Journal of the Korean Data & Information Science Society
2015, **26**(5), 1129–1139

http://dx.doi.org/10.7465/jkdi.2015.26.5.1129
한국데이터정보과학회지

동적 분산병렬 하둡시스템 및 분산추론기에 응용한 서버가상화 빅데이터 플랫폼[†]

송동호[1] · 신지애[2] · 인연진[3] · 이완곤[4] · 이강세[5]

[1]한국항공대학교 소프트웨어학과 · [23]소프트온넷 (주)· [45]숭실대학교 컴퓨터학부

접수 2015년 8월 17일, 수정 2015년 9월 21일, 게재확정 2015년 9월 25일

요 약

시멘틱 웹 기술인 RDF 트리플로 표현된 지식을 추론 과정을 거치면 새로운 트리플들이 생성되어 나온다. 초기 입력된 수억개의 트리플로 구성된 빅데이터와 추가로 생성된 트리플 데이터를 바탕으로 질의응답과 같은 다양한 응용시스템이 만들어 진다. 이 추론기가 수행되는 과정에서 더 많은 컴퓨팅 리소스가 필요해 진다. 이 추가 컴퓨팅 리소스는 하부 클라우드 컴퓨팅의 리소스 풀로부터 공급받아 수행시간을 줄일 수 있다. 본 연구에서는 하둡을 이용하는 환경에서 지식의 크기에 따라 런타임에 동적으로 서버 컴퓨팅 노드를 증감 시키는 방법을 연구하였다. 상부는 응용계층이며, 중간부는 트리플들에 대한 분산병렬추론과 하부는 탄력적 하둡시스템 및 가상화 서버로 구성되는 계층적 모델을 제시한다. 이 시스템의 알고리즘과 시험성능의 결과를 분석한다. 하둡 상에 기 개발된 풍부한 응용소프트웨어들은 이 탄력적 하둡 시스템 상에서 수정 없이 보다 빨리 수행될 수 있는 장점이 있다.

주요용어: 동적 하둡 시스템, 분산추론 플랫폼, 빅데이터 플랫폼, 서버 가상화.

1. 머리말

인터넷 환경에서 생성되는 수많은 데이터는 기존 정형 데이터와는 구분되는 비정형 데이터의 특성을 가지고 있으며, 이러한 비정형 데이터를 분석 활용하고자 하는 요구가 증가 하고 있다. 특히, 음성, 영상 등의 데이터는 비정형 데이터의 특성 뿐 만 아니라 데이터의 양도 엄청나게 많아서 이러한 빅데이터를 분석 활용 하려는 다양한 플랫폼이 등장하고 있다.

이러한 빅데이터를 분석하기 위한 방법의 하나로 하둡 (Hadoop)의 맵리듀서를 활용하는 방법은 일반적이나, 기존의 하둡은 빅데이터 처리에 필요한 동적 탄력성의 한계를 가지고 있다. 하둡은 하둡 분산 파일시스템 (Hadoop distributed file system; HDFS)과 데이터를 분산시켜 처리한 뒤 하나로 합치는 기술인 맵리듀스를 중심으로 구성되어 있다. 하지만 기존의 하둡 시스템에서 사용하는 맵리듀스는 호스트서버의 성능이 상이한 경우 효과적이지 못하고, 한 클러스터에서 여러 맵리듀스 작업을 동

[†] 이 논문은 2015년도 정부 (미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (NO. B0101-15-0047, 대용량 지식처리용 분산 병렬 추론 플랫폼 개발).
[1] 교신저자: (412-791) 경기도 고양시 덕양구 항공대학로 76, 한국항공대학교 소프트웨어학과, 교수. E-mail: dhsong@kau.ac.kr
[2] (135-080) 서울 특별시 강남구 언주로 542, 소프트온넷 (주), 이사.
[3] (135-080) 서울 특별시 강남구 언주로 542, 소프트온넷 (주), 이사.
[4] (156-743) 서울 특별시 동작구 상도로 369, 숭실대학교 컴퓨터학부, 박사과정.
[5] (156-743) 서울 특별시 동작구 상도로 369, 숭실대학교 컴퓨터학부, 박사과정.

시에 수행하는 경우에 대해서 효과적인 다중 작업 스케줄링을 제공하지 못하는 한계성도 가지고 있다 (http://hadoop.apache.org/docs/current/hadoop-yarn/ hadoop- yarn -site/YARN.html).

기존 하둡 시스템에서 사용하는 데이터 처리 방법은 처리를 시작할 때, 정적으로 할당된 고정된 컴퓨팅 자원 내에서 데이터를 처리한다. 그래서 데이터의 양이 런타임에 증가하는 특성을 가지고 있는 경우 추가로 컴퓨팅 자원을 더 끌어다 사용할 수 있는 방법이 필요하다. 즉, 기존 하둡은 데이터 분석을 시작하게 되면 가용 컴퓨팅 노드들이 있더라도 이 유휴 가용 컴퓨팅 자원을 런타임에 추가로 할당받아 사용하지 못하는 단점이 있다. 본 연구에서는 기존 표준 하둡의 이러한 런타임 비확장성을 해결하고자 한다. 실시간으로 런타임에 처리해야 할 데이터의 증감에 따른 필요한 컴퓨팅 자원을 먼저 분석하고, 그 결과에 따라 컴퓨팅 노드를 추가 혹은 감소하게 하여 응용프로그램이 필요로 하는 최적으로 사용할 수 있도록 해주는 탄력적 하둡모델을 제안한다.

