상황인식 서비스의 안정적 운영을 위한 온톨로지 추론 엔진 선택을 위한 사례기반추론 접근법*

심재문** · 권오병***†

A Case-Based Reasoning Approach to Ontology Inference Engine Selection for Robust Context-Aware Services*

Shim Jaemoon** · Kwon Ohbyung***

■ Abstract ■

Owl-based ontology is useful to realize the context-aware services which are composed of the distributed and self-configuring modules. Many ontology-based inference engines are developed to infer useful information from ontology. Since these engines show the uniqueness in terms of speed and information richness, it's difficult to ensure stable operation in providing dynamic context-aware services, especially when they should deal with the complex and big-size ontology. To provide a best inference service, the purpose of this paper is to propose a novel methodology of context-aware engine selection in a contextually prompt manner. Case-based reasoning is applied to identify the causality between context and inference engined to be selected. Finally, a series of experiments is performed with a novel evaluation methodology to what extent the methodology works better than competitive methods on an actual context-aware service.

Keyword : Context-aware Services, Ubiquitous Computing, Case-based Reasoning, Ontology, Ontology-based Inference Engine

논문접수일 : 2007년 08월 23일 논문게재확정일 : 2008년 03월 25일
논문수정일(1차 : 2008년 03월 17일)
* 본 연구는 21세기 프론티어 연구개발사업의 일환으로 추진되고 있는 지식경제부의 유비쿼터스컴퓨팅 및 네트워크원천기반기술개발사업의 08B3-S1-10M 과제로 지원된 것임.
** 경희대학교 비스니스&서비스 연구센터 연구원
*** 경희대학교 국제경영학부 교수
† 교신저자

1. 서 론

최근 유비쿼터스 도시 등 유비쿼터스 공간 개발이 계획되면서 다양한 유비쿼터스 애플리케이션이 제안되고 있다. 이러한 유비쿼터스 애플리케이션들은 분산되어 있는 서버와 스마트 객체 사이의 협동적 작업을 필요로 할 것이며 이를 위해 많은 정보나 지식이 공유되어질 것이다. 현재 이러한 정보나 지식의 공유를 위해서 온톨로지 방식이 제안되고 있다[14]. 하지만 대용량의 온톨로지 자료를 빈번하게 사용하는 상황인식 서비스 환경에서는 시스템의 실패나 끊김이 없이 제공할 수 있을 정도의 추론 엔진 성능 제고가 중요한 관건이 될 것이다. 그런데 현재의 대부분의 온톨로지 기반 추론엔진들은 이러한 유비쿼터스 서비스의 동적환경에 적합하게 설계되어 있지 않고 단일 환경에서 정적으로 운영될 것을 가정하여 개발되어 있다. 이는 향후 상황인식 서비스 처럼 동적으로 온톨로지를 참조하고 추론해야 하는 경우 심각한 성능의 저하 및 오류를 보일 수 있다. 이런 현상은 추론 성능에 의해서 직접적인 영향을 받는 부분이므로 이의 극복을 위해서는 추론 엔진의 성능 향상 또는 재개발이 필요하다[10, 16].

기존의 추론 성능 향상을 위한 연구로는 새로운 추론 엔진 및 알고리즘을 개발 하는 방법과, 필요한 온톨로지만을 추론에 포함 시키는 방법, 그리고 메모리 사이즈를 줄이기 위한 인덱싱 방법 등이 있다[11, 13, 18]. 하지만 동적으로 변화하는 상황에서 이러한 세가지 방법 중 하나를 채택한 단일 엔진로는 만족할 만한 성능을 보장하지 못한다. 특히 단일 추론 엔진은 고정된 상황을 가정하여 추론 알고리즘이 결정 되어서 나오는데, 상황기반 애플리케이션에서는 상황이 동적으로 변화하므로 추론 엔진이 서비스되다가 어느 순간에 비적절한 환경에 처하게 되면 급격히 낮은 추론 성능을 보일 가능성이 높다. 예를 들어 DB방식의 미네르바(Minerva) 추론엔진은 높은 수준의 추론 완전도(completeness)를 나타내므로 정교한 추론 정보 제공에는 적절함에도 불구하고 복잡한 온톨로지 스키마와 이를 이용한 복잡한 데이터 구조로 인해 상당량의 추론 연산을 유발시킨다는 문제점을 가지고 있으며, 이러한 문제는 결국 추론 서비스 응답 시간의 상당한 지연을 초래한다. 이는 상황 정보의 변화가 어는 시점에서 추론 연산 속도 보다 빠르게 바뀌어지면 미네르바는 무의미한 추론 결과를 돌려 줄 것이기 때문이다. 한편 대표적 메모리 기반 추론엔진인 Jena는 비교적 낮은 수준의 추론 서비스를 제공하는 대신 빠른 추론 서비스 응답 시간을 보장하지만, 미네르바가 DB 기반의 스토리지를 관리하는데 반해 Jena는 메모리 기반의 스토리지 관리 기능을 이용하기 때문에 대용량의 온톨로지 데이터를 로드 및 추론 연산을 수행해야 하는 경우 메모리 부족으로 인한 프로그램 오류를 보여 줄 가능성이 크다. 따라서 추론을 하는 상황인식 서비스 환경의 동적인 특성에 따라서 어떤 추론 엔진을 사용하여 구동해야 하는지를 선택해야 한다. 그러나 아직 상황인식 서비스 환경에 따라서 상황에 따라 적절한 추론 엔진을 선택하는 방법론에 대한 연구는 거의 없었다[10 16].

따라서 본 연구의 목적은 상황에 따라서 적합한 온톨로지 기반 추론엔진을 지능적이고 선택하여 최적의 온톨로지 기반 추론 서비스를 자율적으로 제공하도록 하는 방법론을 제안하는 것이다. 이를 통해 추론에 대한 보장 및 추론엔진의 수행 오류를 막아 줌으로써 시스템의 안전성(robustness)과 규모성(scalability)을 제고하고자 한다. 더 나아가서 복합적인 추론 성능 향상도 도모하려 한다. 그리고 통합 엔진 운영시 발생하는 문제점을 해결하기 위해서 새로운 아키텍쳐 사용 방법을 제안하였다.

본 논문의 구성은 다음과 같다. 먼저 온톨로지 기반의 추론엔진에 대한 문헌 연구를 제 2장에 기술하였다. 그리고 제 3장에는 추론 엔진을 지능적으로 선택하는 방법론을 엔진 평가 방법과 함께 제안하였다. 추론 엔진 평가 방법을 활용한 실제 성능 평가 결과가 제 4장에 기술되어 있으며, 본 연구의 공헌과 추후 연구방향을 제 5장에 보였다.

2. 온톨로지 기반 추론 엔진

온톨로지 기반 추론 엔진은 온톨로지로부터 얻은 지식과 사실을 바탕으로 추론 규칙을 적용하여 새로운 개념이나 사실을 유추하는 기능을 실행한다. 즉, 온톨로지를 대상으로 추론 엔진을 통해서 질의를 돌려주게 된다. 이때 추론 엔진은 질의어의 관계를 파악하여 관련 용어를 검색 엔진에 전달하게 된다. 그런데 이 수준은 추론 엔진이 어떤 추론 논리와 표현력을 얼마나 지원하느냐에 따라 다르다[1, 19].

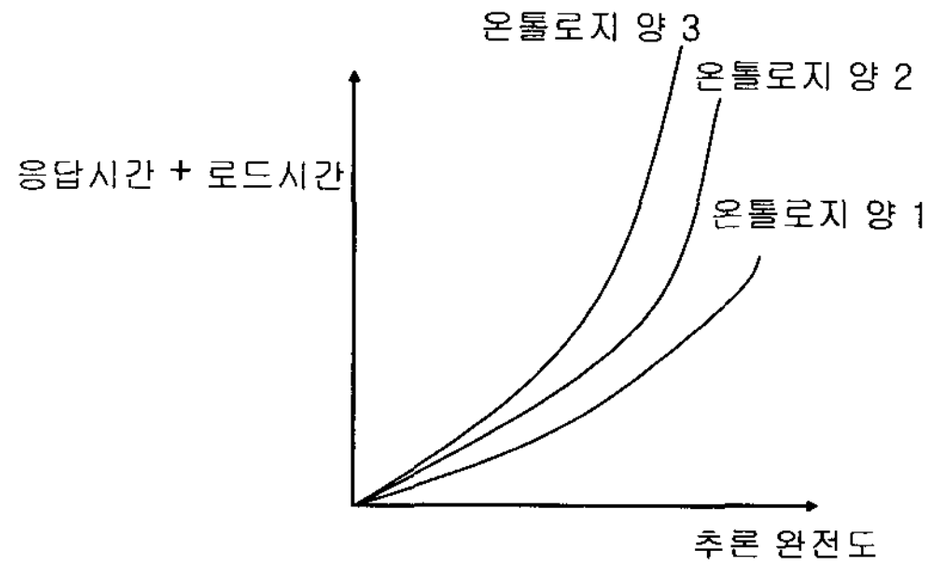
본 논문에서는 온톨로지에 관한 추론법을 DL (Description Logic) 수준으로 보았다. DL은 논리 기반의 지식 표현법의 한 종류인데 이 수준의 지식 표현을 위해 OIL, DAML + OIL, OWL등의 온톨로지 언어들이 주로 채택되고 있으며, 이중 OWL은 W3C에서 권고하는 온톨로지 지식 표현법이다[2, 6, 20]. 그리고 DL 수준의 온톨로지 및 지식을 표현하고 추론할 수 있는 여러 시스템이 존재한다. 첫 번째로, 미네르바는 IBM에서 개발된 추론 엔진이며 현재는 의미망을 위한 IODT(Intergrated Ontology Development Toolkit)안에 포함되어 제공되고 있다. 현존 온톨로지 추론 엔진 중 가장 우수한 추론 수준을 제공하는 것으로 알려져 있으며, DB2를 스토리지로 사용하기 때문에 대용량 추론 혹은 데이터를 관리 할 수 있다[7]. 둘째로, Jena는 자바로 구현된 의미망 어플리케이션이다. Jena를 개발하기 위해 HP Labs Semantic Web Program으로부터 연구가 시작되었으며, 현재 오픈 소스가 제공되므로 Pellet과 DLDB-OWL 등의 다른 추론엔진들이 Jena API를 활용하고 있다[1, 8]. 세 번째로, Pellet은 Midwap 연구소에서 개발하여 제공하며, 자체적인 엔진 뿐만 아니라, Jena를 기반으로 하는 API를 제공한다. 현재 오픈 소스를 제공하고 있으며, 상업용의 서비스를 시도하려고 하고 있다[17]. 마지막으로 DLDB-HAWK는 Lehigh 대학이 구현한 DLDB는 OWL에 대한 추론 역량을 첨부한 관계형 데이터베이스 관리시스템을 확장한 지식베이스 시스템이다[4, 5].

