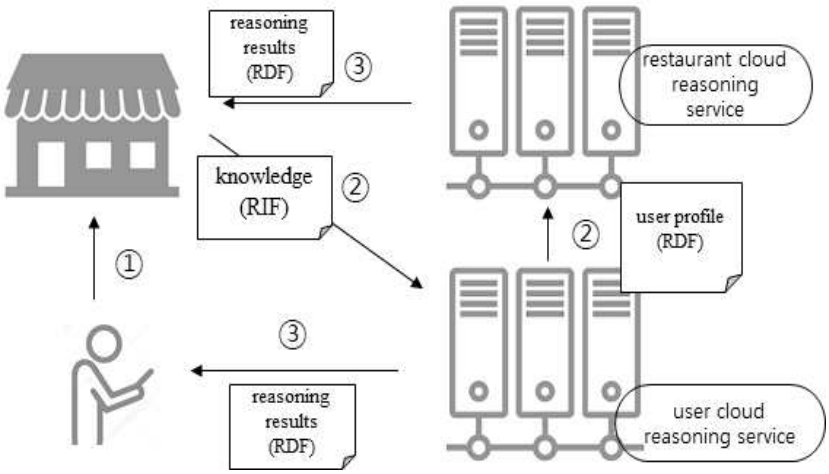


그림 1. SRS의 개념
Fig. 1. Concept of SRS

예로, 그림에서 사용자의 프로파일, 즉 RDF 데이터는 그 사용자가 가입한 클라우드 추론 서비스에서 관리되고 있고, 사용자가 음식점에 들어설 경우 ②와 같이 음식점과 관련하여 다른 사용자들이 작성한 RIF 규칙이 해당 사용자의 추론 서비스로 전송되어 해당 사용자를 위한 유용한 정보를 추론한다. 또한 해당 사용자의 공개가 허용된 일부 프로파일 정보는 ②와 같이 동시에 음식점 추론 서비스로 전송되어 유용한 정보를 추론할 수 있다. 만약 해당 사용자가 특정 질병을 가지고 있다면 사용자가 몰랐던 음식에 대한 유용한 정보를 제공받을 수 있을 것이다.

이러한 공유된 규칙은 질병과 음식점이라는 문제 영역에서 뿐만이 아니라 우리 일상생활에서의 다양한 영역에 걸쳐 작성될 수 있으며, 따라서 개인이 제공받은 공유된 규칙의 수는 매우 클 수 있다. 따라서 대량의 규칙 및 온톨로지 데이터에 대한 가능한 신속하며 효율적인 추론이 요구된다.

1.2 RIF 규칙

SRS에서는 지식 공유를 위한 규칙 언어로 시맨틱 웹의 표준 규칙 언어인 RIF[5]를 고려한다. RIF는 크게 RIF-Core, RIF-BLD(Basic Logic Dialect), RIF-PRD(Production Rule Dialect)로 나누어진다. RIF-BLD는 혼 논리(Horn logic)와 일치하며, RIF- PRD는 생성 시스템의 특징을 갖는 규칙이며, RIF- Core는 RIF-BLD와 RIF-PRD의 공통부분을 정의한다. 본 연구에서는 기본적으로 RIF-BLD를 고려하며, 하지만 규칙의 조건들에 대한 부정(Negation) 그리고 결론에 여러 개의 assert문을 명세할 수 있는 RIF-PRD의 일부 특성을 함께 고려한다.

▪ 정의 1 (RDF 트리플 패턴) [3]

그림 2와 같이 RDF 그래프 $G = (V, E)$를 고려하자. $v, u \in V$는 노드를 나타내고, $e \in E$는 v와 u 사이의 에지를 나타낸다. RDF 트리플은 G에서의 $[v, e, u]$를 의미한다. v, e, u에 대해 각각 임의의 노드와 에지를 매칭한다는 의미로 변수 $\$x$, $\$y$, $\$z$를 사용할 수 있다.

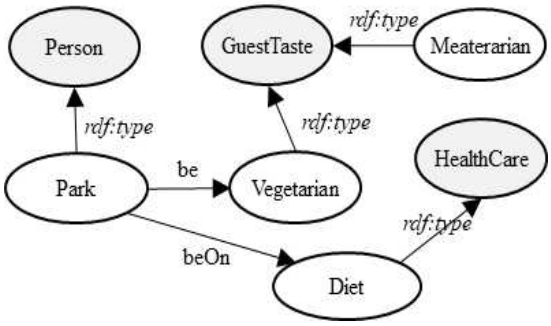


그림 2. RDF 데이터의 그래프 표현
Fig. 2. Graph representation of RDF data

SRS에서의 사용자 정의 지식은 어떤 RDF 온톨로지 데이터에 대하여 다음과 같이 RIF 규칙으로 표현될 수 있다: [$x, be, Vegetarian] ∧ [$x, beOn, Diet] → [$x, like, Salads]; [$x, like, chickenbreastsalad]. 이것은 $x가 채식주의자이며 다이어트 중이면 샐러드를 좋아하고 닭가슴살 샐러드를 좋아한다는 팩트(Fact)를 추론하는 RIF 규칙으로, 그림 2에서 [Park, like, Salads]와 [Park, like, chickenbreastsalad]의 새로운 팩트를 추론할 수 있다.

▪ 정의 2 (RIF 규칙에서의 조건과 추론 결과)

RIF 규칙은 왼편(LHS, Left-Hand Side)에 여러 가지 조건에 해당하는 RDF 트리플 패턴으로 구성되며, 오른편(RHS, Right-Hand Side)에는 추론 결과에 해당하는 트리플이 하나 혹은 여러 개 올 수 있다. 각 조건은 부정 연산(¬)이 적용될 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 rete 알고리즘을 간략히 소개한다. 3장에서는 과거 rete 알고리즘에 대한 병렬 및 분산 처리를 소개하며, 4장부터 6장까지는 최근의 Hadoop, Spark와 같은 빅데이터 기술을 이용한 방법을 소개한다. 7장에서는 이러한 분석을 통해 rete 알고리즘의 효율적 병렬 및 분산 처리를 위해 고려해야 할 점들을 정리한다. 8장에서는 결론 및 향후 연구 내용을 기술한다.

II. Rete 알고리즘

Rete 알고리즘은 생성 시스템에서 다수의 객체에 대한 효율적 패턴 매칭을 위하여 Forgy[1]에 의해

제안된 알고리즘이며, 현재 Drools, Clips, Jess와 같은 규칙 기반(Rule-based) 엔진에서 널리 사용되어진다. 하지만 단일 컴퓨터에서 동작하는 rete 알고리즘이 처리하는 규칙 수가 100,000개가 될 경우 그 성능이 수 십 분으로 매우 안 좋아지며[4], 분명한 한계가 존재한다.

참고문헌 [13]에서 설명한 rete 알고리즘의 일반적 시간 복잡도는 $O(n \cdot m^k)$ 이며, 규칙의 수 n에 선형적(Linear)이며, 매칭되는 팩트의 수 m에 따른 다항식(Polynomial)을 갖는다. k는 모든 규칙 중 어떤 규칙이 갖는 LHS의 최대 조건수이다. 따라서 규칙의 수가 100,000개 보다 더 많아질 경우, 그리고 팩트 수의 증가에 따라 패턴 매칭의 시간이 더 커지는 것은 당연하며, 하지만 기하급수(Exponential)가 아니며 또한 SRS에서의 사용자 프로파일과 같은 팩트 수가 크지 않을 경우 병렬 및 분산 처리에 의한 성능 개선의 가능성이 높다.

그림 3은 rete 네트워크의 구성 및 기본 동작 원리를 보여준다. 그림 3은 참고문헌 [14]에서의 그림을 본 논문의 내용에 맞도록 수정한 것이다.

· Rete 네트워크: 먼저 주어진 규칙들에 대하여 그림과 같은 rete 네트워크를 메모리에 구성한다.

· 작업 기억 메모리: 그림 3에 는 표현되지 않았지만 추론 과정 중 입력된 팩트들을 잠시 기억하는 메모리 장소이다. 따라서 단기 기억 메모리라고도 한다. 이후 이를 WM으로 표기하도록 하겠다.