"탄력적" 하둡 (elastic Hadoop)은 빅데이터 처리 플랫폼인 하둡 시스템을 개선하여 실시간에 가용 컴퓨팅 자원을 자동으로 확장 할 수 있는 시스템이다. 가용 컴퓨팅 자원이란 빅데이터 클라우드 시스템의 가상서버 (virtual machines) 들이다 (http://www.xenproject.org/). 따라서 필요한 맵리듀서 노드들과 매칭되는 가상서버 노드들을 예측하고 그 수요만큼 할당하는 오토스케일 (auto scaling) 분산병렬처리 방식을 의미한다. 즉, 하둡 시스템의 맵리듀서의 처리 과정에서 실시간으로 잡의 크기와 연산 시간을 분석하여 유휴 컴퓨팅 노드를 할당하는 것이다. 본 논문에서는 추론할 트리플들의 초기 개수, 런타임에 추론의 중간결과물로 생성되는 트리플 들의 개수 등 을 고려하여 컴퓨팅 노드 (자원)을 할당하는 방법에 대한 연구가 본 논문의 핵심이다 (Antoniou 등, 2012).

본 논문은 다음과 같이 구성된다. 2절에서 관련연구를 비교검토한 후 본 제안모델과의 차별성에 대해서 논의한다. 3절에서는 본 논문에서 제시하는 3개 층 (상부 응용프로그램 및 분산병렬 추론기 계층, 중간부 하둡 분산병렬 처리 계층, 하부 가상화 서버 계층)으로 구성된 시스템 모델에 대해서 설명하고, 4절과 5절에서는 구현된 시스템, 측정된 성능 및 이 결과에 대한 비교논의를 하고 6절에서 결론에 대해 논한다.

2. 관련연구

탄력적으로 런타임에 동적으로 컴퓨팅 노드를 확대 혹은 축소시키는 모델은 이전의 연구에서 몇 가지 사례를 찾을 수 있다. 가장 빈번히 활용되는 분야가 아마존 탄력적 EMR분야이다. 그리고 기존 맵리듀서가 v.2.0으로 발전하면서 본 연구와 유사한 방향으로 발전을 이루었고, 스파크와 관련된 연구와도 비교 분석하기로 한다. 또한. 유럽에서 연구된 하둡상의 잡의 크기에 기반한 스케쥴링 방법인 HFSP와 잡크기 예상평가하는 연구방향과 본 연구와의 차이점을 아래에서 제시한다.

2.1. 아마존 탄력적 맵리듀서 (아마존 EMR)

아마존 EMR은 빅 데이터 프로세싱을 간소화하여 동적으로 확장할 수 있는 아마존 EC2 인스턴스에 대량의 데이터를 쉽고, 빠르고, 비용 효율적으로 배포하고 처리할 수 있는 관리 하둡 프레임워크를 상용 웹서비스 형태로 제공하는 사례이다. 아마존 EMR을 사용하면 컴퓨팅 인스턴스를 한 개에서 수백 개 또는 심지어 수천 개까지 원하는 대로 프로비저닝해서 데이터를 처리할 수 있다. 인스턴스 수를 쉽게 늘리거나 줄일 수 있으며, 사용한 양에 대해서만 요금을 지불한다 (http://en.wikipedia.org/wiki/Apache_Hadoop#Amazon_Elastic_MapReduce).

아마존 EMR과 본 연구의 차이점은, EMR은 프로그램이 시작되는 시점에서 클러스터의 구성이 정적으로 완성되어 있어야 한다. EMR이 말하는 "탄력적"의 의미는 클러스터의 구성이 런타임 이전에 탄력

적으로 할 수 있다는 의미이지 런타임에 탄력적으로 노드 숫자가 증.감되는 것은 아니며, 이점이 본 연구와 가장 큰 차이점이다.

2.2. 맵리듀서 v.2.0 (YARN)

아파치 하둡의 멥리듀서 v.2.0 (YARN)은 기존 하둡을 완전히 재구성한 새 버전으로서 기본 아이디어는 기존의 잡 트래커의 두 기능을 분리 한 것에 있다. 즉, 리소스관리자와 잡스케쥴링/모니터링을 각각 독립적인 데몬 프로세스로 분리하였다. 이 아이디어 속에는 하나의 글로벌 자원관리자 (RM)와 응용프로그램별로 응용프로그램마스터 (AM)를 갖는 것이다. 하나의 응용프로그램은 기존 멥리듀서에서는 하나의 잡이든가 혹은 잡들의 한 DAG로 표현되었다 (Park, 2013; Song, 2015; Go와 Kim, 2013).

YARN과 본 연구의 차이점은 YARN 이 빅데이터의 분산 처리를 위해 다양한 기능을 제공하고 있지만, 본 연구에서는 빅데이터의 분산 처리를 위한 가상 컴퓨팅 플랫폼을 지원하는 기능을 제공하고 있으며, 특히 YARN에서 지원하고 있지 않은 실시간 오토 스케일링 기술을 제공 한다.

2.3. 스파크

스파크 (Spark)는 UC Berkeley의 AMP Lab.에서 초기 개발된 분산 클러스터 컴퓨팅 프레임워크로서, 맵리듀서 (map reducer)와 비교할 때 가장 큰 차이점은 프로그램과 데이터 모두 메모리 상주한다는 점이다.. 메모리 상주하기 때문에 응용프로그램이 적합할 경우 하둡의 맵리듀서보다 100배 빠른 성능을 보이기도 하는데 이는 디스크 I/O를 대폭 줄이고 데이터를 메모리에 상주시켜놓고 반복적인 쿼리 및 엑세스하기 때문에 이러한 성능이 가능할 수 있다. 스파크는 RDD (resilient distributed dataset)라는 읽기 전용 자료구조를 사용하며 RDD에서 다른 RDD로의 변환을 연쇄적으로 실시함으로써 연산이 이루어진다 (http://spark.apache.org/).

