병렬 정보검색 시스템의 순차적인 검색엔진 알고리즘의 병렬화를 위한 연구

김석영*, 박미영*, 박혁로*, 정인상**
*전남대학교 전자컴퓨터공학부
**한성대학교 컴퓨터공학과
naquartz@moiza.chonnam.ac.kr;openmp@computer.org;
hyukro@chonnam.ac.kr;insang@hansung.ac.kr

A Study for Parallelizing Sequential Algorithms of Search Engine in Parallel Information Retrieval System

Seok Young Kim*, Mi-Young Park*, Hyuk-Ro Park*, In Sang Chung**
*School of Electronics & Commputer Engineering, Chonnam National University
**Department of Computer Engineering, Hansung University

요 약

대규모 데이터를 효율적으로 검색하기 위한 병렬 정보검색 시스템에서는 하드웨어 확장으로 인한 병렬화로 시스템 전체의 작업 처리량을 증가시켰다. 그러나 병렬 시스템 상에서 수행되는 검색엔진의 알고리즘들은 여전히 순차적으로 수행되기 때문에, 사용자의 개별적인 질의처리 시간은 단축되지 않는다. 본 연구는 검색엔진의 병렬화를 위하여 사용자 질의처리 과정과 역색인 파일처리 과정의 순차 알고리즘들을 조사하여 병렬화의 필요성과 가능성을 평가한다. 이러한 평가는 병렬 정보검색 시스템에서 수행되는 순차 알고리즘들의 효과적이고 체계적인 병렬화를 도모하고, 보다 효율적인 병렬 정보검색 시스템의 구축을 가능하게 한다.

1. 서론

오늘날 인터넷이 대중화되고 데이터의 양이 기하급수적으로 증가함에 따라 단일 정보검색 시스템[2, 3]은 그 한계를 드러내고 있다. 문서 컬렉션이 증가함에 따라 정보검색 시스템의 검색과 색인의 비용이 증가하여, 정보검색 시스템의 성능이 점차적으로 감소한다.

문서의 대폭적인 증가와 관련된 이러한 문제는 성능 및 확장 측면에서 병렬 및 분산처리 기술을 이용한 정보검색 시스템으로 해결될 수 있다. 실제로 구글(Google), 야후(Yahoo), 알타비스타(AltaVista)와 같은 웹 검색 시스템은 대 규모 데이터에 대한 효율적인 검색을 지원하기 위해 병렬 정보검색 시스템[1, 4, 5]을 사용하고 있다.

병렬 정보검색 시스템을 구축하는 보편적인 방법은 확장성과 비용을 고려하여 고성능의 단일 병렬 컴퓨터보다는 비교적 가격이 저렴한 여러 대의 컴퓨터들을 하나의 클러스터로 구축하는 것이다. 이러한 병렬 정보검색 시스템에서는 각 노드가 독립적인 검색엔진을 실행하여 하나의 질의를 처리한다. 결과적으로 클러스터를 구성하는 여러 개의 노드들이 동시에 각각의 질의들을 병렬적으로 처리하기 때문에 시스템 전체의 작업처리량은 증가한다. 그러나 사용자의 개별적인 질의를 처리하는 시간은 여전히 감소하지 않는다.

개별적인 질의처리 시간을 감소시키기 위한 기존 연구[1]는 색인 및 문서 컬렉션을 논리적 또는 물리적으로 분할하여 각 프로세서에 할당하여 질의를 처리한다. 이때 효율적인 병렬 처리를 위해서는 각 프로세서들 간에 색인 및 문서들을 균등하게 분할하는 것이 매우 중요하다. 그러나 이러한 균등 분할은 끊임없이 변화하고 증가하는 문서 컬렉션의 특성으로 인해, 시스템의 유지 및 관리 측면에서 많은 비용을 초래한다.

현존하는 병렬 정보검색 시스템에서 검색엔진 그 자체는 여전히 순차 프로그램이기 때문에 사용자의 개별적인 질의처리 시간은 단축되지 않고 있다. 검색엔진을 구성하는 순차 알고리즘들은 병렬 시스템의 구조에 따라 OpenMP[6]나 MPI[7]를 이용하여 병렬화를 할 수 있다. 이러한 병렬화를 통해서 색인 및 문서 컬렉션을 분할하지 않고 개별적인 질의처리 시간을 단축시킬 수 있다.