· 알파 메모리: WM의 팩트들은 패턴 매칭을 통하여 해당 알파 메모리로 저장되고 WM으로 부터는 삭제된다. 그림에서는 각 RDF 트리플 패턴에 해당하는 알파 메모리가 존재함을 볼 수 있다. 예로, 'AM for E1'은 RDF 트리플 패턴 [$x, hasDisease, renaldisease]에 매칭되는 팩트들을 저장하며, WM으로부터 [spiderman, hasDisease, renaldisease], [batman, hasDisease, renaldisease], [superman, hasDisease, reanaldisease], [ironman, hasDisease, reanaldisease]의 팩트들이 매칭되어 저장된 것이다. AM은 알파 메모리를 상징한다. 이후에도 새로운 팩트들이 WM으로 입력될 경우 패턴 매칭에 의하여 해당 패턴의 AM으로 저장되게 되고 WM으로부터는 삭제된다.

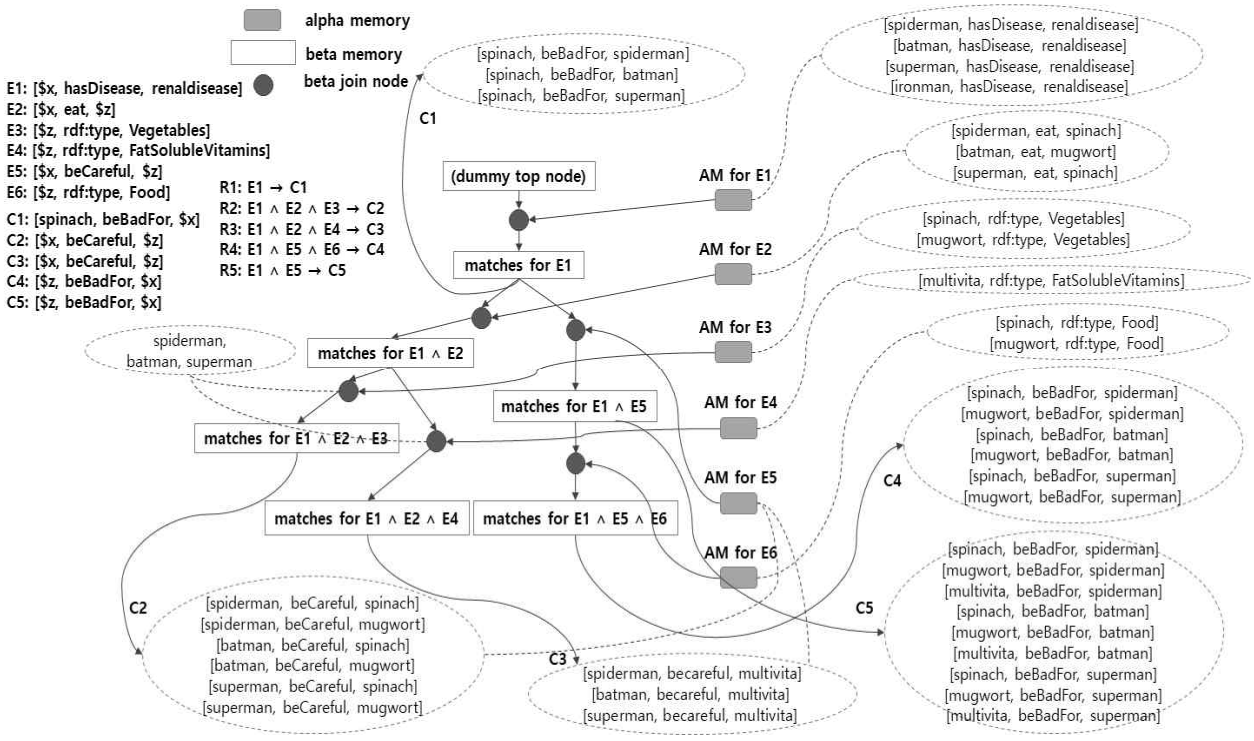


그림 3. Rete 네트워크의 예
Fig. 3. Example of rete network

· 베타 메모리: 이것은 AND(^) 연산에 대한 결과를 저장한다. 예로, 'matches for E1 ^ E2'는 [$x, hasDisease, renaldisease] ^ [$x, eat, $z]의 매칭 결과를 저장한다. 따라서 ironman을 제외한 {[spiderman, hasDisease, renaldisease], [spiderman, eat, spinach]}, {[batman, hasDisease, renaldisease], [batman, eat, mugwort]}, {[superman, hasDisease, renaldisease], [superman, eat, spinach]}의 결과를 저장한다. 이것은 관계형 데이터 모델의 조인(Join) 연산과 같은 개념이다. 이후에는 베타 메모리를 BM으로 표기하도록 하겠다.

· 결과 노드(Conclusion node): C1, C2, C3, C4, C5의 규칙의 결과 노드들은 최종 매칭 결과로 저장되고, 또한 WM으로도 입력되어 반복적인 추론을 발생시킨다. 반복적인 추론의 예로, C2와 C3의 결과는 WM으로부터 E5에 매칭되어 'AM for E5'에 저장되고, E5를 포함하는 규칙 R4와 R5를 추론한다. rete 알고리즘은 더 이상의 추론이 발생되지 않을 때까지 위의 과정을 반복한다.

· 충돌 해결(Conflict resolution): 규칙 기반의 전문가 시스템에서는 어떤 입력이 주어질 경우 결과 노드를 통한 결과는 하나이어야 한다. 예로, 어떤 증상으로 인한 질병 진단 결과는 하나이어야 한다. 따라서 매칭된 여러 규칙 중 어떤 규칙이 실행(Rule firing)되어야 하는 지를 결정하는 것을 충돌 해결이라고 한다. 따라서 그림 3의 첫 번째 사이클(Cycle)에서 어떤 입력에 의해 규칙 R1, R2, R3가 매칭되었을 경우, 그리고 충돌 해결에 의해 R2가 결정되었을 경우, 추론 결과 C2는 다음 사이클의 'AM for E5'에 저장되게 된다.

▪ 관찰 2-1

이러한 rete 매칭이 빠른 이유는 1) AM과 BM을 계속 유지하므로 재계산을 피한다는 것이다. WM에 새로운 팩트가 입력될 경우 매칭되는 해당 AM 및 BM만 재계산된다. 2) 규칙들 사이에서의 같은 조건에 대하여 하나의 AM 혹은 BM만을 형성한다.

이러한 단일 컴퓨터에서의 rete 알고리즘의 성능을 개선하기 위한 과거의 연구들이 있으며, 예로 데이터베이스의 조인 순서 최적화 등을 고려한 TREAT[15], 해싱 인덱싱 등을 활용하는 rete/UL[14], 해싱 및 노드 인덱싱을 활용하는 T. Dong외 3인의 방법[16], 그리고 RETE*[17], D. Xiao외 3인의 방법[18] 등 많은 연구가 있다. 하지만 여전히 대량의 규칙과 팩트가 입력될 경우 단일 컴퓨터에서의 순차 처리(Sequential processing)는 성능 스케일 업에 한계가 있다.

따라서 rete 알고리즘을 다수의 프로세서 및 컴퓨터에서 수행하고자 하는 병렬 및 분산 처리에 대한 몇몇의 연구들이 이루어 졌으며, 다음 장들에서는 이러한 연구들을 소개한다.

III. 과거의 병렬 및 분산 처리 연구

이미 과거에 병렬 및 분산 처리를 통한 생성 시스템의 성능 향상을 위한 몇몇 연구들이 수행되었다[19]. 비록 이러한 방법들이 Hadoop과 같이 모두에게 널리 알려진 일반적인 프레임워크에 기반하지 않고 각자의 특징적인 하드웨어 및 프레임워크를 갖지만, Hadoop 기반의 연구에서도 공통적으로 나타나는 rete 알고리즘을 병렬로 처리할 경우의 다양한 문제들을 이미 소개하고 있다.

당연히 과거에는 값싸지만 좋은 성능을 갖는 컴퓨터 수천 대를 빠른 네트워크 속도로 연결하는 하드웨어의 발전이 없었으므로, 각자의 특징적 하드웨어에 기반할 수밖에 없었을 것이고, 현재의 Hadoop과 같은 일반적 방법의 출현이 없었을 것이다.