스파크와 본 연구의 차이점은 스파크는 단일 노드 상에 메모리에 데이터를 적재하여 수행하는 방식이므로 분산 병렬 처리가 어렵다. 향후 스파크 기술을 분산 병렬 처리하는 것에 연구가 필요할 것이다.

2.4. HFSP 와 기타 잡크기 예상평가 관련 연구

HFSP는 유럽의 FP7과제에서 연구된 하둡상의 잡의 크기에 기반한 스케쥴링 방법이다. 이 연구에서는 하둡에 다양한 수행시간을 가지는 잡들이 입력되었을 때 오래 수행되는 잡과 짧게 수행되는 잡 들에 대해서 우선순위를 조절해 가면서 남아있는 수행시간이 가장 짧는 잡에게 우선순위를 주는 알고리즘 (shortest remaining time first)과 유사한 방법으로 전체적인 스케쥴링의 효율을 올린 사례이다. HFSP 연구는 본 논문의 잡의 수행시간 예측 평가시스템에 도움을 받았으나, 기본적으로 본 논문이 주장하는 런타임에 탄력적 노드 할당정책을 사용하지는 않는 점에서는 큰 차이가 있다 (Pastorelli 등, 2013).

또한, 잡의 크기를 예상평가 하는 관련 연구로서는 반복되는 잡들 내에 있는 질의어 크기를 평가하는 기술적 접근방법이 제시되었다 (Verma 등, 2011; Agarwal 등, 2012). 이들 접근방법은 여전히 일반 OS에서 나타나는 스케쥴링과 유사한 방식으로 볼 수 있으며 본 논문에서 제시하는 탄력적 실시간 노드 할당과는 거리가 있다.

3. 시스템 모델 및 구현

RDF 트리플로 표현된 지식정보로 부터 추론을 하는 과정에서 트리플의 양이 런타임에 기하급수적으로 늘어나게 된다. 따라서, 시스템 리소스풀에서 추가로 사용할 수 있는 노드들이 있다면 동적으로 끌어다 사용할 수 있는 방법이 필요하다. 통상 추론기 성능측정 기준으로 사용되는 8,000 LUBM의 경우 약

18억개의 트리플로 표현된 데이터 세트가 이용된다. 기존 하둡 시스템에서 사용하는 데이터 분석 방법은 데이터 분석을 시작할 때, 그 데이터 분석에 "정적"으로 할당된 고정된 컴퓨팅 자원 내에서 데이터를 처리한다. 즉, 기존 하둡은 데이터 분석을 일단 시작하게 되면 유휴 컴퓨팅 자원이 있더라도 이 유휴 컴퓨팅 자원을 사용하지 못하는 단점이 있다. 표준 하둡의 이러한 비효율성을 개선하기 위해서, 본 연구에서는 실시간으로 런타임에 처리해야 할 데이터를 측정하여 필요한 만큼의 컴퓨팅 자원을 증감하게 하여 컴퓨팅 시간을 최적화 할 수 있도록 해 주는 "탄력적" 하둡 모델을 제안한다.

동적 분산병렬 하둡시스템과 이를 RDF 분산추론기에 응용한 서버가상화 플랫폼 시스템의 모델링은 아래 Figure 3.1과 같다. 이 시스템은 크게 3개 계층으로 구성이 되며 아래 문단에서 각각 차례로 설명한다.