이와 같은 추론 엔진은 크게 추론된 결과 및 온톨로지 데이터를 저장하는 저장소의 유형에 따라 메모리형과 데이터 베이스 형으로 분류되고 있다[5, 12]. 이에 대한 설명은 <표 1>에 있다.

3. 지능형 추론 엔진 선택 방법론

3.1 추론 엔진 선택 프레임워크

추론 엔진은 구현된 알고리즘 스토리지 종류에 의하여 상이한 성능을 보여주는데, 대체로 추론 완전도 성능과 반응시간(응답시간 + 로드시간)은 서로 반비례하는 경향이 있다. 즉, [그림 1]과 같이 높은



단, 추론의 복잡도는 고정
(즉, 온톨로지 스카마는 고정)

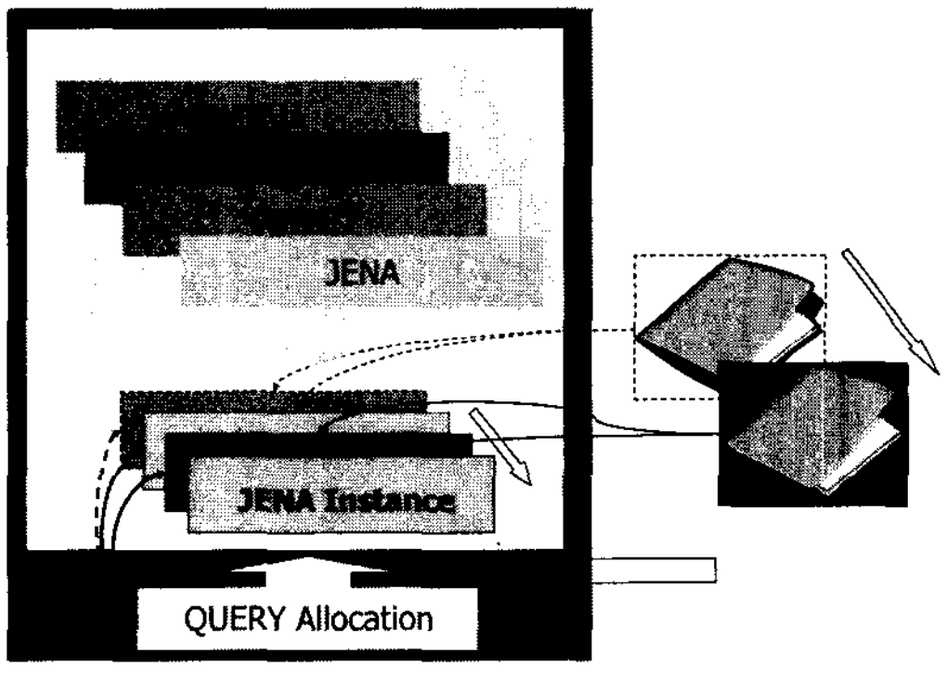
[그림 1] 추론 엔진 성능간 Trade-off 현상(개념도)

〈표 1〉 추론 기관에 대한 주요한 분류 기준

분 류	설 명
메인 메모리형 시스템 (Main Memory Systems)	추론 결과를 메모리에 저장. 메모리가 부족하면 추론 불가함 주로 쿼리 발생 시에 추론을 시작함
데이터베이스형 시스템 (DBMS Based Systems)	추론 결과 및 연산 내용을 DB에 저장하는 방식으로, 메모리 보다 더 큰 용량의 데이터를 다룰 수 있다. 하지만, DB를 관리해야 하는 부가 작업이 요구되며, 데이터 로드 시점에 추론을 시작함

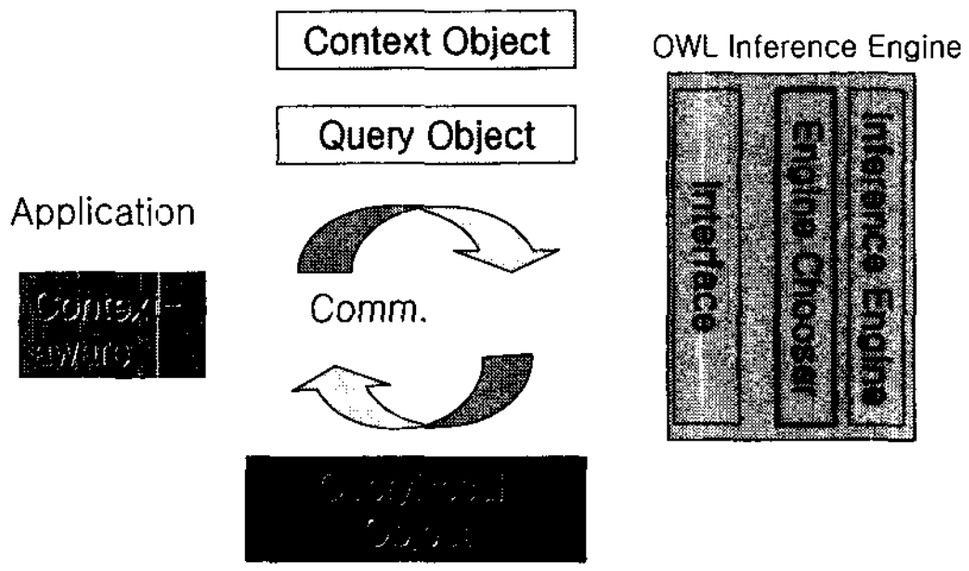
완전도는 높은 질의 추론을 제공하지만 한편으로는 반응시간이 느려지는 결과를 가져온다. 그리고, 각 엔진은 나름의 추론 완전도를 가진다[10, 16, 19].

그런데 현재까지의 추론엔진들은 기본적으로 정적인 상황을 가정하여 설계되었으며 추론 환경이 수시로 변하는 동적인 유비쿼터스 환경에선 부적절할 수 있다. 예를 들어 시장의 상황을 보더라도 시간 대와 날짜 대별로 이용 인원은 수시로 변화하게 되며, 이 경우 상황이 바뀌었을 때 추론 엔진의 안정적인 작업 수행을 보장할 수 없게 만든다. 그래서, 동적으로 변화하는 상황에 맞게 적용할 추론 엔진을 수시로 변경해 줄 필요가 있다. 이를 위해 본 연구에서 제안하는 추론 엔진 선택에 사용되는 개념적 프레임워크는 다음 [그림 2]와 같다.



[그림 2] 지능형 추론 시스템의 개념적 프레임워크

[그림 2]에서 보이는 바와 같이 통합 엔진은 여러 개의 엔진을 묶어서 사용하게 된다. 엔진 선택 주기에 따라서 선택하여 할당을 하는 구조로 운영된다. 이때 선택된 엔진에 대해서 인스턴스가 생성되며, 그 후 데이터를 로드하게 되고, 로드가 완료되면 쿼리 서비스를 할 수 있는 상태가 된다. 쿼리 서비스를 종료하면 인스턴스가 소멸된다. 하지만, 현재의 통합 엔진에서는 라이브러리의 충돌을 피해기 위해서 독립 애플리케이션으로 작동을 하며, 이를 RMI를 통해서 접속하여 엔진을 사용할 수 있도록 한다. 즉, 쿼리 서비스가 종료 되어도 인스턴스는 살아 있게 된다. 그리고 각 엔진은 독립 엔진 구현시 RMI를 이용하지 않는 상태와 같도록 하기 위해서 스텁 클래스를 제공하게 되는데, 이를 사용하면 코드의 변경이 필요없다. 한편 추론 엔진 선택 구조는 다음 [그림 3]과 같다.



[그림 3] 통합 추론엔진 선택 구조

[그림 3]처럼 온톨로지 추론 엔진은 인터페이스를 통해서 접속이 가능한데, 이는 엔진마다 작동하는 API가 다르기 때문에 선택된 엔진을 조작하는 통일성을 주기 위해서이다. 한편 SPARQL언어 형태의 질의 및 쿼리 결과 추출법도 달라서 이에 관현 API도 개발하였다.