각자의 특징적 하드웨어 구조는 주로 공유 메모리 멀티프로세서(Shared memory multiprocessor) 혹은 메시지 패싱(Message passing) 멀티프로세서 방식에 기반한다.

Hadoop 기반의 방법과 비교하여 Hadoop 또한 기본적으로 여러 컴퓨터 사이의 메시지 패싱이지만, 가장 큰 장점은 Hadoop 프레임워크를 이용하는 경우 과거에는 실험되지 못했던 수백 수천대의 컴퓨터로의 확장 시 신뢰성(Credibility)을 자동으로 제공받는 이점이 있다는 것이다. 하지만 Hadoop의 경우 맵 리듀스 기반의 구현을 따라야 하므로, 성능상의 제약사항이 존재할 수 있다. 우리는 이러한 것을 4장과 5장 Hadoop, Spark를 활용하는 방법에서 살펴본다.

3.1 Gupta외 4인의 방법

Gupta외 4인[20]은 16개의 CPU와 32 MByte의 공유 메모리를 가진 Encore Multimax 컴퓨터에서의 rete 알고리즘의 병렬처리를 구현하였다. 실험에 사용된 규칙의 수는 최대 637개로 우리의 가정과 비교하면 작다. 성능 개선은 그들의 실험에서 rete 알고리즘의 AM, BM 매칭 단계에서 2~11배까지의 성능 개선이 이루어짐을 보여준다. 병렬 처리의 단위는 그림 3에서 규칙 단위로 각 프로세서로 할당될 수 있으며, 예로 R1이 하나의 프로세서, R2와 R3가 하나의 프로세서, R4와 R5가 하나의 프로세서로 할당, 혹은 더 세밀한 병렬 처리(More fine-grained parallel processing)로 AM 매칭과 BM 조인 노드 단위로 분할될 수 있다[21]. 노드 단위의 분할은 Hadoop에서 소개할 그림 4의 Li외 4인의 방법과 유사하다.

▪ 관찰 3-1

Rete 알고리즘의 적절한 병렬 처리 단위가 고려될 필요가 있다.

참고문헌 [21]에서는 또한 RHS의 실행에 있어, 모든 매칭된 RHS가 구해질 때까지 충돌해결이 완료될 수 없으므로 궁극적으로 이것은 매칭은 병렬적이나 최종적으로 모든 규칙의 매칭이 완료될 때까지 기다릴 수밖에 없는 동기적(Synchronous) 순차(Sequential) 실행임을 지적하고 있다.

이러한 문제는 Hadoop을 이용하는 방법에서도 나타나며, rete 알고리즘 전체 시간의 90%는 LHS 매칭 단계에서 소모되고[22], 비록 RHS의 실행은 10%이나, 그 10%를 1%로 줄일 수 있음을 가정할 경우, 이러한 비동기적 특성은 이론적으로 최대 10배까지도 전체 시간을 지연시키는 것이다.

▪ 관찰 3-2

만약 충돌 해결을 고려하지 않을 경우 완전한 비동기적 병렬 처리가 고려될 필요가 있다. 여기서 완전함이란 전체의 실행을 의미한다.

3.2 Acharya, Gupta, Tambe의 방법

Acharya와 Tambe[23], Gupta와 Tambe[24]의 방법에서는 수천 수만 개의 프로세서를 가진 컴퓨터에서의 메시지 패싱 멀티프로세서에 의한 rete 알고리즘의 병렬 처리 방법을 고려한다. 각 프로세서는 수십 혹은 수백 KByte의 메모리를 가진다.

그림 3의 AM 매칭과 BM 조인 노드는 각 프로세서에 할당되어 매칭 작업을 수행하며, 매칭될 데이터는 제어 프로세스에 의해 해당 프로세서에 브로드캐스트(Broadcast)된다. 제어 프로세서는 후의 그림 4의 Hadoop에서 살펴볼 Master와 같은 역할을 하며 모든 매칭 프로세서들의 동작을 제어하며, 매칭 프로세서들로부터 한 사이클의 매칭이 완료되었다는 신호를 모두 받았을 때 다음 사이클을 시작한다. 당연히 다음 사이클이 시작하기 전에 RHS가 실행되어 추론된 결과는 제어 프로세스로 보내지며, 이러한 새로이 추론된 결과를 제어 프로세스는 매칭 프로세스들에게 브로드캐스트한다. 시뮬레이션을 통한 성능 실험에서는 25개의 프로세스를 병렬로 사용함에 있어 8 ~ 12배까지의 성능 개선이 이루어짐을 보여준다.

하지만 후의 Hadoop에서도 고려되는 각 프로세서에서의 고르지 않은(Uneven) 매칭 데이터의 분포는 오히려 프로세서가 늘어날수록 실제 성능이 떨어지는 원인이 됨을 보여준다.

3.3 그 외의 방법

Bein과 King[25]은 LAN에 연결된 다수의 컴퓨터에서의 데이터의 수평 분할(Horizontal partitioning)에 의한 병렬 및 분산 처리를 고려한다. 모든 규칙은 모든 컴퓨터에 동일하게 복사된다. 데이터는 기본 키(Primary key) 값에 따라 로드 밸런스를 고려하여 각 컴퓨터로 수평 분할된다. 다음으로 각 컴퓨터에서 추론된 결과는 하나의 제어 컴퓨터로 보내지고, 모든 컴퓨터로부터의 추론 결과가 모아져야지 한 사이클이 끝난다. 즉 동기화된다. 추론 결과가 없는 컴퓨터에 대해서는 일정 시간을 정하여 그 시간이 경과한 경우 추론 결과가 없는 것으로 간주한다. 모아진 추론 결과는 데이터의 분산 기준에 따라 제어 컴퓨터로부터 해당 컴퓨터로 보내지고 다음 사이클의 추론을 수행한다.

Stolfo[26]는 메시지 패싱 멀티프로세서 방식에서의 보다 빠른 메시지 전송을 위하여 이진 트리(Binary tree)의 형태로 연결된 다수의 프로세서를 갖는 특별한 DADO 머신을 고려하였다. DADO2의 경우 1023개의 8bit 프로세서를 가지며, 각 프로세서는 각각 16 KByte의 메모리를 갖는다. rete 알고리즘의 병렬 처리는 앞서 다른 연구에서도 살펴본 바와 같이 규칙 단위로 각 프로세서에 분배될 수 있으며, 또한 노드 수준에서도 분배될 수 있다.

Kelly와 Seviora[27]는 메시지 패싱 멀티프로세서 기반의 CUPID라는 특별한 머신 구조를 고려하였으며, 성능은 앞서 Gupta외 4인[20]의 방법과 유사함을 보였다.

지금까지 살펴 본 바와 같이 과거의 병렬 및 분산 처리는 지금과 같은 하드웨어의 발전이 이루진 상황이 아니었으므로, 단일 머신에서의 병렬 프로세싱에 초점이 맞추어져 있었으며, 현재는 하드웨어 및 네트워크의 발전과 함께 고성능의 값싼 컴퓨터 수백 수천 대를 안전하게 스케일 아웃하는 Hadoop과 같은 병렬 및 분산 처리를 고려하고 있다.

Ⅳ. Hadoop 기반의 연구

Hadoop 프레임워크는 키 값에 따라 작업을 나누어 처리하는 맵(Map)과 이렇게 나누어 처리된 중간 결과를 다시 키 값에 따라 조합하는 리듀스(Reduce) 과정으로 구성되며, 맵의 결과는 HDFS(HaDoop File System)에 기록된다.

