http://dx.doi.org/10.5392/JKCA.2016.16.10.608

Apache Spark을 이용한 병렬 DNA 시퀀스 지역 정렬 기법 구현

Implementation of Parallel Local Alignment Method for DNA Sequence using Apache Spark

김보성*, 김진수**, 최도진***, 김상수****, 송석일**
(주)이글루시큐리티*, 한국교통대학교 컴퓨터공학과**, 충북대학교 정보통신공학과***, 한국산업인력관리공단****

Bosung Kim(jikol2000@gmail.com)*, Jinsu Kim(biosvos@gmail.com)**,
Dojin Choi(mycdj91@gmail.com)***, Sangsoo Kim(kiss4rang@hanmail.net)****,
Seokil Song(sisong@ut.ac.kr)**

요약

Smith-Waterman(SW) 알고리즘은 DNA 시퀀스 분석에서 중요한 연산 중 하나인 지역 정렬을 처리하는 알고리즘이다. SW 알고리즘은 동적 프로그래밍 방법으로 최적의 결과를 도출할 수 있지만 수행시간이 매우 길다는 문제가 있다. 이를 해결하기 위해서 다수의 노드를 이용한 병렬 분산 처리 기반의 SW 알고리즘이 제안되었다. Apache Spark을 기반으로 하는 병렬 분산 DNA 처리 프레임워크인 ADAM에서도 SW 알고리즘을 병렬로 처리하고 있다. 하지만, ADAM의 SW 알고리즘은 Smith-Waterman 이 동적프로그래밍 기법이라는 특성을 고려하지 않고 있어 최대의 성능을 얻지 못하고 있다. 이 논문에서는 ADAM의 병렬 SW 알고리즘을 개선한다. 제안하는 병렬 SW 기법은 두 단계에 걸쳐 실행된다. 첫 번째 단계에서는 지역 정렬 대상인 DNA 시퀀스를 다수의 파티션(partition)으로 분할하고 분할된 각 파티션에 대해서 SW 알고리즘을 병렬로 수행한다. 두 번째 단계에서는 파티션 각각에 대해서 독립적으로 SW를 적용함으로써 발생하는 오류를 보완하는 과정을 역시 병렬로 수행한다. 제안하는 병렬 SW 알고리즘은 ADAM을 기반으로 구현하고 기존 ADAM의 SW와 비교를 통해서 성능을 입증한다. 성능 평가 결과 제안하는 병렬 SW 알고리즘이 기존의 SW에 비해서 2배 이상의 좋은 성능을 내는 것을 확인하였다.

■ 중심어 : | DNA 시퀀스 | 지역 정렬 | 병렬 처리 | Apache Spark | Smith Waterman |

Abstract

The Smith-Watrman (SW) algorithm is a local alignment algorithm which is one of important operations in DNA sequence analysis. The SW algorithm finds the optimal local alignment with respect to the scoring system being used, but it has a problem to demand long execution time. To solve the problem of SW, some methods to perform SW in distributed and parallel manner have been proposed. The ADAM which is a distributed and parallel processing framework for DNA sequence has parallel SW. However, the parallel SW of the ADAM does not consider that the SW is a dynamic programming method, so the parallel SW of the ADAM has the limit of its performance. In this paper, we propose a method to enhance the parallel SW of ADAM. The proposed parallel SW (PSW) is performed in two phases. In the first phase, the PSW splits a DNA sequence into the number of partitions and assigns them to multiple nodes. Then, the original Smith-Waterman algorithm is performed in parallel at each node. In the second phase, the PSW estimates the portion of data sequence that should be recalculated, and the recalculation is performed on the portions in parallel at each node. In the experiment, we compare the proposed PSW to the parallel SW of the ADAM to show the superiority of the PSW.