3.1. 상부 응용프로그램 계층

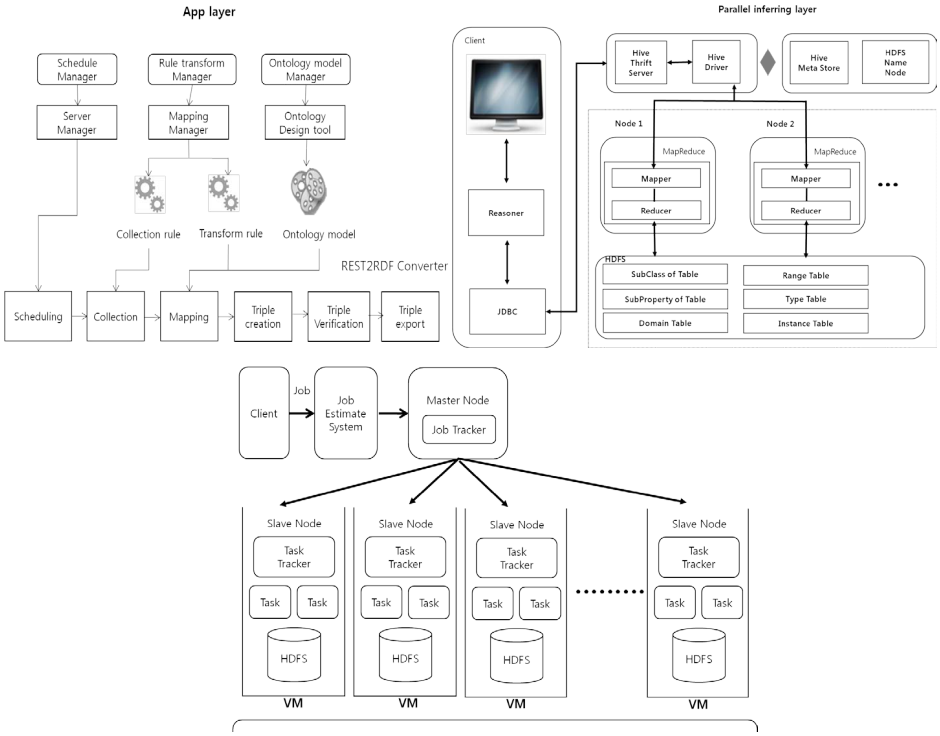

Figure 3.1 System model of three layers

본 논문에서 시험대상으로 삼은 상부 응용프로그램인 하이브는 분산 컴퓨팅 환경에서 질의문을 통해 대용량 데이터 처리와 분석을 위한 프레임워크로 개발되었다. 하이브의 장점은 개발자 또는 데이터 분석가들이 코드를 직접 작성할 필요 없이 RDBMS에서 사용하는 질의문을 통해 테이블 형태로 저장된 대용량 데이터에 대한 분산 처리를 수행할 수 있다. 마스터 노드의 하이브 드라이버는 클라이언트로부터 전달받은 질의문을 분석하여 최적화된 맵리듀스 전략을 자동으로 수립하고, 각 노드의 태스크 트래커에게 명령을 전달하는 과정을 통해 작업을 수행한다. 따라서 대용량 데이터 분석 및 처리를 위한 응용 프로그램 구현 및 관리가 용이하다 (Lee 등, 2014).

3.2. 분산병렬 추론기 계층

분산 병렬 추론기 계층에서는 6개의 RDFS 추론 규칙을 통해 새로운 트리플 데이터를 추론한다. 기존 트리플 데이터를 통해 분산 테이블을 생성하는 전처리 과정이 진행된 후에 실제 추론 작업을 수행한다. 전처리 과정에서는 HDFS에 저장된 입력데이터들의 술어 정보를 기반으로 클래스, 프로퍼티, 도메인, 레인지, 타입, 인스턴스에 대한 6개의 분산 테이블을 생성한다. 이후 클라이언트는 마스터 노드에 RDFS 추론 규칙에 대한 질의문을 전달하고, 마스터 노드는 해당 질의문을 분석하여 최적화된 맵리듀스 작업을 생성한다. 추론을 위해 분산 테이블에서 데이터를 필터링하는 작업은 맵 단계를 통해 수행되고, 필터링된 데이터들을 RDFS 규칙을 기반으로 조인하는 작업은 리듀스 단계를 통해 수행된다. 각 추론 규칙들은 여러 개의 맵리듀스 작업으로 구성된 하나의 잡이 되고, 분산 병렬 처리를 위한 각 노드들의 태스크 트래커들은 생성된 잡의 맵과 리듀스 작업을 동시에 수행하게 된다. LUBM 데이터셋을 대상으로 RDFS 추론 규칙을 적용하는 경우, 약 30% 정도의 트리플이 추가적으로 생성된다. 예를 들어 추론 규칙 중에서 가장 많은 트리플이 추론 되는 도메인 규칙의 경우 스키마 정보에 명시된 특정 프로퍼티의 주어가 속하는 클래스를 통해서 새로운 타입 정보를 추론한다.

Table 3.1 Domain inference algorithm

inst : instance table
domain : domain table
SELECT inst.subject, “rdf#type”, domain.object
JOIN domain.subject **AND** spo.predicate
INSERT result of join **INTO** type table
Remove Duplication

실제 LUBM의 스키마 정보에는 “<advisor> <rdf-schema#domain> <Person>”와 같은 도메인 정보가 존재한다. 이러한 도메인 정보를 통해서 해당하는 프로퍼티를 갖는 “<UndergraduateStudent01> <advisor> <AssistantProfessor3>”와 같은 인스턴스의 주어가 속하는 클래스를 추론할 수 있다. 추론 후 주어의 타입이 다음과 같이 “<UndergraduateStudent01> <rdf#type> <Person>”추론된다 (Lee 등, 2014).

3.3. 하부 하둡 분산병렬 처리 및 가상화 서버 계층

하부 계층은 두개의 서브 계층으로 구분되는데 상부의 오토스케일드 분산병렬 처리 하둡 시스템 부분과 하부의 가상화 서버 계층이다.

3.3.1. 오토스케일드 분산병렬 처리 하둡시스템 계층

오토스케일드 분산병렬 처리 하둡 시스템 서브 계층의 중심은 맵리듀서이며 하나의 잡을 세 단계로 분할하여 처리한다. 첫 단계를 맵 (map)이라고 하는데 하나의 사용자가 프로그래밍한 함수가 입력된 데이터 세트를 대상으로 중간 데이터세트를 만든 후 이를 로컬 파일시스템에 저장한다. 데이터 세트를 입력받아, 적절히 분할하며, 그 결과를 분산파일시스템에 저장한다. 이 로컬 파일시스템은 해당 클러스터의 개별 노드 혹은 머신에 있다. 이 중간 데이터세트들은 소팅되고 파티셔닝된 후 파일시스템에 저장된다. 이후 리듀스 (reduce) 단계가 시작되는데 셔플되는 서브단계에서는 이 중간 데이터세트들이 리듀스에 의해 다시 불리고 다중 메퍼들로 부터 데이터들이 소팅되고 통합되어 출력 데이터가 만들어 진다.

오토스케일드 분산병렬처리 하둡 시스템은 실시간 잡 크기 예측기와 하둡스케쥴러로 구성한다. 하둡 스케쥴러는 다시 잡을 재구조화 하는 기술, 잡 분배 처리 기술, 클라우드 연동 시스템 기술로 구성된다.

본 논문에서는 잡 크기 분석기에 집중해서 논의하기로 하며 나머지 기술들은 향후 논문에서 다루고자 한다.

실시간 잡 크기 예측기

하둡이 수행할 잡의 크기를 분석하는 잡 크기 예측 시스템 (job estimate system)은 어떤 응용프로그램이 얼마나 오랫동안 수행될 것인가에 대한 가까운 미래치를 예측하는 것이다. 수행시간에 대한 정확한 미래 예측은 컴퓨터 운영체제 스케쥴링 연구 분야에서 다양한 방식으로 오랫동안 연구되어 왔으나 현실적으로 채택될 만한 알고리즘 사례는 아직 없다 (Agarwal 등, 2012).