3.2 사례기반추론방법을 이용한 엔진 선택 방법

추론 서비스의 동적인 상황에 따라서 어떤 추론 엔진을 구동시켜야 하는지를 선택하는 방법으로 사례기반추론방법을 채택하였다[9]. 사례기반추론법을 사용한 이유는 기술성 보다는 경제성 측면의 특징때문이다. 예를 들어 규칙기반의 추론방법을 고려하려면 상황기반 서비스의 입력변수에 해당하는 상황 변수들이 매우 다양한 관계로 천문학적인 규칙을 요구할 수도 있어 지식베이스 구축 자체에 매우 많은 시간과 비용이 든다. 또한 지식베이스가 구축 가능하다고 하더라도 상황인식 서비스의 원래 취지 자체가 서비스의 개인화에 있기 때문에 그 과정에서 규칙의 변경은 높은 유지보수 비용을 야기하게 된다. 따라서 본 연구는 이러한 상황인식 서비스라고 하는 도메인의 특성 상 사례기반추론을 선택하였다. 사례기반추론을 위한 사례의 구조는 <표 2>와 같다. 사례기반

〈표 2〉 사례기반추론 구성 속성표

Variable	데이터 형식(단위)	Description
SCHEMA	텍스트(Name)	온톨로지에 대한 스키마의 종류를 나타냄. 스키마는 함의적인 정보를 찾을 수 있도록 복잡합 관계를 맺어 준다.
SET	텍스트(Name)	온톨로지 데이터 집합으로 특정 스키마를 이용해서, 생성된 데이터 종류이다. 본 실험에서는 3가지로 구성이 되어 있다.
INSTANCE	숫자(#)	데이터 셋의 특성을 기술하는 요소 중 하나로 온톨로지 클래스로 생성된 Individual의 개수를 나타낸다.
TRIPLE	숫자(#)	데이터 셋의 특성을 기술하는 요소 중 하나로 subject, predicate, object 관계를 나타내는 기본 관계 데이터의 개수를 나타낸다.
QUERY	텍스트(Index)	추론 서비스를 위한 질의의 종류를 나타내며, 본 실험에서는 16개의 질의 중, 3개의 질의가 추론 서비스에 요청이 된다고 가정하였다.
QUERY_CYCLE	숫자(msec)	질의가 요청되는 주기 시간을 나타내며 본 연구에서는 일정한 주기를 가진다.
UPDATE_CYCLE	숫자(msec)	컨텍스트 데이터(온톨로지 데이터)가 바뀌는 주기 시간을 나타낸다.
DATA_SIZE	숫자(byte)	데이터 셋의 특성을 기술하는 요소 중 하나로서 물리적인 사이즈를 말한다.
CPU	텍스트(Name)	추론 서비스가 구동되고 있는 컴퓨팅 자원에 대한 기술 요소 중 하나로 컴퓨터의 CPU 종류를 기술한다.
MEMORY	숫자(MB)	추론 서비스가 구동되고 있는 컴퓨팅 자원에 대한 기술 요소 중 하나로 컴퓨터의 메모리 크기를 기술한다.
선택할 추론엔진	텍스트(Name)	사례기반추론의 결과로 최종 선택되는 엔진의 이름이며, 위의 환경에서 가정 최적의 성능을 나타내는 엔진을 기술한다.

추론을 위한 속성값 중 평균 컨텍스트 변경 주기, 평균 쿼리 주기, 온톨로지 데이터 로드 사이즈는 서비스 수행 중에 구해지며, 나머지는 서버 구동 전에 파라미터 값으로 주어져서 사용되어 진다.

이중에서 QUERY_CYCLE, UPDATE_CYCLE, DATA_SIZE는 작동 중에 구해지며, 나머지는 엔진 구동 초기에 입력을 하여준다. 또한 LOAD SIZE는 로드할 온톨로지의 성질을 나타내고 있으나 보다 정교하게 측정하려면 INSTANCE와 TRIPLE을 사용할 수도 있다.

특정 상황 하에서 실제 유사사례추출에 의한 추론 엔진 선정은 유클리디언 방법을 사용하였다. 단거리를 구하는 함수에서 텍스트형은 동일한 경우 0, 동일하지 않은 경우 1의 값으로 반응하게 하였고, 숫자의 경우에는 비교 대상과 사례 사이의 값의 차이를 [0, 1]의 범위 내로 정규화하여 구하였다. 다음 [그림 4]는 구축된 사례베이스의 일부이며, 평가가 진행될 때마다 자동적으로 사례가 새롭게 추가 및 갱신될 수 있다.

3.3 평가를 위한 프레임워크

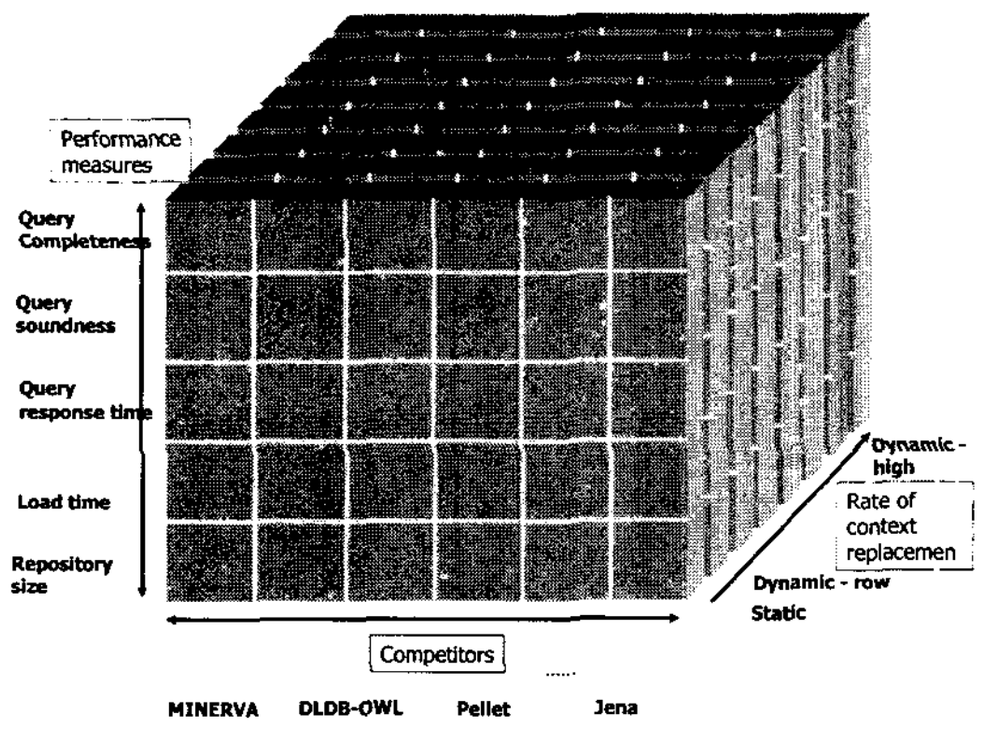
추론 엔진의 구현은 후 성능을 평가하기 위해서 다음 [그림 4]와 같은 평가 프레임워크를 소개한다. 이 평가 프레임워크는 Lehigh University에서 최초로 제안 되었고[5], IBM China Lab에서 수정[12]한 것에 동적 평가 부분을 추가한 것이다[10, 16]. 추론 엔진들에 대한 평가를 위한 프레임워크는 다음 [그림 5]와 같다.

본 프레임워크는 평가 추론 엔진의 기본 수행 능력에 대한 테스트인 정적 평가와 실제 유비쿼터스 환경에서 어떤 성능을 보여주는 지에 대한 테스트인 동적 테스트로 이루어져 있다. 예를 들면, 정적 평가 항목에서 Repository Size를 제외한 항목에 대해서 높은 점수를 받으면 엔진의 기본적인 추론 능력이 높다는 의미이다. 그리고, 동적 평가에서 Dynamic이 높다는 의미는 주어진 시간안에 추론 엔진이 빠른 시간내에 새로운 컨텍스트 데이터를 로드 및 추론해서 응답을 제공해준다는 의미이다. 즉 변화하는 동적 환경에서의 응답 능력을 평가한 것이다.