■ keyword : | DNA Sequence | Local Alignment | Prallel Processing | Apache Spark | Smith Waterman |

* 이 논문은 2014년도 한국교통대학교 교내학술연구비의 지원을 받아 수행한 연구임
접수일자 : 2016년 09월 26일 심사완료일 : 2016년 10월 11일
수정일자 : 2016년 10월 11일 교신저자 : 송석일, e-mail : sisong@ut.ac.kr

I. 서 론

빅데이터 기술의 수많은 활용 분야 중에서 대표적으로 유전자 분석을 예로 들 수 있다. 유전자 분석에 빅데이터 기술을 적용하면 대용량의 지놈(Genom) 데이터를 더욱 효율적으로 저장 및 공유하고 빠르게 분석할 수 있도록 해준다. 이를 통해 새로운 질병에 대한 빠른 진단 서비스를 제공할 수 있고 새로운 치료제 개발 가능성이 더 높일 수 있다[1].

기존에는 대용량의 DNA 시퀀스 데이터를 저장하고 빠르게 분석하기 위해서 Hadoop의 MapReduce가 많이 사용되어 왔다. 최근에는 인-메모리 분산 병렬 처리 프레임워크인 Apache Spark[2]을 이용하여 DNA 시퀀스 데이터의 처리 속도를 높이기 위한 ADAM[3]이 제안되었다.

ADAM은 확장 가능한 DNA 시퀀스 데이터 처리 및 저장을 위한 데이터 형식 및 API 집합을 제공하는 프레임워크이다. ADAM은 Hadoop의 에코시스템의 일부인 Parquet[4], Avro[5], Spark으로 구성된다. Parquet는 DNA 시퀀스 데이터를 컬럼기반으로 저장할 수 있는 파일 형식을 제공하는 역할을 수행하며, Avro는 데이터 직렬화 기능을 제공하여 빠른 전송 및 이식성을 높인다. Spark은 인-메모리 분산 병렬처리 프레임워크로 DNA 시퀀스 데이터에 대한 빠른 분석을 가능하게 한다.

ADAM은 DNA 시퀀스 데이터 분석에 있어 핵심 연산 중 하나인 지역 정렬 (Local Alignment) 기법으로 Smith-Waterman (SW)[6][7]를 사용하고 있다. SW는 동적 계획(Dynamic Programming) 법으로 최적의 해를 얻을 수 있지만, 수행시간이 오래 걸린다는 단점을 가지고 있다. SW의 이런 단점을 극복하기 위해서 근사(Approximate) 접근법을 사용하는 방식[8]들과 근사 접근법 및 SW에 병렬처리를 적용하는 방식[9-13]들이 제안되었다.

ADAM의 SW는 Spark의 인-메모리 분산 병렬 처리 프레임워크를 이용해서 처리속도를 높이는 방식을 사용하고 있다. 하지만, ADAM은 SW이 동적프로그래밍 방법이라는 점 때문에 병렬성을 최대로 얻지 못하고 있어 성능에 제약이 존재한다.

이 논문에서는 ADAM의 SW 알고리즘을 개선한다. 정렬 대상인 DNA 시퀀스를 분할하여 여러 파티션으로 나누고 파티션을 다중 노드에 할당하여 노드별로 독립적인 SW 알고리즘을 수행한다. 하지만, SW는 동적 프로그래밍 기법이므로 분할된 각 파티션들은 상호 의존성이 있어 단순히 분할한 파티션에 대해 SW를 적용해서는 정확한 계산 결과를 얻지 못한다.

이 논문에서 제안하는 병렬 SW (Parallel SW, PSW)는 분할한 파티션에 대해 SW 알고리즘을 적용하여 계산한 후에 파티션간의 의존성을 고려하지 못한 영역을 찾아내어 재계산을 수행한다. 재계산 역시 병렬로 수행되며 재계산을 수행하게 되면 완전한 SW 해를 구할수 있다.

이 논문은 다음과 같이 구성되어 있다. 2장에서는 관련연구를 요약하여 기술하고, 3장에서 제안하는 PSW를 자세하게 기술한다. 4장에서는 제안하는 PSW와 ADAM의 SW를 비교하는 실험을 수행하고, 마지막으로 5장에서 결론을 맺는다.