본 연구에서는 처리할 트리플의 크기를 중요한 요소로 평가하여 필요한 컴퓨팅 노드의 개수를 할당하는 단순한 방식을 사용한다. 분석에 사용되는 리소스는 잡의 데이터 사이즈, CPU 사용량, 네트워크 데이터 사용량을 분석하여 잡의 처리 시간을 분석한다. 이런 방식이 채택된 이유는, 트리플들이 입력되고 추론되는 과정에서 트리플의 개수 비례로 추론 시간이 소요되는 단순관계는 아닐 수 있으나 통계적으로 보면 대부분 트리플의 개수에 비례하는 경향을 보이는 것은 사실이다. 또한 컴퓨팅 자원 관점에서는 하나의 컴퓨팅 노드를 CPU의 컴퓨팅파워, 네트워크 대역폭등을 고려하여 단위화 한 것을 사용한다.

다음과 같은 표기를 사용하여 알고리즘을 제시하고자 한다. 하나의 잡은 다양한 크기의 n개의 트리플 그룹으로 나누어진다.

- $\sum_{i=1}^{n} (S_i * M_i * N_i)$: 하나의 잡의 총 트리플 개수
- S_i : 한 그룹에 속한 트리플의 개수가 어떤 정해진 범위에 속한 데이터 그룹 의 개수
- M_i : S_i 그룹에 속한 트리플의 개수
- N_i : S_i로 표현된 그룹의 개수

하나의 클라우드 풀 내에는 다양한 크기의 n개의 컴퓨터 서버 노드 그룹으로 나누어진다.

- $\sum_{i=1}^{n} (W_i * X_i * Y_i)$: 하나의 클라우드 풀내의 총 노드 개수
- W_i : 한 클라우드 풀에 속한 노드가 어떤 정해진 범위에 속한 노드 그룹 의 개수
- X_i : W_i 그룹에 속한 노드의 개수
- Y_i : W_i로 표현된 그룹의 개수

여기서, 잡의 크기가 런이 정적으로 고정되어 시작이 되지만, 추론이 시작되면 트리플 데이터가 증가하게 되는데, 이 증가하는 트리플의 개수를 n' 라고 하자. 이 n'은 런이 진행됨에 따라 거의 선형적으로 증가된다. 이렇게 추가된 트리플로 인하여 초기 속했던 S_i그룹에서 승급될 수 도 있다.

클라우드 풀 내의 여타 컴퓨팅 노드들은 다른 응용프로그램을 수행시키는데 할당되었다가 프리가 될 수 있다. 그 노드들은 이 잡이 수행되는데 가용자원으로 활용될 수 있기 때문에 런타임에 지속적으로 추가 공급될 수 도 있다. 여기에 공급되는 컴퓨팅 노드는 고성능 노드에서 부터 저성능 노드까지 다양하다. 따라서 이들 컴퓨팅 노드를 등급화 하는 것은 타당하다고 본다.

이러한 컴퓨팅 노드와 트리플의 크기들을 표기하기 위하여 Dijkstra의 Banker 알고리즘과 유사한 표기법을 사용하기로 하자. 잡을 크기에 따라 $S_n = [S_1, S_2, \cdots, S_n]$ 단위로 구분한다. 컴퓨팅 노드는 자원으로서 $W_n = [W_1, W_2, \cdots, W_n]$으로 구분한다. 가용자원을 $A_n = [A_1, A_2, \cdots, A_n]$으로 표기한다. 총 잡은 현재 할당된 자원(current allocation)으로 $C_n = [C_1, C_2, \cdots, C_n]$, 추가로 요청하는 노드 $R_n = [R_1, R_2, \cdots, R_n]$로 표시 한다

예를 들면, 트리플이 크기에 따라 S_n=[1천만개 이하, 1천만개~5천만개이하, 5천만개~2억개 이하, 2억개~5억개 이하, 5억개~10억개 이하]로 하고, 컴퓨팅 노드인 W_i = [극소형, 소형, 중형, 대형, 특대형]로 구분하자.

Table 3.2 Ex: server node and triple vectors

Existing Resources	W_n	=	[2, 0, 1, 0, 1]
Available Resources	A_n	=	[1, 0, 1, 0, 0]
Current Allocation	C_n	=	[1, 0, 0, 0, 1]
Request	R_n	=	[1, 0, 1, 0, 0]
Triples	S_n	=	[2, 0, 1, 0, 1]

즉, 천만 개 이하가 2개, 5천만 개∼2억 개 이하가 1개, 5억 개∼10억 개가 1개 있다는 뜻이다. 초기 상태로 천만 개 이하 트리플 데이터 2개는 현재 할당된 자원을 보면 극소형 컴퓨팅 노드 1개가 할당되었고, 추가로 1개 더 요청하고 있는 상태이며, 5천만∼2억 이하가 1개 있는데, 이것은 중형 컴퓨팅 노드가 1개 필요하나 현재 할당을 못 받고 있어 대기하고 있는 상태이며, 5억∼10억 데이터가 1개 있는데, 이는 초대형 컴퓨팅 노드가 하나 할당되었다는 뜻이다.

트리플의 크기와 이 트리플을 분산병렬 처리하는데 필요한 컴퓨팅 노드의 크기에 따른 매핑을 하는 스케쥴링 알고리즘은 다음과 같다.