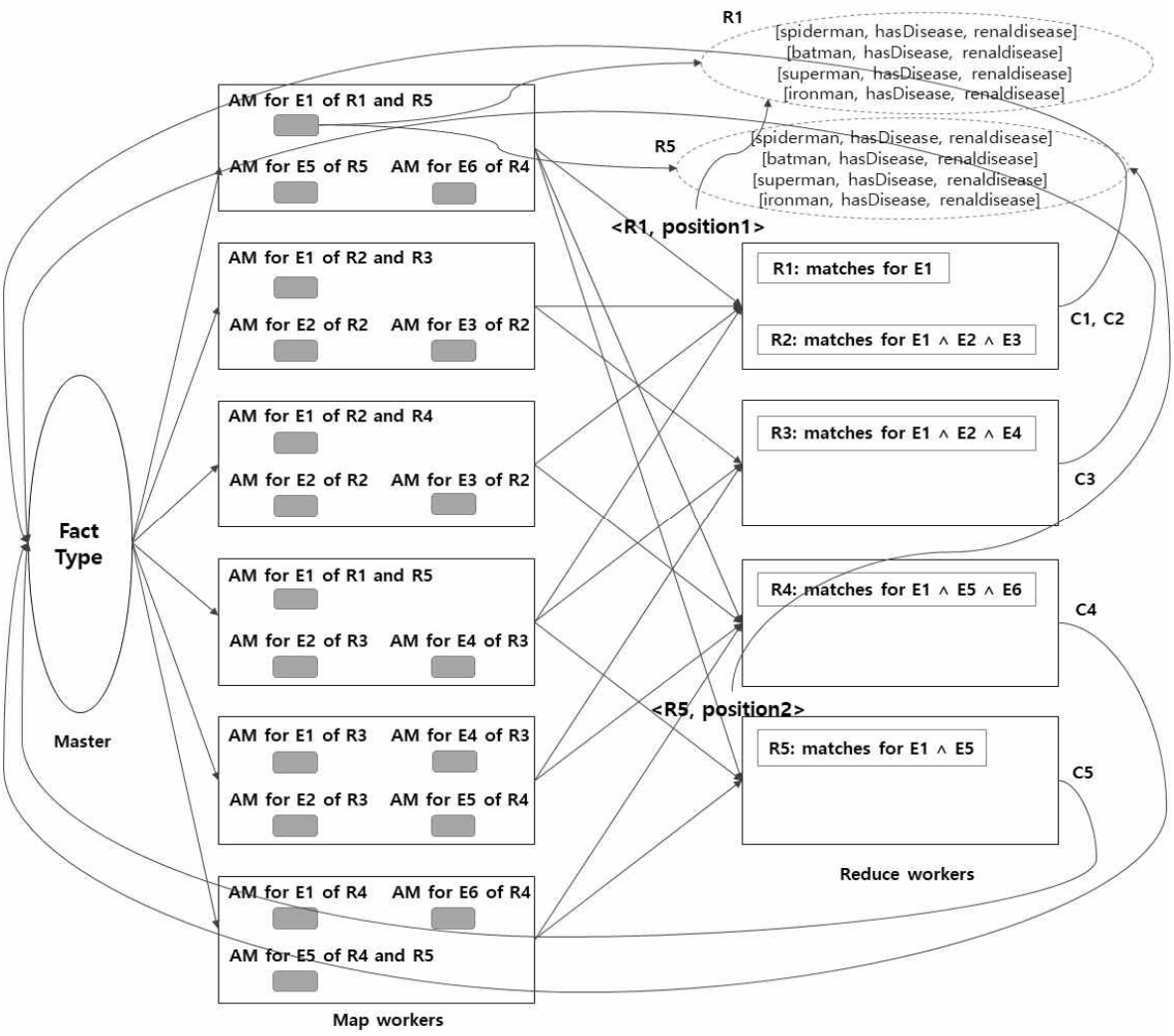


그림 4. Hadoop 기반의 rete 알고리즘의 예
Fig. 4. Example of Hadoop based rete algorithm

4.1 Li외 4인의 방법

먼저 Li외 4인의 방법[7]에서는 그림 3에서의 AM으로의 패턴 매칭을 각 맵 작업 노드로 분산시킨다. 노드로의 분산은 각 노드에서의 작업 오버로드와 패턴 매칭의 복잡도를 예측하여 분산한다. 그림 4는 Li의 맵리듀스 기반의 rete 알고리즘 동작 원리를 보여준다.

· 맵: 그림 3에서의 각 AM과 이에 해당하는 조건의 패턴 매칭이 각 맵 작업 노드로 분산된다. 이때 각 조건은 신뢰성을 위하여 중복되어 분산되며, 그림에서는 두 개씩 중복된 것을 표현하였다. 특별히 'AM for E1 of R1 and R5'와 같이 할당된 어떤 규칙의 조건이 동일할 경우 관찰2-1에서처럼 하나의 패턴으로 합쳐지며, 단 매칭 결과는 그림 4의 R1과 R5와 같이 두 개로 만들어 진다. 하지만 이러한 과정을 Li의 논문에서는 언급하고 있지 않으며, 논문의 내용을 토대로 효율성을 고려하여 유추한 것이다.

각 AM의 메모리상의 매칭 결과는 HDFS에 기록된다. 즉, 그림에서의 R1과 R5는 디스크에 기록된다. 이러한 기록 또한 HDFS 특성상 설정 된 값에 따라 몇 개의 노드에 중복되어 기록될 것이다. 이 과정은 시간이 걸리는 부분이지만 Hadoop 프레임워크에서 어떤 노드의 붕괴 시 다른 중복된 데이터로부터의 복구를 위하여 필요한 과정이다.

▪ 관찰 4-1

Hadoop에서 어떤 노드가 붕괴할 경우 해당 노드의 복구를 위하여 AM의 결과를 디스크에 중복하여 기록할 것이다. 이것은 더 많은 수행 시간을 요구할 것이다.

· 마스터(Master): 마스터는 팩트 타입에 따라 해당 팩트를 해당 맵 워커에 전달한다. 따라서 마스터는 팩트 타입과 해당 맵 워커에 대한 정보를 가지고 있으며, 이러한 비교를 빠르게 처리하기 위하여 인덱스를 사용할 수 있다. 하지만 어떠한 방식의 인덱스를 사용하는지는 논문에서 언급하고 있지 않다. 팩트 타입에 대해서도 구체적으로 언급하고 있지 않지만, 유추한다면 팩트 타입의 수는 AM의 수보다는 작을 것이므로, 예로 RDF 트리플의 subject, predicate, object중 predicate의 값만 가지고 팩트를 분배하는 것을 고려할 수 있다. 따라서 그림 4에서 predicate의 값이 eat일 경우 E2를 포함하는 두 번째부터 다섯 번째 맵 워커로 해당 팩트를 전송하며, predicate의 값이 rdf:type일 경우 모든 맵 워커에 전송한다.

참고문헌 [14][16]에서와 유사하게 마스터에서의 패턴 매칭 시 predicate에 대한 해시 인덱스를 사용하여 보다 빠른 매칭을 지원할 수 있다. 또한 각 맵 워커에서는 [subject, predicate, object]의 전체 패턴 문자열에 대한 해시 인덱스를 사용하여 빠른 매칭을 지원할 수 있다.

이러한 경우 AM으로의 패턴 매칭의 시간 복잡도는 하나의 팩트가 마스터를 거쳐 해당 맵 워커의 AM에 도착하는 시간은 $O(1)$이므로 팩트의 수가 n개일 경우 $O(n)$이나, 맵 워커의 수가 m개일 경우 m개가 병렬로 수행되는 것이므로 $O(n)$과 $O(n/m)$의 사이일 것이다. 하지만 마스터에 병목(Bottleneck) 현상이 있을 경우 $O(n/m)$보다는 많이 느려질 것이다.

▪ 관찰 4-2

팩트의 수를 n, 맵 워커의 수를 m이라고 할 경우, 마스터 노드에서의 병목 현상에 의해 실제 맵 단계에서의 매칭은 $O(n/m)$보다는 많이 느려질 것이다.

· 리듀스: 관찰 4-1에서 각 AM의 매칭된 결과를 디스크에 기록한다고 하였다. 이후 각 맵은 그림 4에서와 같이 <키, 값>의 쌍, 즉 <R1, position1>을 해당 리듀스 워커에 전달한다. 여기서 position1은 해당 매칭 결과를 저장하고 있는 디스크 주소를 의미한다. 각 리듀스는 해당 규칙의 모든 조건에 해당하는 매칭 결과가 전달되었을 경우, 조인 연산을 통해 결과 노드를 계산할 수 있다. 예로, 첫 번째 리듀스 워커에서 R2의 조건, E1, E2, E3의 결과가 모두 전송될 경우 결과노드 C2를 계산한다. 이러한 결과 노드의 팩트들은 다시 마스터로 전달되어 반복적인 추론을 수행한다.