case : 테이블

SCHEMA	SET	INSTANCE	TRIPLE	QUERY	QUERY_CYCL	UPDATE_CYC	DATA_SIZE	CPU	MEMORY	ENGINE
[illegible]	1	8477	134302	6,13,16	10000	600000	12332441	Intel Core2 6300	1	JENA
unvi-bench-dl	1	8477	134302	6,13,16	60000	600000	12332441	Intel Core2 6300	1	JENA
unvi-bench-dl	1	8477	134302	6,13,16	600000	600000	12332441	Intel Core2 6300	1	MINERVA
unvi-bench-dl	1	8477	134302	6,13,16	10000	900000	12332441	Intel Core2 6300	1	JENA
unvi-bench-dl	1	8477	134302	6,13,16	60000	900000	12332441	Intel Core2 6300	1	JENA
unvi-bench-dl	1	8477	134302	6,13,16	600000	900000	12332441	Intel Core2 6300	1	PELLET
unvi-bench-dl	1	8477	134302	6,13,16	10000	1200000	12332441	Intel Core2 6300	1	JENA
unvi-bench-dl	1	8477	134302	6,13,16	60000	1200000	12332441	Intel Core2 6300	1	JENA
unvi-bench-dl	1	8477	134302	6,13,16	600000	1200000	12332441	Intel Core2 6300	1	MINERVA
unvi-bench-dl	2	3857	60791	6,13,16	10000	600000	5580073	Intel Core2 6300	1	JENA
unvi-bench-dl	2	3857	60791	6,13,16	60000	600000	5580073	Intel Core2 6300	1	JENA
unvi-bench-dl	2	3857	60791	6,13,16	600000	600000	5580073	Intel Core2 6300	1	MINERVA
unvi-bench-dl	2	3857	60791	6,13,16	10000	900000	5580073	Intel Core2 6300	1	JENA
unvi-bench-dl	2	3857	60791	6,13,16	60000	900000	5580073	Intel Core2 6300	1	JENA
unvi-bench-dl	2	3857	60791	6,13,16	600000	900000	5580073	Intel Core2 6300	1	MINERVA
unvi-bench-dl	2	3857	60791	6,13,16	10000	1200000	5580073	Intel Core2 6300	1	JENA
unvi-bench-dl	2	3857	60791	6,13,16	60000	1200000	5580073	Intel Core2 6300	1	MINERVA
unvi-bench-dl	2	3857	60791	6,13,16	600000	1200000	5580073	Intel Core2 6300	1	MINERVA
unvi-bench-dl	3	19277	337873	6,13,16	10000	600000	31921701	Intel Core2 6300	1	JENA
unvi-bench-dl	3	19277	337873	6,13,16	60000	600000	31921701	Intel Core2 6300	1	JENA
unvi-bench-dl	3	19277	337873	6,13,16	600000	600000	31921701	Intel Core2 6300	1	JENA
unvi-bench-dl	3	19277	337873	6,13,16	10000	900000	31921701	Intel Core2 6300	1	JENA
unvi-bench-dl	3	19277	337873	6,13,16	60000	900000	31921701	Intel Core2 6300	1	JENA
unvi-bench-dl	3	19277	337873	6,13,16	600000	900000	31921701	Intel Core2 6300	1	JENA
unvi-bench-dl	3	19277	337873	6,13,16	10000	1200000	31921701	Intel Core2 6300	1	JENA
unvi-bench-dl	3	19277	337873	6,13,16	60000	1200000	31921701	Intel Core2 6300	1	JENA
unvi-bench-dl	3	19277	337873	6,13,16	600000	1200000	31921701	Intel Core2 6300	1	JENA

[그림 4] 구축된 사례베이스



[그림 5] 평가 프레임워크

그리고, 엔진의 기본 추론 능력인 정적 평가와 동적환경에서의 응답 능력 평가인 동적 평가를 총괄하여 엔진의 성능을 전반적으로 이해하기 위해 Combined Matirx를 개발하였다.

3.4 실험 환경

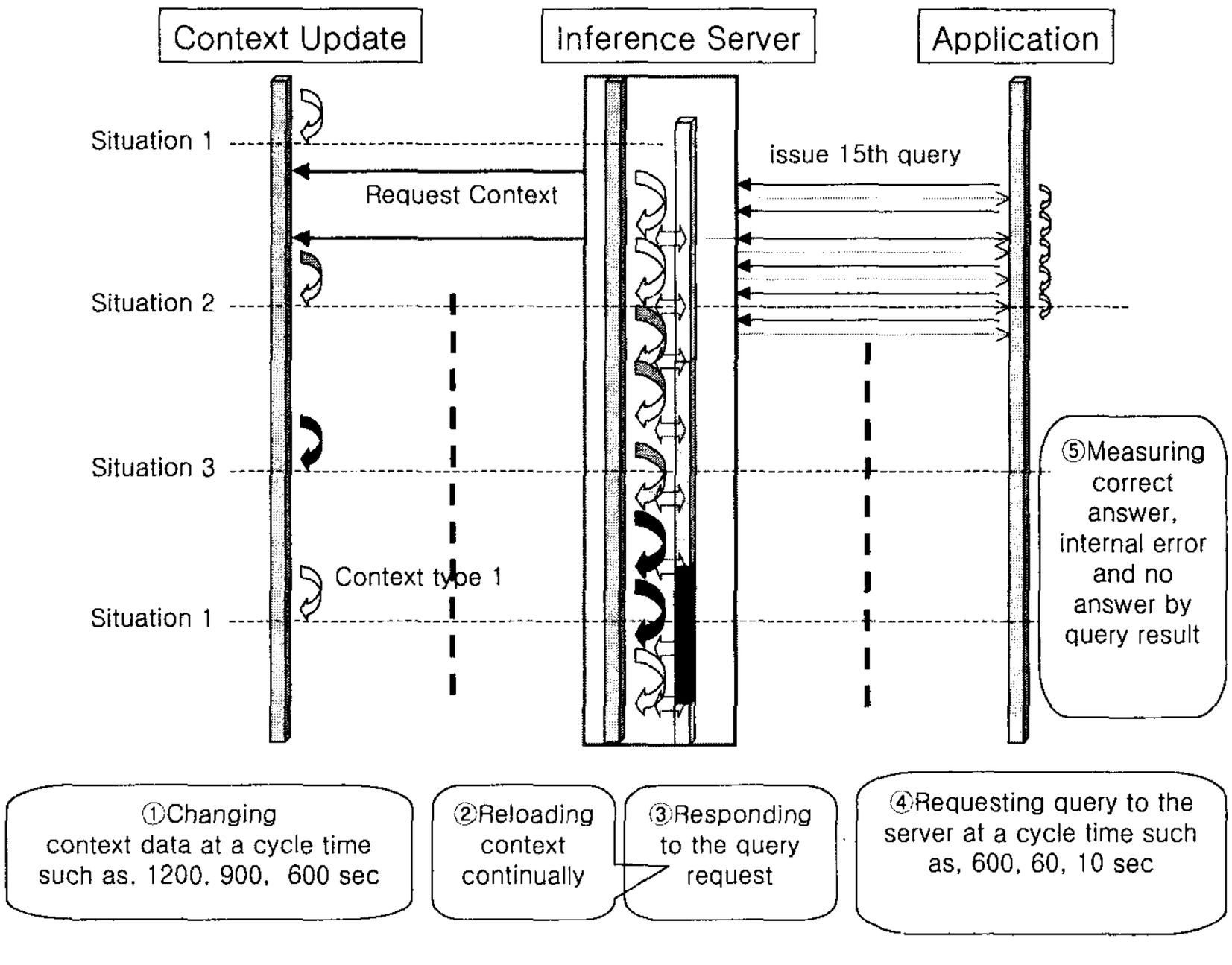
재래 시장에 적용된 온톨로지 서비스를 위해서 온톨로지 추론 서버를 구축 하였다. 구축된 서버의 성능을 측정하기 위해서 널리 쓰이는 Lehigh University와 IBM이 제공하는 온톨로지 테스트 스키마와 데이터를 적용하여 테스트하되, 동적 서비스 테스트에 맞게 확장하였다[10, 16].

먼저 사전 연구를 통해서 기존의 추론 엔진을 서비스 온톨로지가 수시로 변화하는 동적 환경에 맞게 완벽하게 서비스하는 것이 불가능하다는 것을 발견했다. 그래서 다음 [그림 6]과 같은 구현 방법 및 성능 측정 방법에 대한 프레임워크를 제시하였다[10, 16].

추론 서버는 두 개 이상의 인스턴스로 생성되어야 하며 이를 이용하여 한쪽은 온톨로지 데이터를 업데이트를 수행하고 다른 한쪽은 쿼리 서비스를 수행한다. 그리고 업데이트 수행을 마친 인스턴스는 쿼리 서비스를 수행하게 된다. 이때 쿼리 서비스를 수행하고 난 인스턴스는 다시 바인딩되거나 소멸된다.

실험 환경에 따라서, 온톨로지 변경 주기와 쿼리 발생 주기가 조합되게 된다. 또한 실험할 데이터 셋의 조합도 포함되게 된다. 추론 성능은 변경된 온톨로지 데이터를 얼마나 빨리 로드하여 유효한 시간 내에 쿼리 서비스를 제공하느냐에 달려 있다. 각 엔진 당 총 27개의 실험 조건에 따라서 각 조건 당 1시간씩 27시간의 시뮬레이션 실험이 수행되었다. 그래서 총 5개의 엔진을 합하여 총 135시간의 실험이 이루어졌다.

상황 온톨로지(Context Ontology) 데이터를 얼마나 빨리 업데이트 하여 서비스 할 수 있는지가



[그림 6] 동적 서버 구축 및 평가 프레임워크

동적환경에서의 추론 엔진의 주요 성능이 된다. 이를 평가하기 위해서 컨텍스트 온톨로지의 변경주기와 쿼리의 주기를 조정하여 여러조합을 만들어서 측정을 하여 평가를 하였다. 컨텍스트 온톨로지 양 또한 평가 조건이 될 것이다. 컨텍스트 온톨로지 데이터 집합은 앞으로 소개할 세 개의 집합을 대상으로 하여 동일하게 아래와 같은 시험 주기를 가지고 평가하였다. 각 평가 항목별로 구해지는 수치는 다음 <표 3>과 같다.