서버 노드 스케쥴링은 각 $S_i[\,]$에 대해서 $C_i[\,]$에 $A_i[\,]$로 부터 현재 가용한 모든 자원을 할당 받는다. 이렇게 할당을 받은 후에도 여전히 부족한 자원은 있을 수 있는데, 그 자원을 $R_i[\,]$ 에 기록하여 둔다. 이렇게 하면, 트리플 크기 그룹에 따라 필요한 컴퓨팅 노드를 할당 할 수 있는 만큼 한 후 런을 시작할 수 있다. 수행 중 부족한 노드 자원에 관한 요청 정보는 $R_i[\,]$에 남아 있으므로 $A_i[\,]$가 추가로 생길 때 재스케쥴링을 하게 된다. 이렇게 추론이 시작이 되면 앞서 언급한 바와 같이 추론의 결과물로서 트리플들이 추가로 생성이 되는데, 이 생성되는 트리플들도 각 트리플 그룹에 추가가 되고, 이 추가된 트리플들에 대해서 기존 컴퓨팅 노드가 수행시간을 늘려서 처리 할 수 도 있고 아니면 별도의 노드가 할당이 될 수 도 있다.

여기서, 트리플 데이터가 각 컴퓨팅 노드에서 위의 알고리즘에 따라 추론을 하면, 추론된 데이터가 그 노드의 스토리지에 저장되는 로컬리티 (지역성)가 있다는 점이다. 이 데이터 로컬리티는 스케쥴링 함에 있어 가능한 지켜지도록 고려한다. 그 이유는 데이터 로컬리티와 컴퓨팅 로컬리티가 일치하지 않을 경우 이를 일치시키기 위해 데이터 혹은 컴퓨팅 마이그래이션이 일어나야 하는데 이것은 오버헤드로 작용하기 때문에 피하고자 하는 것이다.

하둡 스케쥴러

하둡에서 잡트래커는 테스크 트래커란 워커머신들을 관리한다 (Figure 3.1). 상기 실시간 잡크기 분석기에서 $C_i[\,]$는 테스크 트래커 노드의 성능별 차이를 고려한 자원 할당 방식이다. $C_i[\,]$에 정의된 노드들이 이제 하부 클라우드내 가상머신 풀에서 할당된다.

클라우드 시스템은 대용량 데이터 처리를 위한 클라우드 관리 시스템과 오토 스케일-업, 스케일-다운 시스템으로 구성한다. 클라우드 시스템에서 사용 중인 가상서버 자원에 대한 리소스 사용량을 추적하여, 가용 자원이 발생할 경우 A_n에 추가로 자원 할당 할 수 있도록 준비한다. 또한, 가상머신 노드가 프리가 되어 가용상태로 전환되면 스케쥴링을 상기 콜백 스케쥴링 알고리즘에 따라 호출한다. 하나의 추론 잡이 클러스터에 들어오면 스케쥴러는 여러 개의 맵 테스크들을 입력 데이터의 파티션 수에 따라 할당을 한다. 이는 데이터의 노드에 위치한 로컬리티를 고려한 스케쥴링 정책이기도 하다. 본 연구에서는 하나의 거대한 추론 잡이 들어왔을 때 추가 노드 할당을 고려한 스케쥴링을 하는 것이다. 이러한 추론 잡이 동시에 병렬로 여러 개 들어오는 경우는 본 논문에서는 고려되고 있지 않다.

하둡의 하부 시스템은 호스트 서버들이 가상화된 형태의 서버가상화 된 가상머신 (VM)으로 제공된다. 하둡 분산병렬 처리 계층에서 컴퓨팅 자원에 대한 리소스 할당 요구 사항이 있을 때, 하부 가상화 서버 계층에서는 유휴 컴퓨팅 자원을 가상 머신 형태로 제공한다.

3.3.2. 가상화 서버 계층

가상화 서버 계층은 Xen 기반 하이퍼바이져로 구성되어 있다. 가상화 서버 계층에서는 가상머신에 대한 관리 기능을 수행한다. 가상머신 관리 기능은 가상머신의 할당, 삭제, 수정 등의 기능을 수행하고, 할당된 가상머신의 컴퓨팅 자원 활용률에 대한 모니터링을 통해 가상머신의 사용현황을 파악한다.

가상머신의 컴퓨팅 자원 활용에 대한 모니터링을 통해 가용한 컴퓨팅 자원이 발생하면 가상머신을 추가로 할당하여 가상머신을 할당 할 수 있는 상태인 할당 대기 상태로 전환하여, 오토스케일드 분산병렬처리 하둡시스템 계층에서 가상머신에 대한 요청이 있을 때 가용한 가상 머신을 할당한다.

4. 성능측정 및 비교

본 연구에서 수행한 실험환경에 대한 상세한 설명을 하면 다음과 같다. 실험에는 하나의 클러스터가 한 개의 마스터 노드와 4개의 슬레이브 노드로 구성된다. 하나의 마스터 노드는 4개의 vCPU와 40GB RAM, 슬레이브 노드는 4개의 vCPU와 34GB RAM으로 구성된다.

이 클러스터 상에 하둡을 가장 좋은 구성비로 설치를 하였다. HDFS 블록크기는 128MB와 3개의 복제본으로 설정하였다. 각 테스크 트래커는 2개의 매퍼와 1 리듀서로 구성이 된다. 이 클러스터의 전체로 보면 10개의 매퍼와 5개의 리듀서를 가진다.

본 실험에서 사용한 플렛폼은 위의 Figure 4.1에서 3개의 플랫폼: 표준하둡분산병렬플랫폼, 오토스케일드 분산병렬플렛폼, 스파크 플랫폼, 을 생각할 수 있다. 이중 표준하둡 분산병렬플렛폼은 직접 실험을 할 필요는 없기에 성능측정은 생략하였다. 따라서 두 개의 플렛폼을 이용하여 동일 응용프로그램, 동일 분산병렬 추론기, 동일 호스트 서버 하드웨어 장비를 사용하였으며 성능 데이터를 측정하였다.