이러한 리듀스 과정에서 의문이 되는 것은 첫 째로, 저자는 리듀스가 우리가 해석한 것과는 달리 <키, 값>의 쌍을 단순히 마스터로 전달한다고 설명

하는 것이다. 즉, 각 맵 워커로부터 받은 해당 규칙의 조건들의 매칭 결과를 조인하는 것을 설명하고 있지 않다. 이러한 조인이 맵 워커에서 일어난다고 설명하고 있으나, 그들이 정의한 맵 워커의 역할은 그림 4에서와 같이 AM을 위한 패턴 매칭만 해당한다.

두 번째로 리듀스로 모든 조건에 해당하는 <키, 값>의 시그널이 올 경우 매번 디스크로부터 매칭 결과를 읽게 되므로 더 많은 수행 시간을 요구하게 된다. 앞서 설명한 관찰4-1, 관찰4-2, 그리고 반복적인 추론 사이클을 고려할 경우 제안된 Hadoop 기반의 방법이 성능 향상에 있어 큰 효과가 있을지는 의문이다. 하지만 저자는 그들의 실험 결과에서 6개의 가상 머신을 사용할 경우 단일 컴퓨터에서의 실행과 비교하여 원 추론 시간의 30%로 축소됨을 보이고 있다.

4.2 Maeda외 2인의 방법

Maeda외 2인의 방법[8]에서는 규칙은 적고, 매칭될 팩트의 수가 많은 경우를 고려한다. 또한 팩트가 대량이기 때문에 이러한 팩트들이 HDFS에 분산되어 저장되는 것을 고려한다. 기본 원리는 앞서 Li의 방법을 따르되, 규칙이 적기 때문에 모든 규칙의 조건들이 하나의 맵 워커 안에 구축된다. 그리고 똑같은 맵 워커를 여러 대 생성한다. HDFS에 저장된 팩트들은 각 맵 워커로 전송되어 동시에 처리되고, Li의 방법대로 리듀스된다.

4.3 Urbani외 4인의 방법

Urbani외 4인의 방법 [9][10]은 Li의 방법과 유사하다. 다만 Urbani의 방법은 RDFS와 OWL 추론 규칙만을 다룬다. 중요한 것은 RDFS와 OWL 추론 규칙이 정해져 있으므로, 중복된 추론 결과 및 셔플링(Shuffling)을 줄이거나, 최적화된 실행을 위하여 규칙의 유형 및 특성을 고려하여, 팩트를 각 노드로 분배하는 것, 그리고 규칙의 실행 순서를 고려하는 것 등을 이용한다. 셔플링은 네트워크를 통해 노드 사이에 오고 가는 데이터를 말하며, 이러한 양이 많을수록 분산 처리의 성능은 낮아질 것이다.

▪ 관찰 4-3

효율적 분산 처리를 위하여 셔플링되는 데이터의 양을 최소화하는 것이 필요하다.

4.4 Zhu외 2인의 방법

Zhu외 2인의 방법[11]은 Li의 방법에 근거하나 주된 내용은 각 매칭 패턴을 맵 워커 노드에 분배할 경우 관련성(Relevance)이 높은 것들을 가능한 같은 노드에 배치하는 것이다. 이는 관찰2-1에서 언급한 rete 매칭이 빠른 이유인 AM과 BM을 계속 유지하므로 재계산을 피하거나, 규칙들 사이에서의 같은 조건에 대하여 하나의 AM 혹은 BM만을 형성하도록 하여 계산 량을 줄일 수 있다. 또한 셔플링을 줄일 수 있다. 이를 위해 Zhu et al.은 모든 패턴 사이에서의 X → Y의 연관성 분석을 위해 apriori 알고리즘을 사용한다. 예로, 그림 3에서 {E1}, {E1, E2, E3}, {E1, E2, E4}, {E1, E5, E6}, {E1, E5}의 4개의 행(Row)이 테이블로 입력된 후, apriori 알고리즘이 적용되어 X → Y 연관성 분석을 수행한다.

하지만 연관성이 높은 것들이 몇 개의 노드에만 집중될 경우 오히려 고르지 못한 분포로 병렬 수행의 성능이 떨어질 수 있음을 고려하지 않고 있다.

Li외 4인의 방법[7]에서도 패턴의 각 맵 워커 노드로의 균등한 분배를 위하여 패턴의 복잡도(Complexity)와 각 노드의 작업 오버로드를 고려한다. 패턴의 표현이 보다 복잡한 것은 노드의 자원을 더 많이 사용한다는 가정 하에 이러한 패턴들을 균등하게 분배하려고 하는 것이다. 또한 만약 두 노드의 분배된 패턴의 복잡도가 총괄적으로 동등한 상태라면, 두 번째 조건으로 노드의 작업 오버로드, 즉 현재의 CPU, 메모리, 네트워크 자원 사용량이 더 적은 쪽에 분배한다.

▪ 관찰 4-4

3.2절에서도 살펴본 바와 같이 규칙 및 데이터의 노드로의 고른 분포는 분산 처리에 있어 중요한 요소이다.

마지막으로, Zhu의 방법에서 모든 규칙의 패턴에 대하여 apriori 알고리즘을 수행하여야 하므로 일정량의 규칙이 새로 들어온 후 주기적으로 rete 네트

워크를 다시 구성하여야 한다.

V. Spark 기반의 연구

Hadoop 기반의 rete 분산 알고리즘의 문제점은 앞서 살펴본 바와 같이 맵 단계에서 중간 결과를 HDFS에 기록하여야 하는 것이며, 또한 추론 체인(Reasoning chain)의 경우 반복적인 맵-리듀스 작업이 비효율적이라는 것이다. 이러한 문제를 고려하여 최근에는 인메모리 기반의 분산 처리 프레임워크인 Spark를 활용한 rete 분산 알고리즘의 연구가 있으며, 몇 가지 연구에서는 Spark를 활용하는 것이 Hadoop을 사용하는 것보다 2 ~ 3 배의 개선 효과가 있는 것을 보여준다[12][28].

Spark에서는 메모리에서의 데이터 저장 관리를 위하여 RDD(Resilient Distributed Dataset)라는 객체를 사용할 수 있다. 이러한 RDD는 파티셔닝을 비롯하여 여러 가지 병렬 연산 기능을 제공한다.

5.1 Lee외 2인의 방법

Lee외 2인[12]은 Spark 기반의 rete 알고리즘을 소개하였으며, Wu와 4인의 Hadoop 기반의 방법[29]과 비교하여 최대 2.7배의 개선이 있음을 보여주고 있다. Lee의 방법에서는 먼저 주어진 규칙들에 대해 메타 테이블(Meta table)을 만드는 사전 준비 작업이 요구된다. 이러한 메타 테이블로는 Key-Value 테이블, Common Variable 테이블, Conclusion 테이블의 세 가지 테이블이 작성된다.

먼저 그림 5(a)는 그림 3에서의 규칙들에 대하여 만들어진 Key-Value 테이블이다. 각 조건의 predicate에 대하여, 공통 변수의 위치 정보를 기록한다. 예로, 규칙 R2의 조건 E1의 predicate은 hasDisease이며, subject 위치에 있는 $x는 E2의 $x와 공통 변수이다. R1의 경우 $x가 LHS의 조건 중에 공통 변수가 없으므로 기록되지 않는다.

그림 5(b)는 Common Variable 테이블을 보여준다. Common Variable 테이블은 반복적인 맵 리듀스 과정에서 어떤 규칙의 다음 단계의 조건 평가를 위한 공통 변수의 위치 정보를 표현한다. 예로, R2의 경우 두 번째 단계의 공통 변수 매칭은 E2 조건의 subject의 위치에 있는 $x 변수이다. 세 번째 단계는 E2 조건의 object의 위치에 있는 $z 변수이고, 네 번째 단계는 E3 조건의 subject 위치에 있는 $z 변수이다.