따라서 본 연구에서는 보다 효율적인 병렬 정보검색 시스템의 구축을 위하여 검색엔진을 구성하는 알고리즘들에 대한 병렬화의 필요성과 가능성을 평가한다. 이를 위하여 정보검색 시스템의 사용자 질의처리 과정과 역색인 파일처리 과정에서 수행되는 순차 알고리즘들을 조사한다. 그리고 순차 알고리즘의 병렬화를 위한 고려사항을 제시하고, 검색엔진의 순차 알고리즘에 대해서 병렬화의 가능성을 평가한다.

이러한 평가는 병렬 정보검색 시스템에서 수행되는 순차 알고리즘의 효과적이고 체계적인 병렬화를 도모한다.

또한 색인 및 문서 컬렉션을 균등하게 분할하지 않고 개별적인 질의처리 시간을 단축할 수 있는 병렬 정보검색 시스템의 구축을 가능하게 한다.

이어지는 2절에서는 현존하는 병렬 정보검색 시스템에 대해서 소개하고 문제점을 짚어본다. 3절에서는 사용자 질의 처리와 역색인 파일처리 과정에서 수행되는 알고리즘들을 조사한다. 그리고 4절에서는 순차 알고리즘의 병렬화를 위한 고려사항을 설명하고, 이를 기준으로 검색엔진의 순차 알고리즘에 대한 병렬화 가능성을 평가한다. 마지막으로 5절에서는 향후 연구 계획과 함께 결론을 내린다.

2. 병렬 정보검색 시스템

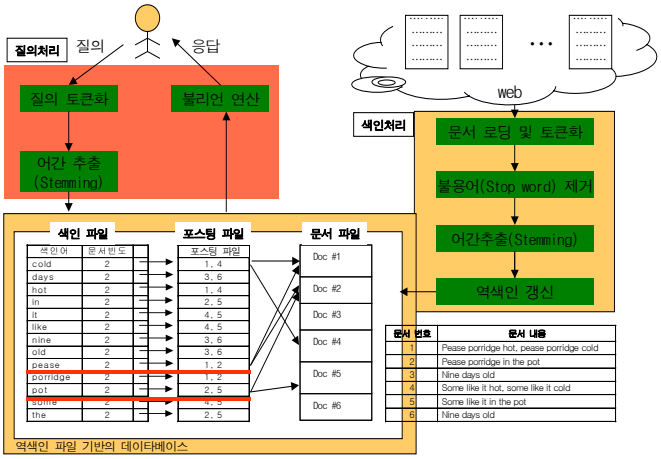
인터넷이 대중화되고 데이터의 양이 기하급수적으로 증가함에 따라 효율적인 검색을 위한 병렬 정보검색 시스템[1, 4, 5]의 필요성이 부각되고 있다. 병렬 정보검색 시스템을 구축하는 보편적인 방법은 확장성과 비용을 고려하여 고성능의 단일 병렬 컴퓨터보다는 비교적 가격이 저렴한 여러 대의 컴퓨터들을 하나의 클러스터로 구축하는 것이다.

클러스터 기반의 병렬 정보검색 시스템의 병렬화는 크게 세 가지 유형으로 이루어진다. 첫째는 클러스터를 구성하는 각 노드가 독립적인 검색엔진을 실행하여 하나의 질의를 처리하는 것이다. 또는 사용자의 질의를 분할하여 각 노드가 질의의 일부를 처리하고, 마지막에 그 결과를 병합하여 사용자에게 보여주는 것이다. 이 유형에서는 클러스터를 구성하는 여러 개의 노드들이 각각의 질의들을 병행적으로 처리하기 때문에, 시스템 전체의 작업처리량은 증가한다. 한국과학기술원 첨단정보기술연구센터에서 개발한 병렬 정보검색 시스템이 이 유형에 속한다[4].

두 번째는 사용자의 개별적인 질의 처리시간을 감소시키기 위한 유형으로써, 색인 및 문서 컬렉션을 논리적 또는 물리적으로 각 프로세서에 균등하게 분할하여 질의를 처리한다. 이때 효율적인 병렬 처리를 위해서는 각 프로세서들 간에 색인 및 문서들의 균등 분할이 매우 중요하다. 부산대학교에서 제시한 병렬 정보검색 시스템이 이 유형에 속한다[1].