II. 관련연구

1. Smith-Waterman (SW) 알고리즘[7]

SW알고리즘은 알파벳 A, C, G, T 로 구성되는 DNA 시퀀스를 일정한 점수 부여 방법을 통해 동일한 문자가 비슷한 위치에 얼마나 연속적으로 나타나는지를 계산하는 형태로 지역 정렬을 수행한다. 이 논문에서는 수식 1과 같은 점수 부여 방법의 SW 알고리즘을 기준으로 설명한다. 수식 1의 SW 알고리즘은 Weight(또는

$$
\begin{aligned}
&H(i,0)=0,\ 0\le i\le m\\
&H(0,j)=0,\ 0\le j\le n\\
&H(i,j)=\max\left\{\begin{array}{cc}0 & \\ H(i-1,j-1)+s(a_i,b_j) & Match/Mismatch\\ H(i-1,j)+W_k & Deletion\\ H(i,j-1)+W_l & Insertion\end{array}\right\},\ 1\le i\le m,1\le j\le n
\end{aligned}
\tag{1}
$$

Gap), Match, Mismatch 값을 가지고 점수를 부여하여 해당 DNA의 염기 서열의 유사도를 판단한다.

수식 1에서 *a*, *b* 는 각각 DNA 시퀀스 이고, *m* 과 *n* 은 각각 *a* 와 *b*의 길이 이다. *s(a, b)*는 *a*와 *b*를 구성하는 알파벳 간의 유사도를 계산해 주는 유사도 함수이다. *H(i,j)*는 *a[1...i]*와 *b[1...j]*의 접미부(suffix)간의 최대 유사도 점수를 의미하며 행렬 H의 i행 j열에 기록한다. 이 논문에서는 *s(a,b)*는 Match일 경우 +5, Mismatch일 경우 -4를 반환한다. 또한, 가중치인 W_i는 -3의 값을 갖는다.

[그림 1]은 SW 알고리즘의 예를 보여준다. 가상의 DNA 서열인 'AGATGA'와 'ACGTGA'의 유사성을 계산한다고 할 때 SW 알고리즘은 [그림 1]에서와 같은 단계로 계산을 진행한다.

	u	A	G	A	T	G	A
u	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0

(a)

	u	A	G	A	T	G	A
u	0	0	0	0	0	0	0
A	0	5	0	0	0	0	0
C	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0

(b)

	u	A	G	A	T	G	A
u	0	0	0	0	0	0	0
A	0	5	1	0	0	0	5
C	0	1	2	0	0	0	1
G	0	0	6	2	3	5	1
T	0	0	2	3	7	8	4
G	0	0	5	1	8	12	8
A	0	5	1	10	6	8	17

(c)

그림 1. Smith Waterman 알고리즘 예

[그림 1](a)는 초기단계 이다. 각 DNA 서열 앞에 u를 추가하고 2차원 행렬을 구축한 후 모두 0으로 초기화 한다. [그림 1](b)는 계산을 진행하는 과정이다. 그림에서처럼 회색 셀부터 오른쪽으로 진행하면서 각 셀의 값을 계산한다. 셀의 값은 [그림 1]의 수식을 이용해서 계산한다. [그림 1](c)는 계산을 모두 끝낸 행렬의 모습을 보여준다. 계산식과 예에서 보는 바와 같이 SW알고리즘은 인접하는 셀의 계산 결과가 다음 셀의 계산 결과에 영향을 미치게 되는 구조이다.

2. ADAM[3]

ADAM은 확장가능하게 DNA 시퀀스 데이터를 처리하기 위해서 데이터 형식 (Format) 및 API 집합을 제공하는 프레임워크이다. [그림 2]에서 ADAM의 전체 구조를 보여준다. 이 그림에서 Physical storage 계층은 실제 DNA 시퀀스 데이터가 물리적으로 기록되는 곳을 말하며 일반적으로 하드 디스크에 기록한다. Data Distribution 계층은 Hadoop의 분산 스토리지 시스템인 HDFS과 같이 DNA 시퀀스 데이터 파일들에 대해 분산 저장과 데이터 손실 방지하기 위한 복제가 이루어지며 저장 용량을 확장할 수 있다.