그림 5(c)는 Conclusion 테이블을 보여준다. Conclusion 테이블은 추론 결과의 새로운 트리플을 생성하기 위해 사용되어 진다. 예로, R2는 E2의 $x를 subject, beCareful을 predicate, E3의 $z를 object로 하는 새로운 트리플을 추론한다.

n개의 주어진 규칙으로부터 이러한 Key-Value 테이블, Common Variable 테이블, Conclusion 테이블의 구축은 모든 규칙에서의 LHS의 조건의 수가 그렇게 크지 않다고 가정할 경우 시간 복잡도는 $O(n)$으로 그렇게 많은 비용을 요구하지 않는다.

term	(rule ID, common variable)
hasDisease	(R2, subject), (R3, subject), (R4, subject), (R5, subject)
eat	(R2, subject), (R2, object), (R3, subject), (R3, object)
rdf:type	(R2, subject), (R3, subject), (R4, subject)
beCareful	(R4, subject), (R4, object), (R5, subject)

(a) Key-value table

phase	rule ID	(term, index)
2	R2	(eat, subject)
3	R2	(eat, object)
4	R2	(rdf:type, subject)
2	R3	(eat, subject)
3	R3	(eat, object)
4	R3	(rdf:type, subject)
2	R4	(beCareful, subject)
3	R4	(beCareful, object)
4	R4	(rdf:type, subject)
2	R5	(beCareful, subject)

(b) Common variable table

rule ID	subject	predicate	object
R1	spinach	beBadFor	(hasDisease, subject)
R2	(eat, subject)	beCareful	(rdf:ype, subject)
R3	(eat, subject)	beCareful	(rdf:type, subject)
R4	(rdf:type, subject)	beBadFor	(hasDisease, subject)
R5	(beCareful, object)	beBadFor	(beCareful, subject)

(c) Conclusion table

그림 5. 메타 테이블 구성

Fig. 5. Configuring meta tables

HDFS에 저장되는 사용자 프로파일은 전처리 과정을 통하여 key-value 형태로 변환되어 저장된다. 이를 위해 Key-Value 테이블을 참조한다. 예로, 트리플 [spiderman, hasDisease, reanldisease]의 경우 그림 5(a)의 Key-Value 테이블에서 term hasDisease에 대하여 <(R2, spiderman), {[spiderman, hasDisease, renaldisease]}>, <(R3, spiderman), {[spiderman, hasDisease, renaldisease]}>, <(R4, spiderman), {[spiderman, hasDisease, renaldisease]}>, <(R5, spiderman), {[spiderman, hasDisease, renaldisease]}>의 4개의 변환된 데이터로 저장된다. 하지만 공통 변수가 없는 조건이 하나인 R1 규칙에 대하여서도 <(R1, spiderman), {[spiderman, hasDisease, renaldisease]}>가 만들어 져야 하며, 그러나 논문에서는 이러한 부분에 대한 설명이 없다. 다음으로 [spiderman, eat, spinach]에 대하여, term eat에 대하여 <(R2, spiderman), {[spiderman, eat, spinach]}>, <(R2, spinach), {[spiderman, eat, spinach]}>, <(R3, spiderman), {[spiderman, eat, spinach]}>, <(R3, spinach), {[spiderman, eat, spinach]}>의 4개의 변환된 데이터로 저장된다.

중요한 것은 이러한 변환된 데이터들은 키 값을 기준으로 Spark의 Hash Partitioner를 통해 클러스터의 각 노드에 파티셔닝(Partitioning)된다는 것이다. 따라서 데이터는 각 노드에 고르게 분포되며, 그림 3에서의 베타 조인을 위한 두 트리플은 같은 노드에 존재하게 된다. 예로, <(R2, spiderman), {[spiderman, hasDisease, renaldisease]}>와 <(R2, spiderman), {[spiderman, eat, spinach]}>는 같은 노드에 존재할 것이다. 하지만 Lee는 어떤 규칙의 모든 변환된 데이터가 같은 노드에 존재하여 셔플링이 발생하지 않는다고 하였는데, 이것은 의문이다. 왜냐하면 <(R2, spinach), {[spiderman, eat, spinach]}>의 경우 키 (R2, spinach)는 해시 값이 다를 것이므로 다른 노드에 할당될 것이고, R2의 세 번째 단계를 평가하기 위하여 셔플링은 발생할 수밖에 없다.

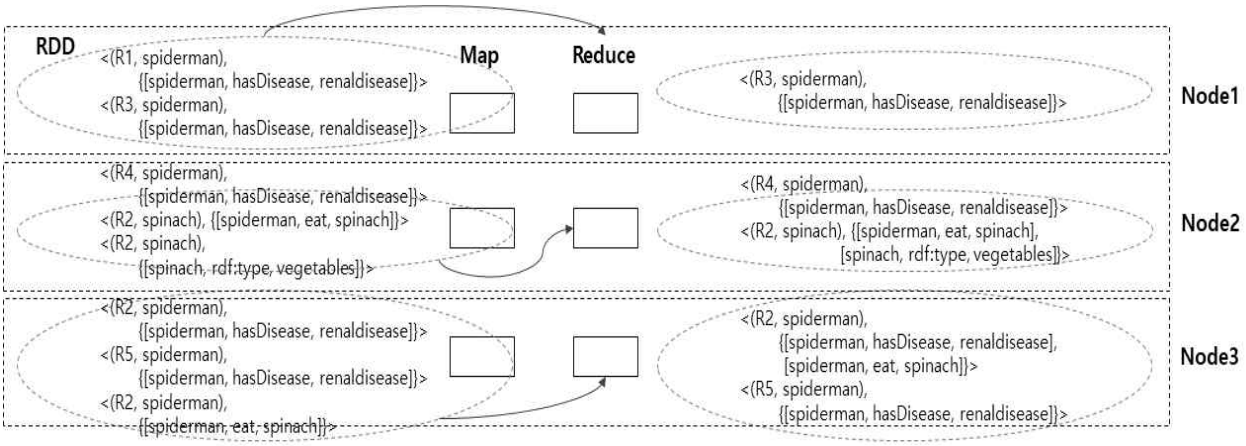
규칙 및 데이터의 전처리 과정 후 진행되는 맵 리듀스 기반의 추론 알고리즘의 동작 원리는 그림 6과 같다.

추론 과정은 리듀스 → 맵 → 리듀스 → ... → 맵 → 리듀스를 반복하며, 그림 5(b)의 Common Variable 테이블에서의 마지막 단계가 끝난 규칙들에 대해서는 그림 5(c)의 Conclusion 테이블에 의하여 결론을 추론한다. 데이터의 전처리 과정이 이미 되어 있으므로 첫 번째 사이클에서는 맵 과정 없이 리듀스 과정을 바로 수행한다.

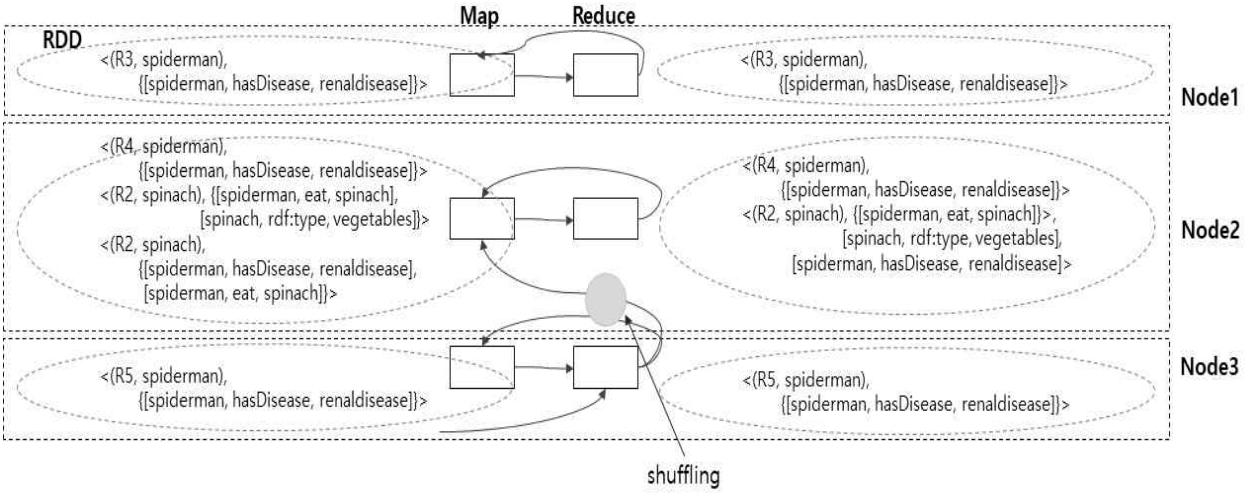
먼저 그림 6(a)의 R1의 경우 전처리 과정에서 Node1의 RDD에 저장된 <(R1, spiderman), {[spiderman, hasDisease, renaldisease]}>에 대하여, 맵 과정 없이, 리듀스 과정을 거친 후, 그림 5(b)의 Common Variable 테이블에서 R1의 2단계가 없으므로 바로 그림 5(C)의 Conclusion 테이블을 참조하여 [spinach, beBadFor, spiderman]의 새로운 트리플을 추론하다.

