데이터로그 시스템들의 재귀 질의 처리 성능 평가

이유경*, 김현지*, 홍기재**, 강혁규**, 한욱신***
*포항공과대학교 창의 IT 융합공학과
**포항공과대학교 컴퓨터공학과
***포항공과대학교 컴퓨터공학과/창의 IT 융합공학과
e-mail : {yklee, hjkim, kjhong, hkkang, wshan}@dblab.postech.ac.kr

Experimental Evaluation of Recursive Query Processing in Datalog Systems

Yukyoung Lee*, Hyeonji Kim*, Ki-Jae Hong**, Hyuk Kyu Kang**, Wook-Shin Han***
*Dept. of Creative IT Engineering, POSTECH
**Dept. of Computer Science and Engineering, POSTECH
***Dept. of Computer Science and Engineering/Dept. of Creative IT Engineering, POSTECH

요 약

데이터로그는 논리형 선언형 프로그래밍 언어로, 특히 재귀적인(recursion) 알고리즘을 표현하기 편리한 언어이다. 대표적인 데이터로그 시스템으로는 CORAL, LogicBlox, XSB, Soufflé 가 있다. 본 논문에서는 이 네 가지 시스템의 특징을 설명하고, 세 가지 벤치마크, 이행적 폐쇄(Transitive closure), 동세대(same generation), 포인터 분석(pointer analysis)으로 데이터로그 시스템들의 재귀 질의(recursive query) 처리 성능을 비교하였다.

1. 서론

데이터로그는 논리형 선언형 프로그래밍 언어로 특히 재귀적인(recursion) 알고리즘 표현하기 편리한 언어이다. 전통적인 명령형(imperative) 언어로 구현했을 때 수천 줄로 표현되는 프로그램이 데이터로그를 사용하면 수십 줄로 표현되는 경우도 존재한다. 이런 편리함 덕분에 프로그래머들은 프로그램을 쉽게 구현할 수 있을 뿐만 아니라, 데이터로그로 구현되어 있는 코드를 이해하기도 쉽다. 데이터로그는 주로 데이터 통합(data integration), 프로그램 분석(program analysis), 정보 추출(information extraction), 네트워크 모니터링(network monitoring), 클라우드 컴퓨팅(cloud computing) 등 다양한 분야에서 활용된다.

데이터로그 프로그램은 알려진 사실로부터 새로운 가능한 모든 사실을 유추하는 방식의 상향식 처리와, 증명해야 할 사실로부터 시작해 증명이 필요한 정리(lemma)들을 유추하는 방식인 하향식 처리 방식으로 나누어진다. 본 논문에서는 두가지 종류에 대해 성능이 좋다고 알려진 데이터로그 시스템들의 성능을 다양한 벤치마크를 사용하여 비교 및 분석하였다.

본 논문은 다음과 같이 구성되어있다. 2 장에서는 데이터로그가 무엇인지 설명한다. 3 장에서는 기존의 데이터로그 시스템들을 설명한다. 4 장에서는 실험 환경과 실험결과를 설명하고 분석한다. 마지막으로 5 장에서는 결론을 설명한다.

이 논문은 2019 년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2017R1A2B3007116)

2. 데이터로그

2.1. 데이터로그 정의

데이터로그 프로그램은 유한한 개수의 사실(fact)들과 규칙(rule)들, 그리고 질의의 집합으로 정의된다. 이들의 정의를 설명하기 위해서는 원자(atom)의 정의가 필요하다. 원자 $p(T)$은 술부(predicate) p 와 튜플(tuple) T 로 구성된다. 술부는 관계(relation)를 의미하며, 튜플은 각 인자가 변수 혹은 상수가 될 수 있다.

사실(fact)은 모든 인자가 상수인 원자로, 특정 관계에 속하는 튜플(tuple)을 의미한다. 규칙(rule)은 머리(head)와 몸통(body)으로 이루어진 명제(proposition)로 식 1 과 같이 표현된다.

$$p_0(T_0) \leftarrow p_1(T_1), p_2(T_2), ..., p_n(T_n) \qquad (식\ 1)$$

머리는 규칙의 좌변을 뜻하며, 하나의 원자(식 1 에서 $p_0(T_0)$)로 구성된다. 몸통은 규칙의 우변을 뜻하며, 쉼표로 구분된 여러 원자(식 1 에서 $p_1(T_1), ..., p_n(T_n)$)로 구성된다. 머리에 존재하는 튜플의 변수 인자는, 몸통에 존재하는 튜플 중 적어도 하나에 포함 되어야한다. 몸통에 존재하는 쉼표는 논리곱(logical conjunction, AND)을 의미하며, 주어진 규칙에서 몸통의 모든 원자가 만족된다면 머리의 원자도 만족된다.