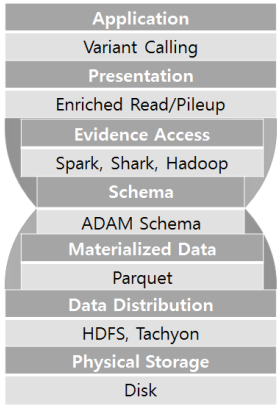


그림 2. ADAM의 구조[3]

Materialized Data 계층에서는 효율적인 데이터 입출력을 위해 인코딩하고 저장하는 패턴 및 데이터 압축기능을 제공한다. 이 계층에는 Hadoop 에코시스템 중 하나인 Parquet가 사용된다. Parquet는 DNA 시퀀스 데이터의 표준 파일 형식인 BAM 파일을 ADAM 파일 형식으로 변환한다. ADAM 파일 형식은 시퀀스 데이터가 하나의 컬럼으로 저장되어 분석에 효과적이다.

Schema 계층은 DNA 시퀀스 데이터에 대한 스키마를 정의하고 표현한다. 이를 통해서 상위 계층에서 스

키마를 통해 DNA 시퀀스 데이터를 접근할 수 있도록 한다. 다음 계층은 Evidence Access계층이다. 이 계층에서는 플랫 파일(Flat File)에서 순차적인 레코드 읽기, 랜덤 데이터베이스 질의(Random Database Queries)와 같은 기능을 제공한다.

3. Apache Spark[2]

Spark은 분산 인-메모리 환경에서 병렬로 데이터 처리를 수행할 수 있도록 하는 프레임워크이다. Spark은 인-메모리에서 처리하는 데이터의 손실을 방지하기 위해서 RDD(Resilient Distributed Data Set)라는 개념을 도입하였다. Spark에서 모든 데이터는 RDD로 만들어져서 처리에 사용된다. RDD는 데이터의 변경 과정을 일으킨 연산들을 저장하고 있다가 시스템 고장으로 메모리의 데이터가 손실되면 연산을 재수행하여 데이터를 복구한다. 이를 통해 처리중인 데이터를 영구 저장 장치에 저장할 필요성을 제거하여 입출력이 없이 빠르게 데이터 처리가 가능하다.

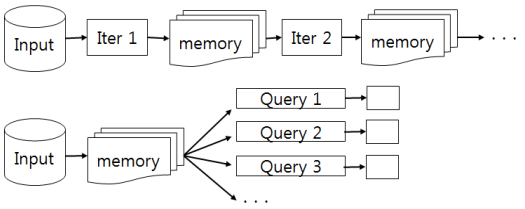


그림 3. Spark의 데이터 처리과정

[그림 3]은 Spark에 데이터를 처리하는 과정을 나타낸 것이다. 그림에서 보는 것처럼 Spark은 맵 리듀스와 다르게 데이터 처리를 위해 입력된 데이터에 대하여 중간 단계의 연산을 가해서 얻은 결과물을 메모리에 유지하고 이를 다시 다음 단계의 연산에 입력으로 넘겨준다. 이러한 접근은 중간 결과물에 대한 하드디스크 저장 필요성을 제거하여 처리속도를 크게 향상 시킬 수 있다.

또한 그림에서처럼 입력 데이터를 모두 메모리에 저장한 상태에서 SQL 문장을 통해 분석을 수행할 수 있다. Spark은 Hadoop의 HDFS와 호환가능하고 데이터 분석에 다단계의 반복연산이 잦은 기계 학습 (Machine Learning) 및 그래프 처리 (Graph Processing) 등의 알고리즘을 구현하는데 적합하다.