또한 Node2의 RDD에 저장된 <(R2, spinach), {[spiderman, eat, spinach]}>, <(R2, spinach), {[spinach, rdf:type, vegetables]}>는 reduce 과정에서 키 (R2, spinach)에 의해 <(R2, spinach), {[spiderman, eat, spinach], [spinach, rdf:type, vegetables]}>로 그룹화 된다. Node3에서도 <(R2, spiderman), {[spiderman, hasDisease, renaldisease]}>와 <(R2, spiderman), {[spiderman, eat, spinach]}>는 키 (R2, spiderman)에 의해 <(R2, spiderman), {[spiderman, hasDisease, renaldisease], [spiderman, eat, spinach]}>로 그룹화 된다. 여기서 주목할 것은 각 데이터가 키 값에 의해 같은 노드에 해시 파티셔닝 되었으므로, 맵리듀스의 한 사이클에서 리듀스 과정의 그룹화를 위해 셔플링이 발생하지 않고 각 노드 내에서 처리된다는 것이다.

그림 6(b)의 두 번째 사이클의 맵 단계에서는 첫 번째 사이클의 추론 결과의 키를 Common Variable 테이블을 참조하여 다음 단계의 공통 인자로 치환한다. 예로, <(R2, spiderman), {[spiderman, hasDisease, renaldisease], [spiderman, eat, spinach]}>의 3단계는 (eat, object)이므로, spiderman → spinach에 의해, <(R2, spinach), {[spiderman, hasDisease, renaldisease], [spiderman, eat, spinach]}>가 된다. 이러한 맵 단계에서의 키 변경은 그림 6(b)에서와 같이 Hash Partitioner에 의해 해당 노드로 이동시키는 셔플링이 발생한다.



(a) 첫 번째 리듀스 사이클
(a) First cycle of reduce



(b) 두 번째 맵-리듀스 사이클
(b) Second cycle of map-reduce
그림 6. Spark 기반의 rete 알고리즘의 예
Fig. 6. Example of spark based rete algorithm

VI. GPU 기반의 연구

최근 딥 러닝과 관련하여 GPU 기반의 고성능 병렬 프로그래밍 기술이 활용되고 있다. Peters외 3인 [30]은 단일 컴퓨터에서의 GPU 기반의 rete 알고리즘의 병렬 처리에 대한 연구를 소개하고 있다.

병렬 프로그래밍 방법으로는 OpenCL[31]을 사용한다. OpenCL에서 문자열(String)은 처리에 효율적이지 않으므로, subject, predicate, object에 대한 문자열 값을 정수 값으로 맵핑하고 이러한 정수 값을 참조하여 사용한다.

GPU에서의 각 쓰레드는 하나의 RDF 트리플당 수행되며, 그림 3에서 그 RDF 트리플이 어떤 AM에 매칭 되는지를 병렬로 수행한다. 따라서 RDF 트리플 수만큼의 쓰레드가 수행된다. 또한 베타 조인 노드에 대해서는 그림 7과 같이 두 입력되는 AM에 대하여, 어느 한 쪽의 AM의 팩트에 대하여 다른 한 쪽 AM의 모든 팩트들을 매칭하는 쓰레드가 실행된다. 따라서 왼쪽의 AM의 팩트의 수가 4개이므로 4개의 쓰레드가 수행된다.

따라서 그림 3의 전체 rete 네트워크상에서의 병렬 수행은 먼저 WM의 각 RDF 트리플의 AM 매칭이 병렬적으로 수행이 되고, 모든 RDF 트리플의 AM 매칭이 끝났을 경우 앞서 설명한 베타 조인 노드들에서의 병렬 수행이 이루어진다.

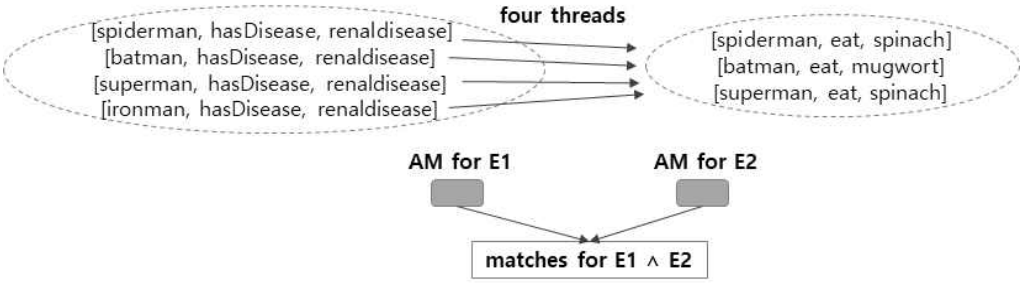


그림 7. GPU를 이용한 병렬 처리의 예
Fig. 7. Example of parallel processing using GPU

표 1. Rete 알고리즘의 병렬 및 분산 처리시 고려사항
Table 1. Considerations in parallel and distributed processing of the rete algorithm

고려사항	비고
· AM과 BM을 계속 유지하는 방법을 고려할 수 있도록 한다.	Rete 알고리즘의 특성
· 규칙들 사이에서의 같은 조건에 대하여 하나의 AM 혹은 BM만을 형성하는 방법을 고려한다.	
· 규칙 단위, 혹은 노드 단위 등의 적절한 병렬 처리 단위가 고려되어야 한다.	과거 병렬 및 분산 처리 및 현재의 Hadoop, Spark 활용에서 함께 고려됨
· 비동기적 병렬 처리 방법을 고려할 수 있도록 한다.	
· 마스터 노드에서의 병목 현상을 고려하도록 한다.	
· AM 혹은 BM 매칭 단계에서의 해싱, 인덱싱 등의 구조를 활용할 수 있도록 한다.	
· 셔플링을 감소시킬 수 있는 방법을 고려한다.	
· 규칙 및 데이터의 분배 시 고른 오버로드가 이루어질 수 있도록 한다.	
· 각 노드에서는 GPU 병렬 처리를 고려할 수 있도록 한다.	GPU 활용

또한 그림 3에서 E1 ∧ E2와 E1 ∧ E2 ∧ E3는 순차적으로 수행되나, E1 ∧ E2 ∧ E3와 E1 ∧ E2 ∧ E4는 병렬로 수행된다. 그들의 실험에서는 비록 GPU가 아닌 CPU에서 코어수를 증가시키며 수행한 실험을 보여주었지만, 1개 코어에 비해 6개 코어를 사용할 경우 5 ~ 6배의 성능 향상이 이루어짐을 보여준다.

Ⅶ. Rete 알고리즘의 병렬 및 분산 처리시의 고려 사항

표 1은 지금까지의 내용을 토대로 rete 알고리즘의 병렬 및 분산 처리 시에 고려해야할 사항들을 정리한 것이다. 앞서 이야기한 대로 이러한 사항들은 과거의 연구에서도 고려된 사항들이지만, 현재는 수백 수천대의 컴퓨터를 안전하게 병렬 및 분산 처리할 수 있는 Hadoop 프레임워크를 활용할 수 있다.

본 논문에서 고려하는 SRS 규칙 추론의 특성은 1) 대량의 규칙 및 온톨로지 데이터를 가능한 신속히 추론해야 하며 2) 연속적으로 유입되는 규칙 및 온톨로지 데이터를 추론할 수 있어야 하는 것이다.

두 번째의 연속적으로 유입되는 규칙 처리라는 것은 기존의 규칙들에 대하여 만들어진 rete 네트워크를 규칙이 유입될 때마다 동적으로 재구성하여 앞으로 유입될 데이터뿐만 아니라 이전 온톨로지 데이터에 대해서도 평가를 수행할 수 있어야 함을 의미한다. 앞서 살펴본 과거 및 기존 연구들은 이러한 특성을 구체적으로 고려하지 않았으며, SRS에서는 이러한 특성을 고려하여 설계할 필요가 있다.