데이터로그 질의 수행은, 주어진 사실과 규칙으로부터 추론할 수 있는 모든 질의 결과를 찾는 것이다.

2.2. 예시

a, *b*, *c* 세 개의 정점과, (*a*,*b*), (*b*,*c*) 두 개의 방향이 있는 간선을 가지는 그래프에서 모든 가능한 경로(path)를 찾는 프로그램을 규칙과 사실, 질의로 표현하면 식 2 와 같다. 이 질의는 대표적인 재귀 질의인 이행적 폐쇄(transitive closure) 질의이다. 이 질의의 결과로 두 정점 사이의 모든 경로들 *path*(*a*,*b*), *path*(*b*,*c*), *path*(*a*,*c*)가 얻어진다.

규칙: $path(X,Y) \leftarrow edge(X,Y).$
$path(X,Y) \leftarrow path(X,Z), edge(Z,Y).$
사실: $edge(a,b), edge(b,c).$ (식 2)
질의: $? path(X,Y).$

3. 기존의 데이터로그 시스템

3.1. CORAL[1]

위스콘신 대학교에서 개발한 데이터로그 엔진으로, 오래되었지만 잘 알려진 시스템이다. 선언형 언어에 C++ 인터페이스까지 지원한다. CORAL 은 상향식과 하향식의 처리 방식을 둘다 지원하며, 불필요한 계산을 줄이기 위한 프로그램 재작성 방법인 매직 셋 변환(magic set transformation, MST)[7]을 사용한 상향식 처리를 기본 설정으로 사용한다.

3.2. LogicBlox[2]

LogicBlox 는 의사결정 자동화, 분석 및 계획 등의 응용을 대상으로 하는 가장 대표적인 상업용 데이터로그 시스템이다. LogicBlox 는 상향식 처리방법을 사용 하며 입력 데이터가 변화함에 따라 질의 결과를 업데이트하는 점진적 관리(incremental maintenance)를 지원한다.

3.3. XSB[3]

스토니브룩 대학교에서 개발한 프로로그(Prolog) 엔진으로, 가장 성능이 좋다고 알려진 프로로그 시스템이다. C 로 작성되어 있으며 오픈소스이다. 하향식 처리 방식을 사용하며 이형 테이블링(variant tabling)[8]과 포함 테이블링(subsumptive tabling)[9]을 사용한다.

3.4. Soufflé[4]

Soufflé 는 가장 최신 데이터로그 시스템으로 상향식 처리 방식을 사용하고 있다. Soufflé 는 데이터로그 프로그램을 템플릿화된 C++ 코드로 바꾸어 이를 컴파일 함으로써 성능을 개선한다. 트라이(trie)와 비트리(B-tree) 자료구조 기반의 색인(index)을 사용하여 색인 중첩 루프 조인(index nested-loop join)과 효율적인 중복 체크를 지원한다. 또한, Soufflé 는 병렬 처리를 지원한다.

4. 실험

4.1. 실험 환경

앞서 설명한 네 개의 시스템에 대한 성능 분석을 진행하였다. 네 시스템 모두 기본 설정으로 실험하였으며, 구체적으로 실험에 사용한 시스템 버전은 CORAL 1.5.1, LogicBlox 3.10.30, XSB 3.6, Soufflé 1.0.0 이다.

실험에 사용한 질의는 대표적인 데이터로그 벤치마크 질의인 이행적 폐쇄(transitive closure), 동세대(same generation)와 데이터로그의 대표적인 응용인 프로그램 분석의 근간이 되는 포인터 분석(pointer analysis) 질의를 사용하였다.

이행적 폐쇄, 동세대 질의에 사용한 데이터셋은 GraphStream 라이브러리[5]를 이용해 생성한 합성 그래프이다. 사용한 데이터셋의 특성은 표 1 과 같다. tree11, tree17 데이터셋은 각각 높이가 11, 17 인 트리 형태의 그래프를 의미하고, grid150 데이터셋은 150*150 개의 셀(cell)을 갖는 격자 형태의 그래프이며, sf_100k 는 100k 개의 정점을 갖는 스케일-프리(scale-free) 그래프, gnp_0.001, gnp_0.01, gnp_0.1, gnp_0.5 는 10000 개의 정점을 가지고 두 정점 사이의 간선이 존재할 확률이 각각 0.001, 0.01, 0.1, 0.5 가 되는 랜덤 그래프이다.