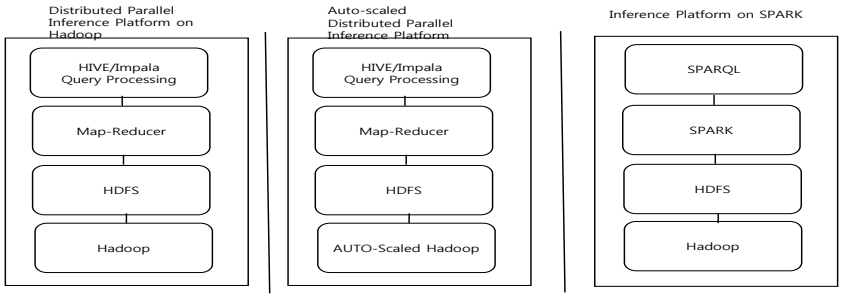

Figure 4.1 Comparison of architecture: Standard hadoop, autoscaled hadoop, Spark platform

위의 Figure 4.1에서 좌측은 표준 하둡상의 분산병렬 추론 플렛폼 모델이며 런타임에 노드의 증감이 없다. 중앙은 본 연구에서 제안한 오토스케일드 분산병렬플렛폼 모델이며, 우측은 최근 크게 확산되고 있는 스파크 플랫폼이다. 하단의 Table 4.1을 보면, HIVE를 이용하는 방식으로 호스트 서버 노드 개수를 1개와 5개로 변화시켜 성능측정을 하였다. 스파크를 이용하는 방식에서도 동일한 서버 노드를 적용하여 성능측정을 하였다. Table 4.1은 측정된 성능 결과를 보여주고 있다.

Table 4.1 Performance: Auto-scaled MR vs. SPARK platform

Test cases					Inferred time (Single node)	Inferred time (5 nodes)
Ontology	Platform	Number of triples (Million)	Data size (GB)	Number of inferred triples (Million)	Elapsed time (Min)	Elapsed time (Min)
LUBM100	SPARK	13	1.8	5.1	3.15	1.1
	Auto-scaled MR				12.1	11.2
LUBM500	SPARK	65	8.7	25.6	12.7	3.5
	Auto-scaled MR				33.7	28.3
LUBM1000	SPARK	133	17.9	54	NA	7.4
	Auto-scaled MR				61.7	49.6

두 개의 플렛폼에 대해서 각각 LUBM100, LUBM500, LUBM1,000을 입력데이타로 활용한 분산병렬추론기를 수행했을때의 시간을 분단위로 측정한 데이터이다. LUBM100일 경우 1,300만개의 트리플 입력 데이터에 대해서 그 저장용량이 1.8GB이며, 추론과정에서 추가로 생성된 트리플들이 510만 개다. 추론 시간이 단일 노드에서, 즉, 마스터노드와 슬레이브 노드가 각각 1VM씩 있는 경우에는 12.1.분이 소요되었으며, 5노드, 즉, 마스터노드1개와 슬레이브 노드 4개, 일때는 11.2분이 소요되었다. 단, 마스터 노드에도 테스크트래커 워크가 수행되기 때문에 전체 워크는 5개이다. 이 경우, 트리플 데이터의 크기가 충분히 크지 않기 때문에 노드수가 증가하더라도 성능에 큰 차이는 없다. 이는 데이터를 복제하는데 시간이 소요되기 때문이라고 판단된다. LUBM1,000인 경우 오토스케일드 하둡인 경우 단일노드인 경우 61.7분이 소요되었으며 5노드인 경우 49.6분이 소요되었다. 이는 약 20%의 성능 개선을 의미한다. LUBM500인 경우 동일한 환경에서 16.1%의 개선효과를 보여준다.

자동 분산병렬 처리 하둡시스템을 지원하는 가상 클라우드 시스템을 적용하여 트리플 개수가 4.13억 정도 되는 크기의 빅데이터 LUBM3000을 적용하여 처리한 결과 표준 하둡 플렛폼 대비 약 3.5배 빠른 처리 성능을 보였다. 이 LUBM3,000의 경우는 스파크에서 필요로 하는 메모리 용량이 워낙 커서 실험을 이번기회에 해 볼 수 없었던 것이 아쉽다 (Lee 등, 2014; Lee와 Park, 2015; Zaharia 등, 2012; Song, 2015).

5. 결론

온톨로지 추론과정은 초기 입력된 수억개의 트리플로 구성된 빅데이터와 추가로 생성된 20% 이상의 트리플 데이터를 처리하는 데이터 프로세싱이다. 분산 병렬 추론기는 수행 과정중에 보다 많은 컴퓨팅 리소스가 필요해 진다. 이 컴퓨팅 리소스는 탄력적 하둡 맵리듀서의 필요에 의해 하부 클라우드 컴퓨팅의 리소스 풀로부터 런타임에 추가로 공급받아 수행시간을 줄일 수 있다. 본 연구에서는 지식을 크기에 따라 카테고리화 하고 또한 서버 컴퓨팅 노드도 성능별로 카테고리화 한 후 서로 매핑하는 과정을 모델로 제시하고 그 알고리즘을 구현하였다.

성능 시험 결과를 보면 LUBM100처럼 작은수의 트리플에 대해서 성능 개선효과는 미미하나, LUBM 500, 즉, 6,500만개의 입력 트리플의 경우 성능이 16%부터 LUBM1,000 일 경우 성능개선이 20% 개선이 되며, 본 논문의 범위 밖이지만 연구실에서 수행한 타 연구결과와 비교해 보면 LUBM3,000 인 경우 300% 이상 개선됨을 알수 있었다 (Lee 등, 2014).