마지막으로 세 번째는 위의 두 유형이 결합된 형태로써 현재 Google이 세 번째 유형을 택하고 있다[5]. 세 번째 유형에서는 단일 컴퓨터 시스템에서 병렬 컴퓨터 시스템으로 확장함으로써 시스템 전체의 작업처리량을 증가시키고, 색인 및 문서를 각 노드에 균등하게 분할함으로써 개별적인 질의처리 시간도 단축시켰다.

앞서 소개된 세 가지 유형에서 병렬화가 최대화된 시스템은 Google이 택한 세 번째 유형이다. 그러나 현존하는 병렬 정보검색 시스템에서 수행되고 있는 검색엔진 그 자체는 여전히 순차 프로그램이다. 이는 병렬 정보검색 시스템에서 개별적인 질의처리 시간이 감소하지 않은 주된 원인이기도 하다. 만약에 검색엔진의 알고리즘들이 병렬 시스템에 적합하게 병렬 프로그램으로 재 작성 된다면, 색인 및 문서 컬렉션을 분할할 필요도 없이, 개별적인 질의처리 시간을 단축시킬 수 있다.



(그림 1) 불리언 정보검색 시스템

3. 불리언 정보검색 시스템

본 절에서는 순차 알고리즘의 병렬화에 앞서 정보검색 시스템에서 수행되는 주요한 알고리즘을 먼저 살펴본다. (그림 1)은 전형적인 불리언 정보검색 시스템[2, 3]의 일반적인 수행과정을 보여준다. 그림에서와 같이 정보검색 시스템의 운영은 크게 두 가지로 구분되는데, 하나는 사용자 질의처리 부분이고 다른 하나는 역색인 파일처리 부분이다.

사용자 질의처리 부분은 “질의 토큰화” 단계, “어간추출” 단계, 그리고 “불리언 연산” 단계로 구성된다. 질의 토큰화 단계에서는 사용자의 질의를 키워드 단위로 분리하며, 어간추출 단계에서는 분리된 키워드의 어간을 추출해 낸다. 추출된 어간을 이용하여 색인 파일에 접근하고, 해당 색인어가 가리키는 포스팅 파일의 문서 링크를 이용하여 해당 문서를 추출한다. 그리고 추출된 문서들에 대해서 적절한 불리언 연산을 수행하여 그 결과를 사용자에게 제공한다.

어간 추출은 사용자 질의처리 부분 뿐만 아니라 역색인 파일처리 부분에서도 사용된다. 일반 용어 대신에 어간을 추출하여 저장함으로써 단일 어간을 다수의 완전한 용어와 대응시킨다. 이것은 저장 공간의 절약뿐만 아니라 검색 시간의 단축에도 효과적이다.

역색인 파일처리 부분은 “문서 로딩 및 토큰화” 단계, “불용어 제거” 단계, “어간추출” 단계, 그리고 “역색인 파일갱신” 단계로 구성된다. 문서 로딩 및 토큰화 단계에서는 웹 크롤러가 가져온 웹 문서의 HTML 태그를 제거한 후 색인어의 대상이 되는 단어들로 분할한다.

그리고 불용어 제거 단계에서는 해당 단어가 불용어인지를 검사한다. 불용어란 영어의 관사 “a, an, the”와 같이 문서에서 매우 빈번하게 사용되는 단어로서, 문서를 식별하는 능력이 낮아 색인어로써 부적합 것들이다. 이 단계에서 색인어 후보로부터 불용어를 제거함으로써, 색인어의 수를 줄이고 검색 시스템의 처리를 고속화한다.

마지막으로 어간추출 단계를 거쳐 최종적으로 추출된 색인어는 역파일 구조로 저장된다. 역 파일은 (그림 1)의

아래와 같이 크게 색인 파일, 포스팅 파일, 그리고 문서 파일로 구성된다. 색인 파일은 색인어, 색인어의 빈도수, 그리고 해당 색인어에 대응되는 포스팅 파일로의 링크 정보로 구성된다. 포스팅 파일은 해당 색인어를 가지는 문서들을 가리키는 링크 정보로 구성되고, 각 링크는 해당 문서를 가르킨다. 이러한 역색인 파일 정보를 이용하여 사용자가 입력한 키워드로부터 해당 문서를 추출하여 사용자에게 제공한다.