Ⅷ. 결론 및 향후 연구 내용

본 논문에서는 rete 알고리즘의 병렬 및 분산 처리에 대한 과거로부터 현재까지의 연구 내용을 정리 분석하였다. 또한 이러한 분석을 통하여 향후 rete 알고리즘을 개발할 경우의 고려 사항 등을 정리하였다. 이러한 병렬 및 분산 처리를 통한 성능이 개선된 rete 알고리즘은 서론에서 언급한 것처럼 향

후 규칙을 통한 지능적 서비스 및 지식 공유 서비스에 활용될 가능성이 높다.

향후 연구 계획은 본 논문의 분석을 토대로 실제적으로 활용 가능한 rete 알고리즘의 신속하며 효율적인 병렬 및 분산 처리 방법을 개발하는 것이다. 마지막으로 본 논문에서는 rete 알고리즘의 병렬 및 분산 처리에 대한 현재까지의 모든 연구를 가능한 조사하려고 하였으며, 이러한 정리가 관련 연구자들에게 다른 시각에서의 참고자료로 활용될 수 있길 바란다.

References

[1] C. L. Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", Artificial Intelligence, Vol. 19, No. 1, pp. 17-37, Sep. 1982.

[2] J. Choi and Y. Park, "Research Trends in Ontology Reasoning", Communications of KIISE, Vol. 24, No. 12, pp. 47-55, Dec. 2006.

[3] J. Kim, "An Analysis of Uncertainty Problems in Knowledge Sharing Service Using RIF Rules", Journal of KIIT, Vol. 16, No. 9, pp 1-11, Sep. Sep. 2018.

[4] J. Kim and H. Seo, "Shared Rules: Knowledge Sharing Service by RIF Rules", IEEE BigComp 2019, pp. 156-159, Feb. 2019.

[5] RIF Overview, http://www.w3.org/TR/2010/NOTE-rif-overview-20100622/, [accessed: Apr. 16, 2019]

[6] SWRL: A Semantic Web Rule Language Combining OWL and RuleML, http://www.w3.org/Submission/SWRL, [accessed: Apr. 16, 2019]

[7] Y. Li, W. Liu, B. Cao, J. Yin, and M. Yao, "An Efficient MapReduce-based Rule Matching Method for Production System", Future Generation Computer Systems, Vol. 54, pp. 478-489, Jan. 2016.

[8] R. Maeda, N. Ohta, and K. Kuwabara, "MapReduce-based Implementation of a Rule System", Studies in Computational Intelligence 513, pp. 197-206, 2014.

[9] J. Urbani, S. Kotoulas, J. Maassen, F. V. Harmelen, and H. Bal, "WebPIE: A Web-scale Parallel Inference Engine using MapReduce", Journal of Web Semantics, Vol. 10, pp. 59-75, Jan. 2012.

[10] J. Urbani, S. Kotoulas, J. Maassen, F. V. Harmelen, and H. Bal, "OWL Reasoning with WebPIE: Calculating the Closure of 100 Billion Triples", Lecture Notes in Computer Science, Vol. 6088, pp. 213-227, Jan. 2010.

[11] S. Zhu, H. Huang, and L. Zhang, "A Distributed Architecture for Rule Engine to Deal with Big Data", ICACT 2016, pp. 602-606, Jan. 2016.

[12] W. Lee, S. Bae, and Y. Park, "A Study on Distributed Parallel SWRL Inference in an In-Memory-Based Cluster Environment", Journal of KIISE, Vol. 45, No. 3, pp. 224-233, Mar. 2018.

[13] L. Albert, "Average Case Complexity Analysis of RETE Pattern-Match Algorithm and Average Size of Join in Databases", INRIA Research Report, 1989.

[14] R. B. Doorenbos, "Production Matching for Large Learning Systems", Doctoral Dissertation, Carnegie Mellon University, 1995.

[15] P. Nayak, A. Gupta, and P. Rosenblum, "Comparison of the Rete an Treat Production Matchers for Soar (A Summary)", Proc. of AAAI-88, pp. 693-698, 1988.

[16] T. Dong, J. Shi, J. Fan, and L. Zhang, "An Improved Rete Algorithm Based on Double Hash Filter and Node Indexing for Distributed Rule Engine", IEICE trans. on Information and Systems, Vol. E96-D, No. 12, pp. 2635-2644, Dec. 2013.

[17] I. Wright and J. Marshall, "The Execution Kernel of RC++: RETE*, A Faster Rete with Treat as A Special Case", Int. J. of Intelligent Games and Simulation, Vol. 2, No. 1, pp. 36-48, Feb. 2003.

[18] D. Xiao, Y. Tong, H. Yang, and M. Cao, "The Improvement for Rete Algorithm", Proc. of the

2009 First IEEE Int. Conf. on Information Science and Engineering, pp. 5222-5225, Dec. 2009.

[19] S. Kuo and D. Moldovan, "The State of Art in Parallel Production Systems", Journal of Parallel and Distributed Computing, Vol. 15, No. 1, pp. 1-26, May 1992.

[20] A. Gupta, C. L. Forgy, D. Kalp, A. Newell, and M. Tambe, "Parallel OPS5 on the Encore Multimax", Proc. of 1988 Int. Conf. on Parallel Processing, Aug. 1988.

[21] H. G. Okuna and A. Gupta, "Parallel Execution of OPS5 in QLISP", Proc. of The Fourth Conf. on Artificial Intelligence Applications, Mar. 1988.

[22] T. Ishida, "Parallel, Distributed and Multiagent Production Systems", Proc. of the First Int. Conf. on Multi-Agent Systems, pp. 416-422, 1995.

[23] A. Acharya and M. Tambe, "Production Systems on Message Passing Computers: Simulation Results and Analysis", Proc. of 1989 Int. Conf. on Parallel Processing, 1989.

[24] A. Gupta and M. Tambe, "Suitability of Message Passing Computers for Implementing Production Systems", Proc. of AAAI-88, Aug. 1988.

[25] J. Bein and R. King, "MOBY: An Architecture for Distributed Expert Database Systems", Proc. of the 13th VLDB, pp. 13-20, 1987.

[26] S. J. Stolfo, "Five Parallel Algorithms For Production System Execution on the DADO Machine", Proc. of AAAI-84, pp. 300-307, Aug. 1984.

[27] M. A. Kelly and R. E. Seviora, "An Evaluation of DRete on CUPID for OPS5 Matching", Proc. of the 11th Int. Joint Conf. on Artificial intelligence, pp. 84-90, Aug. 1989.

[28] W. Lee, S. Bang, and Y. Park, "Large Scale Incremental Reasoning Using SWRL Rules in a Distributed Framework", Jornal of KIISE, Vol. 44, No. 4, pp. 383-391, Apr. 2017.

[29] H. Wu, J. Liu, D. Ye, J. Wei, and H. Zhong, "Scalable Horn-Like Rule Inference of Semantic Data Using MapReduce", Int. Conf. on Knowledge Science, Engineering and Management, pp. 270-277, 2014.

[30] M. Peters, C. Brink, S. Sachweh, and A. Zündorf, "Rule-based Reasoning on Massively Parallel Hardware", Proc. of the 9th Int. Conf. on Scalable Semantic Web Knowledge Base Systems, pp. 33-48, 2013.

[31] OpenCL, https://www.khronos.org/opencl, [accessed: Apr. 16, 2019]

저자소개

김 재 훈 (Jaehoon Kim)

1997년 2월 : 건국대학교 전자계산학과(공학사)
1999년 2월 : 건국대학교 컴퓨터·정보통신공학과(공학석사)
2005년 2월 : 서강대학교 컴퓨터공학과(공학박사)
2005년 3월 ~ 2006년 9월 : 삼성전자 정보통신총괄 통신연구소 책임연구원
2006년 9월 ~ 2009년 2월 : 서강대학교 컴퓨터공학과 연구교수
2009년 3월 ~ 현재 : 서일대학교 정보통신공학과 부교수
관심분야 : 데이터베이스, 시맨틱 웹, 데이터 프라이버시