이 모델은 약 2년 전 부터 본격 개발되어 확산되고 있는 스파크와 비교해 볼 때 대부분의 경우에 성능이 빠르지는 않으나 다음과 같은 장점을 가지고 있다. 첫째, 탄력적 하둡은 표준 하둡의 인터페이스를 그대로 이용하므로 호환성을 제공한다. 즉, 하둡기반 기존 추론 알고리즘을 수정없이 그대로 수행할 수 있다는 장점이 있다. 스파크의 경우 SPARKQL방식으로 쿼리를 처리해야 하나 하둡에서는 Hive 방식으로 처리할 수 있다. 둘째, 분산 데이터를 처리함에 있어 초기 컴퓨팅 리소스를 할당하여 처리하는 런타임 중에 유휴 컴퓨팅 자원이 다른 응용프로그램의 종료로 인해서 생길 시 이 가용 컴퓨팅 자원을 할당하여 처리할 수 있다. 셋째, 컴퓨팅 자원이 빈약하여 데이터 처리 시간이 오래 걸리는 잡을 컴퓨팅 자원이 풍부한 노드로 이전하여 처리함으로써 전체 처리 시간을 줄일 수 있다. 탄력적 하둡의 이러한 강점으로 인하여 동적 분산병렬 하둡 시스템은 경제성과 효용성측면에서 전망이 밝다고 본다.

본 연구에서는 하나의 잡에 대해서만 스케쥴링 하는 알고리즘을 제시하였으나, 현재 이를 확장하여 다수 잡에 대해서 스케쥴링을 확장하는 연구가 진행 중에 있다. 이 확장 모델은 보다 일반화된 알고리즘으로서 상용시스템으로 적용할 것을 고려하고 있다.

References

Agarwal, S., Kandula, S., Bruno, N., Wu, M.C., Stoica, I. and Zhou, J. (2012). Re-optimizing data-parallel computing. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation*, San Jose, USA.

Antoniou, G., Groth, P., Van Harmelen, F. and Hoekstra, R. (2012). *A semantic web primer*, 3rd Ed., The MIT Press, Cambridge, Massachusetts, London, England.

Go, Y. and Kim, J. (2013). Bigdata processing and analysis using rhipe. *Journal of the Korean Data & Information Science Society*, **24**, 975-987.

Lee, W. G., Kim, J. M. and Park, Y. T. (2014). (2014). Distributed table join for scalable RDFS reasoning on cloud computing environment. *Journal of KIISE*, **41**, 674-685.

Lee, W. G and Park, Y. T. (2015). ABox realization reasoning in distributed in-memory system. *Journal of KIISE*, **42**, 852-859.

Park, J., Lee S., Kang, D. and Won, J. (2013). Hadoop and Mapreduce. *Journal of the Korean Data & Information Science Society*, **24**, 1013-1027.

Pastorelli, M., Barbuzzi, A., Carra, D., Dell'Amico, M. and Michiardi, P. (2013). HFSP: Size-based scheduling for Hadoop. In *Proceedings of IEEE International Conference on Big Data. Silicon Valley*, CA, USA.

Song, D. (2015). *Annual report on distributed parallel infernce platform for large scale knowledge processing*, 13-912-03-005, IITP, Korea.

Verma, A., Cherkasova, L. and Campbell, R. H. (2011). Aria: Automatic resource inference and allocation for MapReduce environments. In *Proceedings of International Conference on Automation and computing*, Huddersfield, United Kingdom.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker D. and Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, USENIX Association Berkeley*, CA, USA.

Journal of the Korean Data & Information Science Society
2015, **26**(5), 1129–1139

http://dx.doi.org/10.7465/jkdi.2015.26.5.1129
한국데이터정보과학회지

An elastic distributed parallel Hadoop system for bigdata platform and distributed inference engines[†]

Dong Ho Song[1] · Ji Ae Shin[2] · Yean Jin In[3] · Wan Gon Lee[4] · Kang Se Lee[5]

[1]Software University, Korea Aerospace University
[23]SoftOnNet
[45]Computer Science Department, Sungsil University

Received 17 August 2015, revised 21 September 2015, accepted 25 September 2015

Abstract

Inference process generates additional triples from knowledge represented in RDF triples of semantic web technology. Tens of million of triples as an initial big data and the additionally inferred triples become a knowledge base for applications such as QA(question&answer) system. The inference engine requires more computing resources to process the triples generated while inferencing. The additional computing resources supplied by underlying resource pool in cloud computing can shorten the execution time. This paper addresses an algorithm to allocate the number of computing nodes "elastically" at runtime on Hadoop, depending on the size of knowledge data fed. The model proposed in this paper is composed of the layered architecture: the top layer for applications, the middle layer for distributed parallel inference engine to process the triples, and lower layer for elastic Hadoop and server visualization. System algorithms and test data are analyzed and discussed in this paper. The model hast the benefit that rich legacy Hadoop applications can be run faster on this system without any modification.

Keywords: Bigdata platform, distributed parallel inference engine, elastic Hadoop system, server virtualization.

[†] This work was partially supported by the Ministry of Science, ICT and Future Planning in (NO. B0101-15-0047, A Distributed Parallel Inference Platform for Bigdata Knowledge Processing).
[1] Corresponding author: Professor, Software University, Korea Aerospace University, Kyunggi-Do 412-791, Korea. E-mail: dhsong@kau.ac.kr
[2] Executive director, SoftOnNet, Kangnam-Gu, Eunjoo-Ro 542, Seoul 135-080, Korea.
[3] Executive director, SoftOnNet, Kangnam-Gu, Eunjoo-Ro 542, Seoul 135-080, Korea.
[4] Ph.D. student, Computer Science Department, Sungsil University, Seoul 156-743, Korea.
[5] Ph.D. student, Computer Science Department, Sungsil University, Seoul 156-743, Korea.