4. Spark-SW (Spark Smith Waterman)[9]

Spark-SW는 DNA 시퀀스 데이터를 HDFS에 분산 저장 하며 처리시에는 DNA 시퀀스 데이터를 다수의 파티션으로 분할하여 맵을 수행하는 구조이다. 전체적인 처리과정이 [그림 4]에 나타나 있다. Spark-SW의 각 맵들은 분할된 각 파티션에 대해서 SW 알고리즘을 수행하고 최대 유사도 점수를 반환한다.

SW 알고리즘을 통해서 생성된 행렬은 RDD 형태로 저장된다. Spark-SW의 리듀스 과정에서는 맵의 결과인 각 파티션들의 최대 유사도 점수를 모아서 정렬하고 RDD형태로 저장되어 있던 행렬을 최대 유사도 점수를 기반으로 순차적으로 수정한다. Spark-SW는 맵과정은 병렬로 수행되지만, 리듀스 과정은 순차적으로 진행되어 전 과정의 병렬화는 달성하지 못한다.

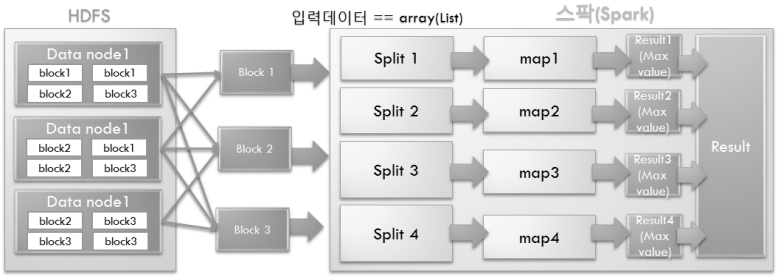


그림 4. Spark-SW의 전체 흐름

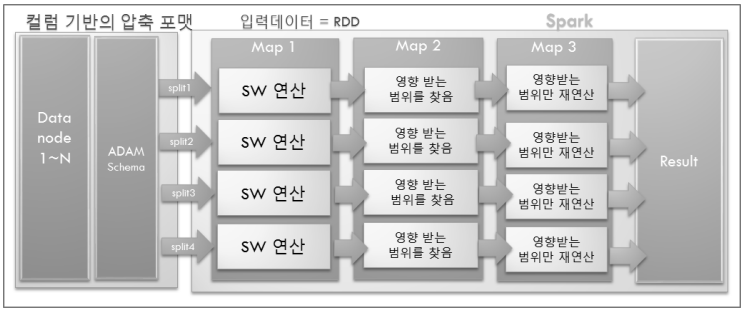
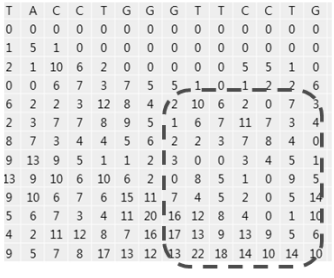


그림 5. 제안하는 PSW의 처리 과정



(a) 기존의 SW 알고리즘 실행 결과

(b) 병렬 SW 알고리즘의 실행 결과

그림 6. 병렬로 SW를 수행할 때 발생하는 문제

III. 제안하는 병렬 Smith-Waterman (PSW) 기법

이 논문에서는 DNA 시퀀스 데이터에 대한 분산 병렬 처리 프레임워크인 ADAM의 SW 알고리즘을 개선하는 방안을 제안하고 이를 구현한다. [그림 5]는 제안하는 PSW의 처리 과정을 보여준다. 그림에서처럼 제안하는 PSW는 ADAM 파일 형식으로 변환된 DNA 시퀀스 데이터를 처리한다. 앞서 설명한 것처럼 ADAM 파일 형식이 컬럼 기반의 압축을 지원하므로 PSW에서 효과적으로 DNA 시퀀스 데이터를 로드하여 처리할 수 있다.