예를 들어 (그림 1)과 같은 자료를 가진 정보검색 시스템에서 사용자가 "pease AND pot"이라는 질의를 하면 질의 토큰화 단계에서는 pease와 pot으로 질의가 분해되고 해당 키워드를 색인 파일에서 검색한다. 그리고 해당 색인어에 대응되는 포스팅 파일을 통해서 문서#1과 문서#2가 "pease"를 가지고 있고, 문서#2와 문서#5가 "pot"를 가지고 있음을 알 수 있다. 불리언 연산 단계에서는 해당 문서#1, 문서#2와 문서#2, 문서#5에 대해서 "AND"연산을 수행하여 최종적인 결과로 문서#2를 사용자에게 제공한다. 주어진 그림에서와 같이 오직 문서#2만이 두 단어를 가지고 있음을 알 수 있다.

4. 검색엔진 알고리즘의 병렬화

4.1 병렬화를 위한 고려사항

현존하는 병렬 정보검색 시스템에서는 하드웨어 확장으로 인한 병렬화로 시스템 전체의 작업처리량을 증가시켰으나, 병렬 시스템에서 수행되고 있는 검색엔진 그 자체는 여전히 순차 프로그램이다. 이는 병렬 정보검색 시스템에서 사용자의 개별적인 질의처리 시간이 감소하지 않은 주된 원인이기도 하다.

검색엔진을 구성하는 순차 알고리즘들은 병렬 시스템의 구조에 따라 OpenMP나 MPI를 이용하여 병렬화를 할 수 있다. 그러나 모든 순차 알고리즘들이 병렬화가 가능하거나 또는 성능 향상의 효과를 가지는 것은 아니다. 순차 알고리즘의 효과적인 병렬화를 위해서는 알고리즘의 특성을 파악하여 병렬화가 가능한 부분을 찾고, 효율성 및 효과성을 고려하여 어떻게 병렬화할 것인가를 심사숙고해야 한다.

순차 알고리즘의 병렬화를 위해서 다음과 같이 다섯가지 사항을 고려해야 한다. 첫째, 각 프로세서가 동일한 영역에 접근하는지를 조사해야 한다. 동일한 영역에 접근하는 경우, 읽기 작업만으로 이루어지면 병렬화에 적합하지만, 하나 이상의 쓰기 작업이 존재하면 프로세서들 간의 접근을 적절하게 동기화해야 한다. 이때 동기화는 시스템의 성능저하를 초래하므로, 병렬화의 효과성을 신중히 검토해야 한다.

둘째, 병렬화하고자 하는 부분에서 각 프로세서가 접근하는 데이터들 간에 의존성이 존재하는지를 조사해야 한다. 만약에 데이터들 간에 의존성이 존재하면 병렬화에 적합하지 않다. 예를 들어 프로세서 2가 데이터를 처리하기 위해서 프로세서 1의 처리 결과가 필요하다면 그들은 결코 동시에 수행될 수가 없다. 이런 부분이 병렬화가 되면 순차 프로그램의 속도보다 못한 성능을 초래한다. (그림 2)는 병렬화에 적합한 코드와 적합하지 않은 코드의 예를 보여 주고 있다. (그림 2)의 (a)에서는 매번 반복되는 쓰기 작업의 위치가 서로 다르고, array_1의 각 원소값은 이전 반복문의 처리결과와는 완전히 독립적이다. 따라서 각 반복문을 각 프로세서에 할당하여 병렬적으로 수행 할 수 있다. 반면에 (그림 2)의 (b)에서는 비록 쓰기 작업의 위치가 서로 다르지만, 이전 반복문의 데이터를 읽어서 현재의 반복문을 처리하기 때문에 각 반복문을 각 프로세서에 할당하여 병렬적으로 수행할 수 없다. 이러한 코드를 병렬화하여 컴파일하면 오류없이 수행이 되겠지만, 프로세서의 스케쥴링에 의해서 그 수행 결과는 매번 다를 수 있으며, 사용자가 의도한 결과를 얻을 수 없다.

```
for (i = 1; i < 10; i++){
    array_1[i] = array_2[i] + array_3[i];
}
```