포인터 분석에서 사용한 데이터셋은 총 1.4M 개의 변수, 350K 개의 객체, 160K 개의 메서드 등으로 이루어진 OpenJDK7 라이브러리 코드에 대한 관계형 데이터셋인 OpenJDK-b147[6]을 사용하였다.

	# vertices	# edges	# paths	# SGs
tree11	71,391	71,390	805,001	2,086,343,364
tree17	13,766,856	13,766,855	237,977,708	-
grid150	22,801	45,300	131,675,775	2,295,350
sf_100k	100,000	399,101	145,082,754	221,944,925
gnp_0.001	10,000	100,185	100,000,000	100,000,000
gnp_0.01	10,000	999,720	100,000,000	100,000,000
gnp_0.1	10,000	9,999,550	100,000,000	100,000,000
gnp_0.5	10,000	49,986,806	100,000,000	100,000,000

<표 1> 합성 그래프 데이터셋의 특성

실험에 사용한 컴퓨터의 환경은 Intel(R) Xeon(R) CPU E5-2680 v3 CPU, 256GB RAM, Centos 6.7 이다.

4.2. 이행적 폐쇄

방향그래프에서 간선을 따라 접근 가능한 모든 정점을 찾는 질의이다. 구체적으로, 아래와 같은 규칙들의 집합(ruleset)으로 표현된다.

$path(X,Y) \leftarrow edge(X,Y).$
$path(X,Y) \leftarrow path(X,Z), edge(Z,Y).$ (식 3)
$? path(X,Y).$

표 2 는 tree, grid, sf, gnp 데이터셋에 대한 네 가지 시스템들의 이행적 폐쇄 질의 성능을 보여준다. tree17 을 제외한 모든 데이터셋에서 Soufflé, XSB, LogicBlox, CORAL 순서대로 좋은 성능을 보였다. 또한, 네 시스템 모두 전반적으로 결과 개수(표 1 에서 # paths)가 적은 데이터셋보다 결과 개수가 많은 데이터셋에서 더 오래 걸리는 경향을 보였다. 특히 grid 와 gnp 는

그래프의 특징상 정점의 개수에 비해 결과의 개수가 많고, 중복되는 결과를 많이 생성한다. CORAL 은 생성된 결과의 중복을 확인하는 연산이 가장 큰 병목이 되는 시스템으로 이 두 데이터셋에서 특히 매우 안좋은 성능을 보였다. 또한, gnp 데이터셋에서 간선의 수가 늘어남에 따라 중복된 결과가 많이 발생하여, CORAL 과 LogicBlox 는 각각 24 시간, 6 시간의 타임아웃에 걸렸다. Soufflé 는 작성된 데이터로그 프로그램에 따라 트라이, 비트리 등의 색인을 선택적으로 사용함으로써 효과적으로 조인 및 중복 확인 연산을 처리하고, 모든 데이터셋에서 가장 좋은 성능을 보였다.

	CORAL	LogicBlox	XSB	Soufflé
tree11	6.25	1.34	0.54	0.15
tree17	2,517.32	6h++	311.67	38.96
grid150	16,658.56	391.69	124.32	52.44
sf_100k	6,456.12	740.73	177.98	66.15
gnp_0.001	76,172.13	1,078.56	221.49	87.843
gnp_0.01	24h++	4,755.39	1,661.63	395.50
gnp_0.1	24h++	6h++	15,344.20	1,823.76
gnp_0.5	24h++	6h++	65,453.98	1,096.29

<표 2> 이행적 폐쇄 질의 결과 (단위: 초)

4.3. 동세대

동세대 질의는 한 정점으로부터 같은 거리만큼 떨어져있는 모든 정점을 찾는 질의이다. 구체적으로 아래와 같은 규칙 집합으로 구성된다.

$$sg(X,Y) \leftarrow par(X,Z), par(Y,Z).$$
$$sg(X,Y) \leftarrow par(X,Z), sg(Z,W), par(Y,W). \qquad (식\ 4)$$
$$?\ sg(X,Y).$$