PSW는 ADAM 형식으로 저장된 DNA 시퀀스 데이터를 노드의 수와 코어의 수를 고려하여 분할한다. 이때 분할 된 각각을 파티션이라 한다. PSW는 Spark의 맵 연산을 이용하여 각 파티션에 대해 SW 알고리즘을 적용하여 행렬을 생성한다.

하지만, SW가 동적프로그래밍 기법이어서 행렬을 구축할 때 이웃하는 셀의 값을 이용하게 되어 있으므로 각 파티션의 시작 열의 셀 값들은 정확하지 않을 수 있다. [그림 6]에서 이에 대한 예를 보여준다. [그림 6](a)는 분할 없이 SW를 적용한 결과이고, [그림 6](b)는 DNA 시퀀스 데이터를 두 개의 파티션 N과 N+1로 분할한 후에 각각의 파티션에 독립적으로 SW를 적용한 결과를 보여주고 있다.

두 그림에서 각 행렬의 점선으로 표시한 영역의 셀 값들 중 일부 셀들이 서로 다른 것을 볼 수 있다. 이것은 [그림 6](b)의 경우 파티션 N+1의 시작하는 열의 셀 값들을 계산할 때 파티션 N의 마지막 열의 셀값들을 고려하지 않아서 발생하는 문제이다. 이 문제를 해결하기 위해서는 [그림 6](b)의 파티션 N의 마지막 열의 셀값들을 이용하여 파티션 N+1의 시작 열부터 다시 계산을 수행해야 한다. 이를 위해서는 어느 셀들까지 계산을 다시 해야 하는지 확인하는 절차가 선행되어야 한다.

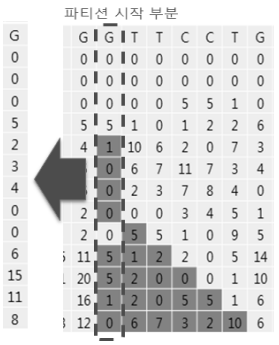


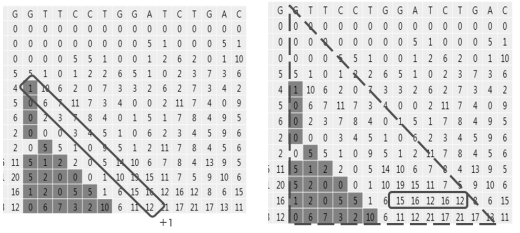
그림 7. 재계산 영역 추정-1

재계산이 필요한 영역을 추정하기 위해서 [그림 7]에서처럼 파티션 N의 시작 열을 재계산한 후에 재계산 전의 열의 값들과 재계산 후의 열의 값을 비교한다. SW는 셀의 값을 [그림 1]의 수식에 따라서 계산하므로 재계산한 셀의 값은 변화가 없거나 증가하게 된다. 이때 각 셀의 증가하는 값을 *diffVal[]*이라 한다. 이 *diffVal[]*은 경계 부분의 열 (파티션의 시작 열)부터 어디까지 재계산을 수행해야 하는지 계산하기 위해서 사용된다.

행렬의 행들 중 최초로 값이 다른 행부터 그 아래의 각 행에 대해서 *diffVal[]*의 해당 행에 대한 값(*diffVal*)을 수식 2에 대입하여 해당 행에 재계산을 해야 하는 범위를 계산한다. [그림 8](a) 의 경우 최초로 값이 다른 행이 5번째 이다. 이 행에 대한 *diffVal*이 1이고, 해당 행의 셀의 값(*curVal*)이 5, 행의 인덱스(*rowIdx*)이 4, 질의 길이 (행렬의 행의 개수, *len(Query)*) 가 13이므로 수식의 계산 결과는 8이 된다. 이것의 의미는 해당 셀의 현재 열로 부터 8이 증가된 열까지가 영향을 받게 된다는 것을 의미한다.

$$affectedRange = \frac{(prevCurVal + prevDiffVal)}{weight} + \frac{(curVal + diffVal)}{weight} + (len(Query) - rowIdx) \quad (2)$$