(a) 병렬화에 적합한 코드의 예

```
for (i = 1; i < 10; i++){
    array_1[i] = array_1[i-1] + array_2[i];
}
```

(b) 병렬화에 부적합한 코드의 예

(그림 2) 병렬화 대상 코드 비교

셋째, 병렬 프로그램은 반드시 순차 프로그램의 수행결과와 동일하도록 작성되어야 하고, 그것의 정확성을 검증해야 한다. 왜냐하면 순차 프로그램은 동일한 입력에 대해 항상 동일한 수행 결과만을 보이지만, 병렬 프로그램은 동일한 입력에 대해 서로 다른 결과를 보이는 비결정적 수행 특징을 가질 수 있기 때문이다.

넷째, 순차 프로그램에 대한 병렬화의 복잡성을 고려해야 한다. 만약에 병렬화로 가져오는 성능 향상보다 병렬화하는 과정의 비용이 크다면 병렬화는 재고되어야 할 것이다. 마지막으로 주어진 병렬 시스템 또는 알고리즘의 특성 등을 면밀히 분석하여 그것에 적합한 언어를 선택하는 것이다. 예를 들어 (그림 2)의 (a)와 같은 코드에서는 MPI보다는 OpenMP를 이용한 병렬화가 매우 적합하다. 왜냐하면 OpenMP는 공유된 자료에 대해 프로세서들이 독립적으로 데이터를 처리하는데 적합하고, 그것들의 병렬화가 매우 용이하기 때문이다. (그림 2)의 (a)와 같은 코드에서는 "#pragma omp parallel for"를 for 문 위에 삽입하면 병렬 컴퓨터상에 있는 프로세서들이 해당 반복수를 균등하게 분할해서 동시에 수행하게 된다. 이때 시스템에 의해 자동적으로 균등하게 분할되므로 프로그래머가 프로세서별로 각 반복문을 분할할 필요가 없다.

4.2 검색엔진 알고리즘의 병렬화 가능성 평가

본 절에서는 3절에서 언급한 불리언 정보검색 시스템의 주요 단계에 대한 병렬화 가능성을 분석한다. 먼저 사용자 질의처리 부분을 살펴보면, 질의 토큰화 단계, 어간추출

단계, 그리고 불리언 연산 단계들 중에서 병렬화가 필요한 것은 불리언 연산 단계이다. 질의 토큰화 단계 또는 어간 추출 단계에서는 단일 사용자의 질의에 포함된 색인어의 수가 서너 개에 불과하기 때문에 병렬화의 효과는 적다.

반면에 불리언 연산 단계는 사용자 질의응답 시간의 상당 부분을 차지하므로 병렬화가 필요하다. 왜냐하면 사용자가 제시한 하나의 색인어에 대해서 수천 또는 수만의 문서들이 연결된 리스트에 대해서 "AND"와 같은 불리언 연산을 순차적으로 수행하기 때문이다. 따라서 추출된 수천 또는 수만의 문서들에 대한 리스트를 각 프로세서에 분할하여 불리언 연산을 병행적으로 수행하면 보다 빠르게 사용자의 질의에 응답할 수 있다.

역색인 파일처리 부분에서는 불용어 제거 단계, 어간추출 단계, 역색인 파일갱신 단계 등에서 병렬화가 가능하다. 예를 들어, 불용어 제거 단계에서는 수백 또는 수천의 단어들로 구성된 한 문서내의 각 단어에 대해서 불용어 목록 사전의 모든 단어와 하나씩 비교해서 불용어 여부를 결정한다. 이러한 작업을 psuedo 코드로 표현하면 (그림 3)과 같다. 2행의 token_list_size는 한 문서 내 단어들의 수이고, 5행의 stop_words_size는 미리 만들어진 불용어 목록내의 단어들의 수이다. 따라서 바깥 for문의 4행에서는 문서내의 한 단어(token)를 읽어오고, 내포 for문의 7행에서는 해당 단어가 불용어 목록(stop_words)내의 단어와 일치하는지를 비교한다.