성능 측정을 위한 데이터 집합은 현실성을 증가시키기 위해서 경희대학교를 실례로 다음과 같이 선정하였다. 또한 데이터 집합 3은 그동안 Benchmakring 자료로 활용되어 왔던 Campus 예제를 사용하여 분석한 다양한 평가 결과와 비교하기 위하여 동일한 자료를 상정하였다. 평가에 활용한 데이터 집합의 특성은 그동안 수행한 연구의 일관성 유지를 위해서 다음 <표 4>와 같이 동일하게 사용하였다[10].

위의 데이터 집합의 특성은 다음과 같은 방법으로 측정되었다. Instances 값은 IBM China 연구실에서 활용한 생성 프로그램의 로그 메시지를 참조했다. Triples은 China 연구실에서 쿼리 정답을

〈표 3〉 평가 항목별 분류

항 목	설 명
정답	쿼리 시점에 해당하는 컨텍스트 정보를 바르게 돌려 줬을 때, 즉, 추론 엔진이 바뀌어진 컨텍스트 정보를 업로드 완료한 상태일 때.
오답	쿼리 시점에 해당하는 컨텍스트 정보를 바르게 돌려 주지 못했을 때, 즉, 추론 엔진이 바뀌어진 컨텍스트 정보를 업로드 하지 못한 상태일 때.
에러	Context Server의 성능 한계로 인해서, 오류가 발생 했을 때. 그리고, 앞의 쿼리가 지연되어서 뒤의 쿼리가 실행되지 못한 경우에도 오류 상황으로 인정한다.

〈표 4〉 데이터 집합 설명

데이터 집합	내 용
데이터 집합 1	경희대학교 수원캠퍼스 내 국제경영대학 및 전자정보대학 수준
	8477Ir.stances
	134302Triples
	11.8MB
데이터 집합 2	경희대학교 수원캠퍼스 내 국제경영대학 수준
	3857Instances
	60791Triples
	5.33MB
데이터 집합 3	Campus 예제 데이터 집합 1(IBM China 연구실)
	19277Instances
	337873Triples
	30.5MB

얻기 위해서 만든 프로그램을 이용해서, 데이터 집합을 모두 로드하였으며, DB 상에서 Triples 단위로 파싱해 놓은 숫자를 구하였다. 위 데이터 집합 중 1번과 2번은 3번 데이터 집합 보다 훨씬 많은 Transitive Closure triples을 가지므로 복잡하다. Transitivity는 더 많은 추론 정보를 생성해 내어야 하므로 불가피하게 연산량을 더 많이 유발하게 된다. 이렇게 데이터 집합을 구성한 이유는 단순히 온톨로지의 양이 많은 경우와 온톨로지가 복잡한 경우를 모두 고려하여 평가하기 위해서이다. 실제로 상황인식 서비스에서는 정적인 서비스에 비해 더욱 다양한 센서 데이터를 사용하게 되기 때문에 온톨로지의 양이 많은 경우도 있으며, 동시에 상황정보간 혹은 상황 정보와 기본 정보간 관계성의 복잡성으로 온톨로지가 복잡해 지는 경우도 발생한다.

4. 평가 결과

4.1 개별적 추론 엔진 평가

먼저 미네르바 추론 엔진은 상당히 높은 수준의 추론 결과를 내는 반면에 동적 환경에서는 많은 에러를 발생시켰다. 주로 발생시키는 에러는 DB와 관련된 것과 OntologyStore의 Inference Instance 생성에 관한 것이었다. Factory를 통해서 생성됨으로 두 개의 인스턴스 생성이 불가능하기 때문에, 두 개의 애플리케이션을 생성하여 Java RMI를 호출하여 서비스하도록 설계를 하여야만 했다.

HAWK는 동적 서버를 구현을 위해서 미네르바와 같은 형태의 서버 아키텍처를 구현하였다. 하지만, HAWK는 단일 애플리케이션에서 독립적인 추론 엔진 인스턴스를 생성할 수 있기 때문에 RMI로 구현될 필요가 없었다. 하지만, DL 서버인 Racer의 조작 문제로 인해서 잘 작동하지 않아 Simple DB 모드로 실험을 하였다.

한편 Pellet과 Jena는 Hawk나 미네르바와 다르게 메모리를 저장장치로 사용한다. 따라서 생성자를 통해서 새로운 추론엔진 인스턴스만 생성해 주면 되므로 비교적 간단하게 추론 서버를 구현하였다. 그러나 Pellet은 데이터 집합 3의 경우 로드가 되지 않는 문제점을 보였다. 참고로, 메모리 기반의 추론엔진은 온톨로지의 문법적 오류 이외에는 대부분 Java Heap 오류 즉, 메모리 부족 관련 에러를 보여주고 있다.

Jena는 현재의 데이터 셋 내에서는 가장 높은 수행 능력을 보여주었다. 하지만 쿼리 완전도나 안전도를 고려하면 성능의 상충성이 발생되는 것을 관찰할 수 있었다.

4.2 엔진 선택기에 의한 추론 엔진 선택법 평가 방법

위의 네 개의 추론 엔진을 단일로 사용하는 방법 외에 사례기반추론을 기반으로 한 추론 엔진 선택 방법을 고려하여 선택된 추론 엔진을 가지고 마찬가지 환경에서 동일한 문제를 가지고 성능 측정을 수행하여 보았다. 이때 각 추론엔진은 라이브러리 상의 충돌을 막기 위해서 RMI를 통한 접속을 사용하였다. 그 결과 정답과 오답과 에러 결과는 부록B와 같다. 성능의 순위 결정에는 큰 차이는 없지만 하드웨어의 메모리가 증가하면 메모리기반의 엔진이, CPU의 성능이 증가하면 일일이 추론을 수행해야 하는 DB 기반의 추론 엔진이 상대적으로 더 우수한 결과가 나올 수는 있다. 따라서 부록 B의 결과는 평가를 수행하는 환경에 따라 다소 차이날 수 있을 것이다. 본 연구에서는 10MBPS급의 네트워크에 2GB메모리를 가지는 랩탑 컴퓨터를 서버로 사용하여 평가하였다.

이에 대한 평가를 위해 <표 5>과 같은 평가 항목을 제시하였다. 이중에서 완전도와 완성도는 LUBM에서 제안하고 있는 측정치이며, 이 두 가지를 활용하여 결합측정치를 개발한 바가 있다[5]. 하지만 현재 버전의 평가툴은 OWL-Lite만을 평가할 수 있는 제약이 있다. IBM China Lab에서는 OWL-Lite 평가에 더해서, OWL-DL을 평가할 수 있는 툴을 확장 개발하였으며[12], 본 연구에서도 이 방식을 도입하였다. 그런데 이는 정적인 상황에서의 온톨로지 추론이었기 때문에 본 논문에서 관심을 가지는 상황인식 상황에서 맞지 않아 새롭게 안전도를 더 제시하였다. 안전도를 통하여 대규모의 복잡한 온톨로지 정보에 대한 잦은 쿼리 요청에 에러를 발생시키지 않고 얼마나 잘 대응하는지를 평가할 수 있었다. 따라서 본 논문에서는 LUBM에서 제시한 결합측정치를 동적 상황에서도 평가하기 위한 확장된 결합측정치(Enahnced Combined Metric, ECM)를 다음 식 (1)과 같이 제안하였다.

$$ECM(d) = \frac{1}{M}\sum_{q=1}^{M} \frac{(\alpha^2+1) * P(d, q) * F(d, q)}{\alpha^2 * P(d, q) + F(d, q)} \quad (1)$$

단, 여기서 P(d, q)와 F(d, q)는 각각 d번째 데이터 집합에서 q번째 쿼리에서의 응답 시간에 대한 성능치와 F값으로서 식 (2)와 같이 <표 5>에서 제시한 네가지 평가항목에 의하여 작성된다. 이중 좌항은 LUBM에서 제안하는 함수식이며 우항은 동적 상황에서의 안전도에 대한 평균치이고, 이 둘을 가중 평균한 것이다. 따라서 값이 낮을수록 동적인 상황을 중시하는 것이 되며, 높을수록 정적인 상황을 중시하는 것이 될 것이다. 한편 α, β 등의 조절 변수는 각각 1, 1로 설정하였다.

$$F(d, q) = \delta * \frac{(\beta^2+1) * CT(d, q) * SD(d, q)}{\beta^2 * CT(d, q) + SD(d, q)} + \quad (2)$$

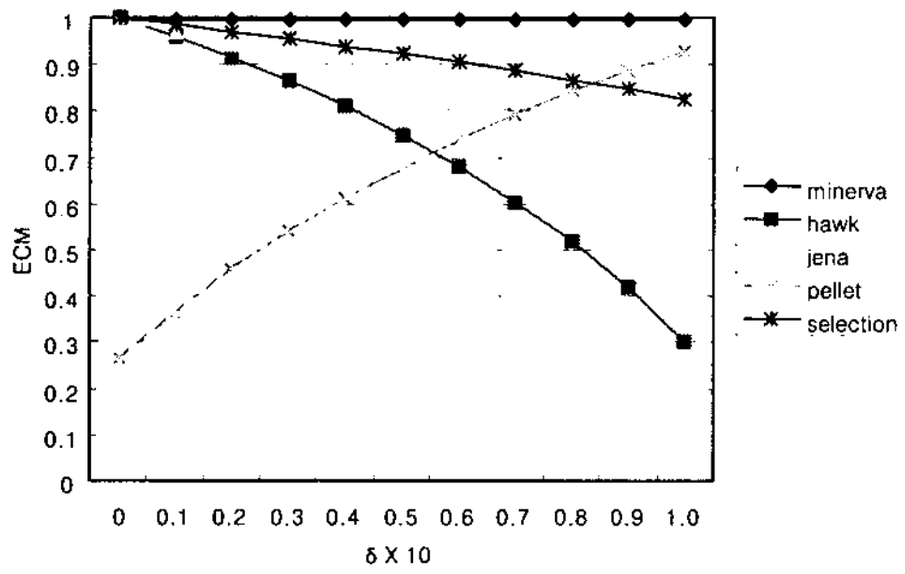
$$(1-\delta) * \frac{1}{S*T}\sum_{c1=1}^{S}\sum_{c2=1}^{T} RT(d, q, c1, c2)$$