이와 같은 방식으로 최초의 행으로 부터 아래의 모든 행에 대해서 계산을 해 나가고 그중에 affectedRange의 값이 가장 큰 것이 재계산을 수행해야 되는 영역이다. 최종적으로 계산한 결과가 [그림 8](b) 이다. PSW에서는 맵 연산을 통해 파티션 별로 병렬로 범위를 구한 후 다시 맵 연산을 통해서 파티션 별로 해당 범위 내에서 재계산을 병렬로 수행한다. 이렇게 재계산이 끝나고 나면 분할하지 않고 SW를 수행한 결과와 동일한 결과를 얻게 된다.



(a) 재계산 영역 추정 (b) 최종 추정 재계산 영역

그림 8. 재계산 영역 추정-2

[그림 9]는 지금까지 설명한 재계산이 필요한 영역을 추정하는 알고리즘을 보여준다. 이 연산의 입력은 *scoreMat[][]*과 *lcPrevPart[]* 이다. 각각은 N 번째 파티션의 유사도 점수 행렬과 N-1 파티션의 마지막 열을 의미한다. 알고리즘은 가장 먼저 *scoreMat[0][]*과 *lcPrevPart[]*의 대응하는 각 셀을 비교하여 값의 차이를 *diffVal[]*에 저장한다. 그리고, 최초로 값의 차이가 나타나는 열의 인덱스를 *diffIdx* 에 저장한다. 다음은 최초로 값이 차이가나는 열로 부터 수식 2의 계산을 반복하여 수행한다. 알고리즘에서는 이를 반복문으로 기술하였는데 실제로는 map연산을 통해서 수행된다.

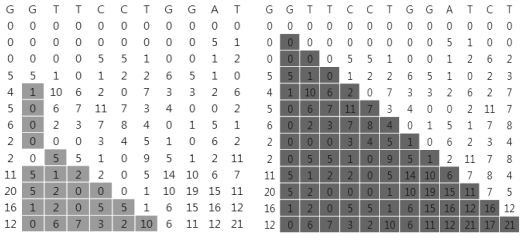
```
Algorithm get_area4recalculation()
입력 :
  scoreMat[][] : Partition N 의 유사도 점수 행렬
  lcPrevPart[] : Partition N-1 의 마지막 컬럼
출력 :
  area4recalc : 재계산을 해야 되는 영역
{
  diffVal[], diffIdx = scoreMat[0][]과 lcPrevPart[]의 값의 차이,
        값이 차이나는 첫번째 요소 인덱스;

  for (rowIdx = diffIdx+1; i < columnLen; i++)
    affectedRange[rowIdx] =
                (scoreMat[0][rowIdx-1]-diffVal)/weight
                + (scoreMat[0][rowIdx]-diffVal)/weight
                + len(Query) - rowIdx;
  return affectedRange[];
}
```

그림 9. 재계산이 필요한 영역 계산 알고리즘

[그림 9]의 알고리즘에 의해 계산된 재계산 필요 영역은 실제 재계산을 수행해야 하는 영역과는 다소 차이

가 있다. [그림 10]은 실제 재계산이 필요한 영역과 제안하는 알고리즘으로 추정한 재계산이 필요한 영역간의 차이를 보여준다. 다소 차이는 있지만, 이 차이는 수식 2에 따르면 질의 시퀀스 (Query)의 길이에 영향을 받는다. 하지만, 일반적으로 질의 시퀀스는 각 파티션의 행의 길이에 비해 매우 작으므로 성능상에 큰 영향을 주지 않을 것으로 판단된다.



(a) 실제 재계산이 필요한 영역 (b) 추정한 재계산 영역

그림 10. 실제 재계산 영역과 추정한 재계산 영역의 차이

Ⅳ. 성능평가

이 논문에서는 제안하는 PSW의 성능을 검증하기 위해서 ADAM을 기반으로 PSW를 구현하였다. 그리고, 구현한 PSW와 ADAM의 SW의 성능을 비교하였다. 실험 환경은 [표 1]과 같다. 4개의 노드로 구성되는 클러스터 환경을 구축하였다. 각 노드에는 CentOS 6.6의 리눅스 운영체제를 설치하였으며, Spark 1.5, ADAM 0.17, Hadoop 2.3 버전을 설치하였다.