	CORAL	LogicBlox	XSB	Soufflé
tree11	12h++	8h++	1074.18	202.33
tree17	12h++	8h++	16h++	16h++
grid150	82.06	4h++	4.05	4.38
sf_100k	12h++	4h++	1106.43	456.085
gnp_0.001	4days++	8h++	2,304.73	712.39
gnp_0.01	4days++	8h++	281,579.61	36,033.27
gnp_0.1	4days++	8h++	3days++	10h++
gnp_0.5	4days++	8h++	3days++	10h++

<표 3> 동세대 질의 결과 (단위: 초)

표 3 은 tree, grid, sf, gnp 데이터셋에 대한 네 가지 시스템들의 동세대 질의 성능을 보여준다. 동세대 질의는 이행적 폐쇄 질의에 비하여 더 많은 조인 연산을 필요로 하고, 데이터셋에 따라 중복된 결과를 매우 많이 생성할 수 있다. CORAL 은 결과 개수(표 1 에서 # SGs)가 적은 grid150 을 제외한 모든 데이터셋에서, LogicBlox 는 모든 데이터셋에서 타임아웃에 걸렸다. XSB 는 결과 개수가 적은 grid150 에서는 Soufflé 와 비슷한 성능을 보였지만 결과 개수가 많은 다른 데이터셋들에서는 Soufflé 에 비해 최대 7.8 배정도 느린 성능을 보였다.

4.4. 포인터 분석

포인터 분석은 프로그램의 변수, 객체, 메서드 호출, 타입 등의 정보로부터 포인터 변수들이 가리킬 수 있는 객체들을 찾는 질의이며, 데이터로그의 가장 대표적인 응용인 다양한 프로그램 분석의 근간이 된다.

	CORAL	LogicBlox	XSB	Soufflé
OpenJDK7_b147	out of memory	12h++	12h++	124.77

<표 4> 포인터 분석 질의 결과 (단위: 초)

포인터 분석 질의는 총 32 개의 규칙을 갖는 복잡한 질의이며, 사용한 데이터셋인 OpenJDK-b147 는 총 4.9M 개의 튜플을 갖고 있어 앞의 실험에 비해 비용이 크다. 표 4 는 포인터 분석 질의에 대한 실험 결과를 보여준다. CORAL 은 out-of-memory 에러로 실험을 할 수 없었고, LogicBlox 와 XSB 는 12 시간 타임아웃에 걸렸으며 Soufflé 만이 124.77 초로 합리적인 시간 안에 결과를 얻을 수 있었다.

5. 결론

본 연구에서는 다양한 데이터로그 시스템을 조사하고, 이들의 성능을 비교하였다. 상향식 처리 시스템으로 CORAL, LogicBlox, Soufflé 를, 하향식 처리 시스템으로 XSB 를 선택하여, 이행적 폐쇄, 동세대, 포인터 분석 세가지 질의에 대해 실험하였다. 실험 결과 Soufflé 가 대체적으로 가장 좋은 성능을 보였다.

참고문헌

[1] Ramakrishnan, R., Srivastava, D., Sudarshan, S., and Seshadri, P., “The CORAL Deductive System,” *The VLDB Journal—The International Journal on Very Large Data Bases*, 3(2), pp. 161-210, 1994.

[2] Aref, M., Cate, B. t., Green, T. J., Kimelfeld, B., Olteanu, D., Pasalic, E., Veldhuizen, T. L., and Washburn, G., “Design and Implementation of the LogicBlox System,” In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 1371-1382, 2015.

[3] Swift, T., and Warren, D. S., “XSB: Extending Prolog with Tabled Logic Programming,” *Theory and Practice of Logic Programming*, 12(1-2), pp. 157-187, 2012.

[4] Scholz, B., Jordan, H., Subotic, P., and Westmann, T., “On fast large-scale program analysis in Datalog,” In *Proceedings of the 25th International Conference on Compiler Construction*, pp. 196-206, 2016.

[5] GraphStream library, http://graphstream-project.org.

[6] Oracle OpenJDK7-b147 dataset, https://www.oracle.com/technetwork/oracle-labs/datasets/downloads/index.html

[7] Beeri, C., and Ramakrishnan, R., “On the power of magic,” *The journal of logic programming*, 10(3-4), pp. 255–299, 1991.

[8] Chen, W., and Warren, D. S., "Tabled evaluation with delaying for general logic programs," *Journal of the ACM (JACM)*, 43(1), pp. 20-74, 1996.

[9] Rao, P., Ramakrishnan, C. R., and Ramakrishnan, I. V., "A Thread in Time Saves Tabling Time," In *JICSLP*, pp. 112-126, 1996.