〈표 5〉 평가 항목

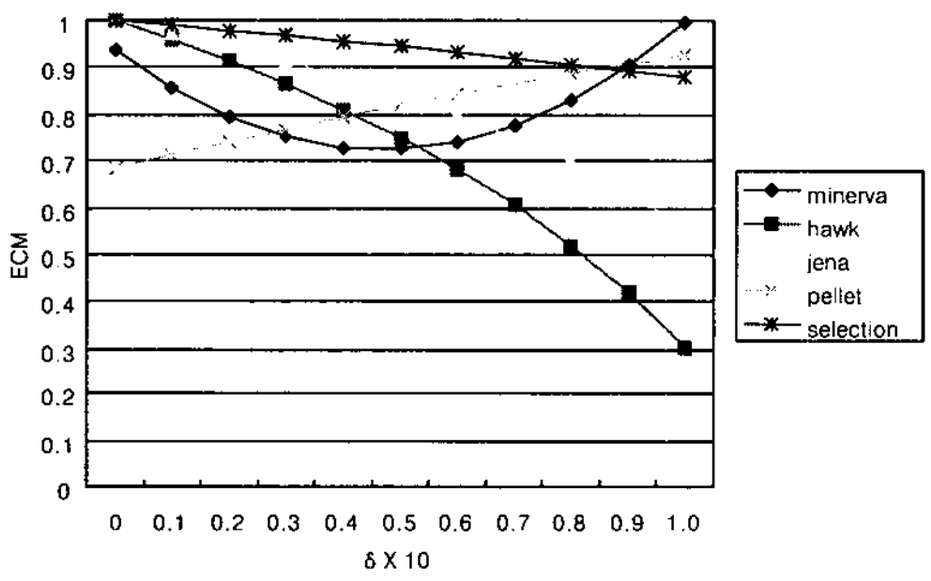
평가항목	기 호	내 용
Completeness (완전도)	CT(d, q)	추론 엔진이 정답값을 얼마나 돌려주는 것인지를 백분율로 측정
Soundness (완성도)	SD(d, q)	추론 엔진이 돌려주는 값중에, 정답이 얼마나 포함되어 있는지를 백분율로 측정
Robustness (안전도)	RT(d, q, C1, C2)	추론 엔진이 특정 온톨로지 로딩 주기(c1) 및 특정 추론 주기(c2)에서 작동을 멈추지 않고 값을 돌려주는 비율로 측정
Enahcned Combined Metric (확장된 결합 측정치)	ECM(δ)	완전도, 완성도, 안전도, 동적내구도를 포함한 결합측정치이며, δ는 정적인 평가를 중시하는 비중으로 0에서 1까지의 값을 가짐

단, d는 [vy4]에 표기된 데이터 셋 일련번호이며 q는 쿼리 번호로 1번부터 16번까지 있다.

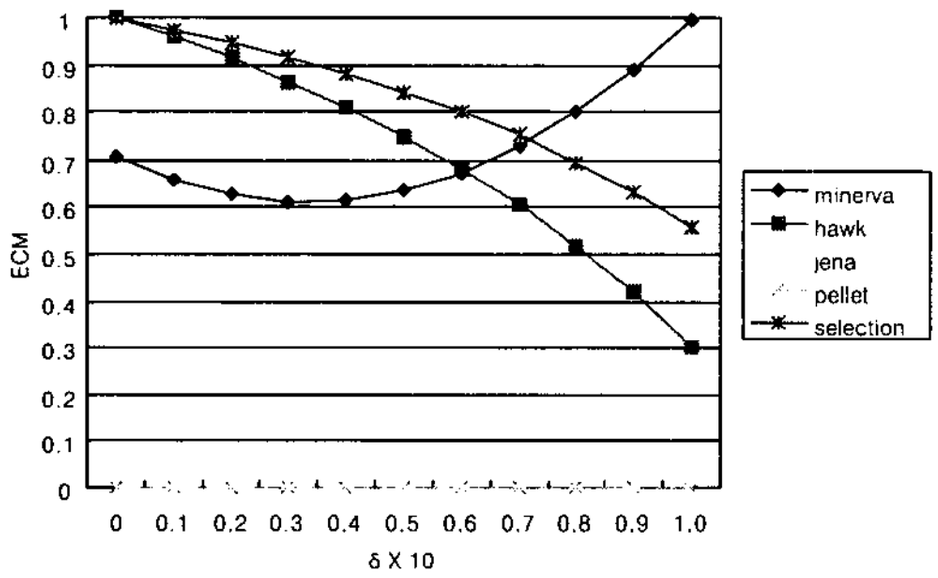
동적 결합측정치를 활용하여 데이터 집합 1, 집합 2, 집합 3에 대해 각각 평가한 결과를 각각 [그림 7]~[그림 9]에서 보였다.



[그림 7] ECM(1) 평가 결과



[그림 8] ECM(2) 평가 결과



[그림 9] ECM(3) 평가 결과

하지만, 동적으로 변화하는 환경에서는 현재의 단일엔진으로는 극단적인 한계 상황에 필연적으로 마주하기 된다. 이때 단일엔진은 서비스를 멈춰야 할 만큼의 심각한 오류를 내부에서 생성하게 된다. 우선, 엔진 선택 기법은 보이지 않는 결과로써 서비스가 다운되지 않게 한다는 결과를 보장해 준다. 또한 전체적인 성능 지표도 또한 반드시 정적 평가를 감안해야할 것이다.

데이터 집합 1은 적정한 복잡도에 데이터의 양이 다소 많은 특징을 보이고 있다. 이러한 경우 대량의 데이터 처리에 우수한 MINERVA가 전체적으로 거의 완벽한 성능을 보이고 있다. 그리고 Pellet의 경우 정적인 상황을 중시하는 도메인에서는 두 번째로 우수한 결과를 보이고 있다. 사례기반추론을 활용한 온톨로지 추론 엔진 선택법도 전체적으로 양호한 성능을 보이고 있으나 동적인 상황에서 더 좋은 성능을 보이고 있는 것을 알 수 있다.

데이터 집합 2는 적정한 복잡도에 비교적 경량의 데이터의 양을 가지는 특징을 보이고 있다. 이러한 경우 잦은 쿼리가 요구되는 동적인 상황에서는 DB중심의 추론 엔진보다는 Jena와 같은 메모리 중심의 엔진이 상대적으로 좋은 성능을 보이고 있다. 그리고 사례기반추론을 활용한 온톨로지 추론엔진 선택법이 완전히 정적인 상황이 아닌 한 가장 우수한 성능을 나타내고 있다.

데이터 집합 3은 다소 높은 복잡도에 대량의 데이터 양을 가지는 특징을 보이고 있다. 이때 Pellet은 메모리의 부족으로 대응하지 못하고 가동되지 않았다. 따라서 Pellet은 대규모적이고 복잡한 진능형 시스템 구축에 적절하지 않은 것으로 나타났다. 전체적으로는 사례기반추론을 활용한 온톨로지 추론엔진 선택법과 Jena가 거의 동일한 성능을 보였고 특히 동적인 상황에서 가장 우수한 성능을 보이고 있다. 한편 정적인 상황에서는 역시 MINERVA가 좋은 결과를 보였다.

전체적으로 보았을 때 추론 완전도가 높은 미네르바는 정적 평가에 가중치(δ값 증가)를 둘수록 높은 성능을 보여 줌을 알 수 있다. 그리고 사례기반추론을 활용한 온톨로지 추론 엔진 선택법을 활용한 통합 엔진은 비교적 안정적인 성능을 보여주었

다. 결국 동적인 상황에서의 엔진 평가를 중심으로 하여 사례기반추론을 활용한 온톨로지 추론 엔진 선택법이 우수한 성능을 보인 것으로 보인다. 특히 시장이나 대학과 같은 실제 도메인에 추론 엔진을 적용한 상황인식 서비스를 적용할 때 발생될 수 있는 가장 위중한 문제 중 하나인 추론 엔진의 다운 우려는 방지할 수 있어 매우 안전한 추론 엔진 관리 및 호출법인 것으로 나타났다. 특히 이러한 안전성은 엔진 별로 처리하지 못하는 쿼리에 대해 사례기반추론을 통해 사전에 예측하여 엔진을 선택하게 함으로써 가능했다.