이때 문서 내의 각 단어들은 서로 독립적으로 불용어 목록의 단어들과 비교되므로, 각 단어에 대한 병렬화가 가능하다. 즉 바깥 for문의 각 반복을 각 프로세서에 할당하여 병행적으로 내포된 for문을 수행하는 것이다. OpenMP에서는 (그림 3)의 1행과 같이 "#pragma omp parallel for"디렉티브를 바깥 for문 위에 삽입함으로써 간단히 병렬화한다.

병렬화전의 불용화 제거 단계에서는 n개의 단어로 구성된 하나의 문서에 대해서 $O(n)$만큼의 시간이 소요되는 반면에, OpenMP를 이용하여 n개의 프로세서로 병행하게 처리하면 시간 복잡도는 $O(1)$가 된다. 어간추출 단계 역시 불용어 제거 단계와 유사하다. 한 문서내의 수백 또는 수천의 단어들에 대해서 독립적으로 어간을 추출하게 되는데, 이들 또한 순차적으로 처리하지 않고 불용어 제거의 병렬화와 같이 동시에 처리될 수 있다.

역색인 파일갱신 단계에서는 어간 추출을 거친 하나의 색인어가 주어지면, 색인 파일에서 해당 색인어를 검색하여 빈도수를 증가시킨다. 그리고 해당 포스팅 파일에서는 문헌 링크수를 증가시키고, 관련 문서를 연결한다. 이때 한 문서 당 수백에서 수천에 이르는 각 단어에 대해서 위와 같은 작업을 반복한다. 따라서 한 문서 내의 단어들을 각 프로세서에 분할하여 역색인 파일갱신 단계를 병렬화할 수 있다. 이때 색인 파일이 쓰기 작업으로 갱신이 되더라도, 갱신되는 위치는 각 색인어마다 다르므로 동기화 없이 병렬화가 가능하다.

```
01: #pragma omp parallel for
02: for (i = 1; i < token_list_size; i++)
03: {
04:         token = token_list[i];
05:         for(j = 1; j < stop_words_size; j++)
06:         {
07:             if (token == stop_words[j])
08:             {
09:                 token는 불용어
10:             }
11:         }
```

(그림 3) 불용어 제거를 위한 pseudo 코드

5. 결론

본 연구에서는 전형적인 불리언 정보검색 시스템 상에서 수행되는 검색엔진의 순차 알고리즘에 대한 병렬화 가능성을 살펴보았다. 즉, 사용자 질의처리 부분에서는 불리언 연산 단계가 병렬화에 적합하고, 역색인 파일처리 부분에서는 불용어 제거 단계, 어간추출 단계, 역색인 파일갱신 단계에서 병렬화가 필요함을 살펴보았다. 또한 OpenMP를 이용하여 불용어 제거 단계의 순차 처리를 간단하게 병렬화하는 예를 보였다. 향후 연구에서는 병렬 정보검색 시스템의 각 순차 알고리즘을 병렬화하여 실제적으로 정보검색 시스템의 성능에 어느 정도 영향을 미치는지를 객관적으로 실험하고자 한다.

참고문헌

[1] 강재호, 양재완, 정성원 외, "*효율적인 병렬정보검색을 위한 색인어 군집화 및 분산저장 기법*", 한국정보과학회, 정보과학회논문지: 소프트웨어 및 응용 제30권 제1·2호, pp. 129-139, 2003. 2.

[2] 김명철, 김덕봉, 김유성 외, 최신정보검색론, 홍릉과학출판사, 2001.

[3] 류근호, 김진호 공역, 정보검색, 시그마프레스, 1995.

[4] 성경복, 이재길, 황규영, "*오디세우스/Parallel-OOSQL에 기반한 대규모 병렬 정보검색 서비스 시스템 아키텍처*", 한국정보과학회, 한국정보과학회 학술발표논문집, 제31권 제2호(Ⅱ), pp. 109-111, 2004. 10.

[5] Barroso, L.A., Dean, J., and Holzle, U., *"Web search for a planet: The Google cluster architecture,"* IEEE Micro, Vol. 23(2), pp. 22-28, March-April 2003

[6] OpenMP Architecture Review Board, OpenMP Application Program Interface Version 2.5, 2005

[7] Snir, M., S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, MPI: The Complete Reference, MIT Press, 1996.