다만 본 논문에서 제안한 추론 엔진 선택법이 항상 최적의 성과를 보이지 못하였는데, 그 주된 원인은 각 추론 엔진들이 기본적으로 사용하고 있는 Jena API의 버전이 각기 다름으로 인해 직접적으로 사용하지 못하고 이를 다시 RMI를 통해서 호출하도록 구현하였기 때문이다. 이는 결국 네트워크의 과부화와 불필요한 메모리 소모를 가져왔으며, 전체적인 성능이 예상한 것보다는 많이 낮아지는 결과를 낳은 것이다. 향후 엔진들이 최신의 Jena 엔진으로 업데이트 한다면, 충돌을 피할 수 있을 것으로 생각되며 그로 인해 불필요한 RMI 기법을 사용하지 않아서 추론 엔진 선택법은 더욱 높은 성능 수준을 보여줄 수 있을 것으로 생각된다.

5. 결 론

단일의 온톨로지 추론 엔진만을 가지고 대규모적이고 복잡한 동적 상황을 고려해야 하는 상황인식 서비스를 수행함에 있어서 높은 추론 수준과 빠른 추론 속도를 제공하는 하는데는 한계가 있어보인다. 그러므로 본 논문에서는 최적의 추론 활용 방법을 위해서 상황에 맞는 추론 엔진 선택법을 제안하였다. 추론 엔진의 성능 저하 현상은 추론 연산의 근본적인 성능에 대한 상충작용 때문에 발생하므로 더욱 개선된 추론 엔진이 개발될 향후에도 성능 최적화 문제는 남아 있을 것이다. 이런 의미에서 상황에 맞는 추론 엔진 선택법을 고안한 것은 의미가 있을 것이다.

추후 연구로는 미지의 온톨로지를 대상으로 추론 연산 반응 속도를 알기 위해서 보다 일반적인 측정 방법이 필요할 것이다. 그리고, 현재는 간단한 규칙 기반의 엔진 선택 방법을 사용하였는데, 향후에는 추론 엔진을 서비스 받는 상황들에 대해서도 고려하는 보다 더 지능적인 엔진 선택법에 대한 추가 연구가 진행될 것이다. 또한 향후 사례기반추론을 활용한 온톨로지 추론엔진 선택법의 성능 향상을 위해서는 사례베이스의 질 향상을 검토하면 더욱 좋은 성능을 보일 것으로 기대한다. 또한 선택된 추론 엔진 호출 과정에서 추론 엔진의 라이브러리 충돌로 우회적인 방법으로 구현되었기 때문에 이로 인해서 다소 성능의 저하가 관찰되었으므로 이 부분에 대한 보완이 필요하다. 마지막으로, 이번 연구에서는 반영되지 못했지만, 사례기반추론의 사례 지정시 사용자의 정적 혹은 정적 가중치를 고려한다면 보다 높은 성능을 보여줄 수 있을 것이다.

참 고 문 헌

[1] A Query and Inference Service for RDF. See http://www.w3.org/TandS/QL/QL98/pp/queryservice.html.

[2] Baader, F., D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, "The Description Logic Handbook : Theory, Implementation and Application," Cambridge University Press, 2002.

[3] Carroll, J.J., I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena : Implementing the Semantic Web Recommendations," Proceedings of the 13th International World Wide Web Conference, ACM Press, New York, (2004), pp.74-83.

[4] Guo, Y., Z. Pan, and J. Heflin., "An Evaluation of Knowledge Base Systems for

Large OWL Datasets," Proceedings of the 3rd International Semantic Web Conference, Hiroshima. *LNCS*, Vol.3298(2004), pp. 274-288.

[5] Guo, Y., Z. Pan, and J. Heflin, "LUBM : A Benchmark for OWL Knowledge Base Systems," *Journal of Web Semantics*, Vol.3, No.2(2005), pp.158-182.

[6] Horrocks, I., "Description Logics in Ontology Applications," *LNCS*, Vol.3702(2005), pp.2-13.

[7] IBM's IODT/Minerva team : Minerva Reasoner, See, http://www.alphaworks.ibm.com/tech/semanticstk or http://www.ifcomputer.com/MINERVA/.

[8] Kevin, W., C. Sayers, and H. Kuno, "Efficient RDF Storage and Retrieval in Jena2," *Proceedings of First International Workshop on Semantic Web and Databases*, (2003), pp.131-151.

[9] Kolodner, J.L., "An Introduction to Case-Based Reasoning," *Artificial Intelligence Review*, Vol.6, No.1(1992), pp.3-34.

[10] Kwon, O.B., J.M. Sim, and M.C. Lee, "OWL-DL Based Ontology Inference Engine Assessment for Context-Aware Services," KES-AMSTA 2007, LNAI, Vol.4496(2007), pp. 338- 347.

[11] Lee, J.H., I.S. Park, D.M. Lee, and S.J. Hyun, "An Application-Oriented Context Pre-fetch Method for Improving Inference Performance in Ontology-based Context Management," *American Association for Artificial Intelligence,* 2005.

[12] Ma, L., Yang Y., Zhaoming Q., Guotong X., Yue P., and Shengping L., "Towards A Complete OWL Ontology Benchmark," *3rd European Semantic Web Conference*, 2006.

[13] Motik, B. and U. Sattler., "A comparison of reasoning techniques for querying large description logic aboxes," In Proc. of the 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2006), 2006.

[14] Noy, N.F. and Hafner C.D., "The State of the Art in Ontology Design : A Survey and Comparative Review," *AI Magazine*, Vol. 18, No.3(1997), pp.53-74.

[15] Shadbolt, N., K. O'Hara, and L. Crow, "The Experimental Evaluation of Knowledge Acquisition Techniques and Methods : History, Problems, and New Directions," *International Journal Human-Computer Studies*, Vol.51, No.4(1999), pp.729-755.

[16] Sim, J.M., J.H. Kim, O.B. Kwon, S.S. Lee, J.H. Kim, H.K. Jang, and M.C. Lee, "Applying Inference Engine to Context-Aware Computing Services," UbiPCMM06, California, USA(2006).

[17] Sirin, E. and B. Parsia, "Pellet : An owl dl reasoner," Proceedings Of the Int. Third International Semantic Web Conference (ISWC 2004).

[18] Tug ba O¨.Z., O¨. vu¨nc, O¨. ztu¨rk, and M.O. U¨ nalır, "Optimizing a Rete-based Inference Engine using a Hybrid Heuristic and Pyramid based Indexes on Ontological Data," *Journal of computers*, Vol.2, No. 4(2007), pp.41-48.

[19] Wang, X.H., Zhang, D.Q., Gu, T., Pung, H. K., "Ontology based context modeling and reasoning using OWL," Pervasive Computing and Communications Workshops 2004. Proceedings of the Second IEEE Annual Conference, (2004), pp.18-22.

[20] Web Ontology Language (OWL), See http://www.w3.org/2004/OWL/, (2004).

〈부록 A〉 : 평가에 활용된 SPARQL 형태의 쿼리

[Query 1]

SELECT DISTINCT ?x

WHERE {?x rdf : type benchmark : UndergraduateStudent. ?x benchmark : takesCourse
http://www.Department0.University0.edu/Course0}

Description : All undergraduate students who take course
http://www.Department0.University0.edu/Course0.

It only needs simple conjunction

[Query 2]

SELECT DISTINCT ?x

WHERE {?x rdf : type benchmark : Employee}

Description : Find out all employees

Domain(worksFor, Employee), <a worksFor b> _ <a rdf : type Employee>

Domain(worksFor,Employee), researchAssistant _
_worksFor.ResearchGroup_researchAssistant_ Employee

[Query 3]

SELECT DISTINCT ?x

WHERE {?x rdf : type benchmark : Student. ?x benchmark : isMemberOf
http://www.Department0.University0.edu}

Description : Find out all students of http://www.Department0.University0.edu

Range(takeCourse, Student) , GraduateStudent _ _1 takeCourse _ GraduateStudent _ Student

[Query 4]

SELECT DISTINCT ?x

WHERE {?x rdf : type benchmark : Publication. ?x benchmark : publicationAuthor ?y.
?y rdf : type benchmark : Faculty. ?y benchmark : isMemberOf
http://www.Department0.University0.edu}

Description : All the publications by faculty of http://www.Department0.University0.edu

SubClass : Faculty = FullProfessor _ AssociateProfessor _…_ClericStaff, Publication = Article _
…_ Journal

[Query 5]
SELECT DISTINCT ?x
WHERE {?x rdf : type benchmark : ResearchGroup. ?x benchmark : subOrganizationOf http://www.University0.edu}
Description : All research groups of http://www.University0.edu
Transitive(subOrganizationOf), <a subOrganizationOf b>, <b subOrganizationOf http://www.University0.edu>
_ <a subOrganizationOf http://www.University0.edu>

[Query 6]
SELECT DISTINCT ?x
WHERE {?x rdf : type benchmark : Person. http://www.University0.edu benchmark : hasAlumnus ?x}
Description : All alumni of http://www.University0.edu
Inverse(hasAlumni, hasDegreeFrom), <a hasDegreeFrom b> _ <b hasAlumnus a>

[Query 7]
SELECT DISTINCT ?x
WHERE {?x rdf : type benchmark : Person. ?x benchmark : hasSameHomeTownWith http://www.Department0.University0.edu/FullProfessor0}
Description : Those who has same home town with http://www.Department0.University0.edu/FullProfessor0
Transitive(hasSameHomeTownWith), Symmetric(hasSameHomeTownWith), <a hasSameHomeTownWIth b>,
<c hasSameHomeTownWIth b> _ < a hasSameHomeTownWith c>

[Query 8]
SELECT DISTINCT ?x
WHERE {?x rdf : type benchmark : SportsLover. http://www.Department0.University0.edu benchmark : hasMember ?x}
Description : All sports lovers of http://www.Department0.University0.edu
<x like y>, <y rdf : type Sports>, SportLover__like.Sports _ <x rdf : type SportLover>
subProperty(isCrazyAbout, like), SportFan__isCrazyAbout.Sports_ SportFan _ SportLover

[Query 9]

SELECT DISTINCT ?x

WHERE {?x rdf : type benchmark : GraduateCourse. ?x benchmark : isTaughtBy ?y.

?y benchmark : isMemberOf ?z .?z benchmark : subOrganizationOf http://www.University0.edu }

Description : All Graduate Courses of http://www.University0.edu

GraduateStudent__takesCourse.GraduateCourse, <a rdf : type GraduateStudent>, <a takesCourse b> _ <b rdf : type GraduateCourse>

[Query 10]

SELECT DISTINCT ?x

WHERE {?x benchmark : isFriendOf http://www.Department0.University0.edu/FullProfessor0}

Description : All friends of http://www.Department0.University0.edu/FullProfessor0

Symmetric(isFriendOf), <a isFriendOf b> _ <b isFriendOf a>

[Query 11]

SELECT DISTINCT ?x

WHERE {?x rdf : type benchmark : Person. ?x benchmark : like ?y. ?z rdf : type benchmark : Chair.

?z benchmark : isHeadOf http://www.Department0.University0.edu. ?z benchmark : like ?y}

Description : All people who has same interest with the chair of

http://www.Department0.University0.edu

FunctionalProperty(isHeadOf), <a isHeadof b>, <c isHeadOf b) _ <a sameAs c> // there are some same individuals

of chair0

[Query 12]

SELECT DISTINCT ?x

WHERE {?x rdf : type benchmark : Student . ?x benchmark : takesCourse ?y

.?y benchmark : isTaughtBy http://www.Department0.University0.edu/FullProfessor0}

Description All students who take course taught by

http://www.Department0.University0.edu/FullProfessor0

GraduateStudent _ _takesCourse.GraduateCourse _ _1.takesCourse, Domain(takesCourse, Student) _ Student

_ GraduateStudent

[Query 13]

SELECT DISTINCT ?x

WHERE {?x rdf : type benchmark : PeopleWithHobby. ?x benchmark : isMemberOf http://www.Department0.University0.edu}

Description All people who has some kind of hobbies in http://www.Department0.University0.edu Lite Cardinality : PeopleWithHobby(_1like) _ SportLover, <a like b> _ <a rdf : type PeopleWithHobby>

[Query 14]

SELECT DISTINCT ?x

WHERE {?x rdf : type benchmark : Woman. ?x rdf : type benchmark : Student. ?x benchmark : isMemberOf ?y.

?y benchmark : subOrganizationOf http://www.University0.edu}

Description : All woman students of http://www. University0.edu

<a,isStudentof b>, <b rdf : type WomanCollege>, WomanCollege __hasStudent.(_Man), disjoint(Man,

Woman), Man_Woman _ Person _ <a rdf : type Woman>

[Query 15]

SELECT DISTINCT ?x

WHERE {?x rdf : type benchmark : PeopleWithManyHobbies. ?x benchmark : isMemberOf http://www.Department0.University0.edu}

Description : All people who has many hobbies in http://www. Department0.University0.edu PeopleWithManyHobbies__3like, <a like b1> ··· <a like bn>, all different(b1, b2···bn) _ <a rdf : type PeopleWithManyHobbies>

[Query 16]

SELECT DISTINCT ?person ?zone ?Hobby

WHERE

(?person benchmark : locatedIn <http://rcubs.kyunghee.ac.kr/owl/univ-bench-dl.owl#Zone1>)

(?person benchmark : locatedIn ?zone)

(?person benchmark : like ?Hobby)

〈부록 B〉: 각 추론 엔진에 대한 동적 테스트 결과

컨텍스트 데이터 수정 주기	쿼리 수행 주기	데이터 집합 1					데이터 집합 2				
		Minerva	DLDB OWL	Pellet	Jena	Selection	Minerva	DLDB OWL	Pellet	Jena	Selection
1200SEC	600sec	83.33%*	100.00%	83.33%	83.33%	100.00%	100.00%	100.00%	83.33%	83.33%	100.00%
		16.67%**	0.00%	16.67%	16.67%	0.00%	0.00%	0.00%	16.67%	16.67%	0.00%
		0.00%***	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	60sec	41.67%	95.00%	16.67%	93.33%	98.33%	91.67%	100.00%	60.00%	98.33%	86.67%
		58.33%	5.00%	48.33%	6.67%	1.67%	8.33%	0.00%	40.00%	1.67%	13.33%
		0.00%	0.00%	35.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	10sec	30.83%	94.72%	2.50%	90.28%	71.94%	87.50%	100.00%	10.83%	96.94%	100%
		69.17%	5.00%	13.06%	8.33%	28.06%	12.50%	0.00%	40.83%	3.06%	0.00%
		0.00%	0.28%	84.44%	1.39%	0.00%	0.00%	0.00%	48.33%	0.00%	0.00%
900SEC	600sec	66.67%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	83.33%
		33.33%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	16.67%
		0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	60sec	53.33%	91.67%	16.67%	95.00%	100.00%	86.67%	96.67%	55.00%	100.00%	90.00%
		46.67%	8.33%	30.00%	5.00%	0.00%	13.33%	3.33%	43.33%	0.00%	10.00%
		0.00%	0.00%	53.33%	0.00%	0.00%	0.00%	0.00%	1.67%	0.00%	0.00%
	10sec	52.50%	87.22%	2.22%	90.56%	75.83%	79.17%	95.00%	9.17%	95.83%	75.00%
		47.50%	12.78%	12.78%	8.61%	24.17%	11.94%	5.00%	42.50%	4.17%	25.00%
		0.00%	0.00%	85.00%	0.83%	0.00%	8.89%	0.00%	48.33%	0.00%	0.00%
600SEC	600sec	100.00%	100.00%	66.67%	66.67%	50.00%	100.00%	100.00%	66.67%	83.33%	100.00%
		0.00%	0.00%	33.33%	33.33%	50.00%	0.00%	0.00%	33.33%	16.67%	0.00%
		0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	60sec	41.67%	96.67%	5.00%	88.33%	81.67%	70.00%	96.67%	36.67%	98.33%	86.67%
		58.33%	3.33%	41.67%	11.67%	18.33%	28.33%	3.33%	61.67%	1.67%	13.33%
		0.00%	0.00%	53.33%	0.00%	0.00%	1.67%	0.00%	1.67%	0.00%	0.00%
	10sec	9.72%	87.22%	0.56%	84.17%	72.22%	66.94%	94.44%	6.94%	93.61%	99.72%
		90.28%	12.78%	14.44%	14.44%	27.78%	23.61%	5.56%	45.00%	6.39%	0.28%
		0.00%	0.00%	85.00%	1.39%	0.00%	9.44%	0.00%	48.06%	0.00%	0.00%

주) * : 성공율, ** : 실패율로서 오답을 낸 경우의 비율, *** : 실패율로서 작동을 멈춘 경우의 비율.

컨텍스트 데이터 수정 주기	쿼리 수행 주기	데이터 집합 3				
		Minerva	DLDB-OWL	Pellet	Jena	Selection
1200SEC	600sec	66.67%	100.00%	0.00%	100.00%	83.33%
		33.33%	0.00%	0.00%	0.00%	16.67%
		0.00%	0.00%	100.00%	0.00%	0.00%
	60sec	38.33%	88.33%	0.00%	85.00%	66.67%
		55.00%	11.67%	0.00%	15.00%	33.33%
		6.67%	0.00%	100.00%	0.00%	0.00%
	10sec	19.72%	86.67%	0.00%	82.78%	32.50%
		59.17%	13.33%	0.00%	17.22%	67.50%
		21.11%	0.00%	100.00%	0.00%	0.00%
900SEC	600sec	33.33%	66.67%	0.00%	100.00%	100.00%
		66.67%	33.33%	0.00%	0.00%	0.00%
		0.00%	0.00%	100.00%	0.00%	0.00%
	60sec	20.00%	76.67%	0.00%	86.67%	75.00%
		63.33%	23.33%	0.00%	13.33%	25.00%
		16.67%	0.00%	100.00%	0.00%	0.00%
	10sec	9.72%	81.11%	0.00%	84.44%	95.83%
		52.50%	18.89%	0.00%	15.56%	4.17%
		37.78%	0.00%	100.00%	0.00%	0.00%
600SEC	600sec	33.33%	100.00%	0.00%	83.33%	66.67%
		66.67%	0.00%	0.00%	16.67%	33.33%
		0.00%	0.00%	100.00%	0.00%	0.00%
	60sec	0.00%	55.00%	0.00%	73.33%	66.67%
		88.33%	45.00%	0.00%	26.67%	33.33%
		11.67%	0.00%	100.00%	0.00%	0.00%
	10sec	2.50%	68.61%	0.00%	71.39%	78.33%
		68.89%	31.39%	0.00%	28.61%	21.67%
		28.61%	0.00%	100.00%	0.00%	0.00%