실험에 사용된 각 노드는 Intel(R) Core(TM) i3-2100 CPU에 8GB의 RAM의 하드웨어 사양을 갖는다. 또한, 각 CPU는 4개의 코어를 가지고 있어 클러스터에 포함된 전체 코어 수는 16개가 된다. 실제 실험시에는 1개의 마스터(Master) 노드와 3개의 슬레이브(Slave) 노드로 구성하였다. 마스터 노드를 제외한 슬레이브 노드들은 6GB의 RAM을 ADAM에 할당하여 사용하였다. 실험에 사용한 DNA 시퀀스데이터는 410,868,898개의 염기서열로 구성되어 있으며, 질의 시퀀스는 10개의 염기서열로 구성된다.

실험은 DNA 시퀀스 데이터의 파티션수를 변화시켜가면서 쿼리 시퀀스와의 지역 정렬을 완료하는데 걸리는 시간을 측정하였다. 파티션의 수는 1에서 100,000개까지 변화시켜가면서 실험을 진행하였다.

이때 파티션의 수가 1이라는 것은 ADAM의 SW 알고리즘을 수행했다는 것을 의미한다. 즉, [그림 10]의 그래프에서 파티션의 수가 1인 것은 ADAM의 SW 의 성능을 의미한다. 파티션의 수가 1보다 큰 실험의 결과는 이 논문에서 제안한 PSW 의 성능을 의미한다. 그림에서처럼 파티션의 수가 증가되면서 실행시간은 점점 줄어드는 것을 볼 수 있다. 하지만, 파티션의 수가 50보다 커지면 실행시간이 오히려 증가하는 것을 볼 수 있다.

표 1. 실험 환경

실험 환경	값
Node 수	4
운영체제	CentOS 6.6
CPU	Intel(R) Core(TM) i3-2100 CPU @ 3.10GHz / 16
RAM	18
Spark	1.5.0
ADAM	0.17.0
Hadoop	2.3.0
시퀀스 데이터 수	410,868,898
질의 시퀀스 수	10

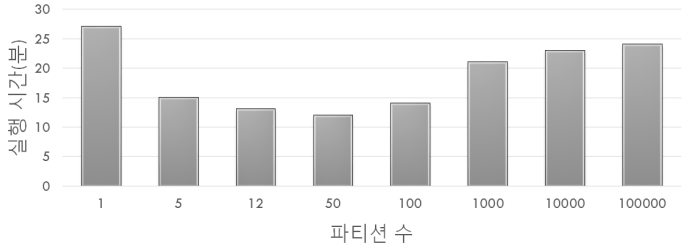


그림 11. 실험 결과 : 파티션 개수에 따른 실행시간

실험에서 지역정렬을 수행하는데 사용된 노드의 수는 3개에 코어의 전체수는 12개였다. 실험 결과를 보면 파티션의 수가 12개와 가까울 때 성능이 가장 좋음을 알 수 있다. 파티션의 수가 코어의 수보다 커지게 되면 파티션의 크기가 작아지고, 작아지는 만큼 재계산을 수행해야 하는 범위가 넓어지게 된다. 또한, 코어의 수보다 훨씬 더 많은 파티션은 오히려 CPU에 부담을 주게 되어 성능을 저하시키는 것을 볼 수 있다. 결론적으로 이 논문에서 제안한 PSW는 병렬성을 이용하여 SW 알고리즘의 실행 시간을 절반 정도로 줄였다.

V. 결 론

이 논문에서는 ADAM에 저장되어 있는 DNA 시퀀스 데이터에 대한 지역정렬을 병렬로 처리하기 위해 병렬 SW (PSW) 알고리즘을 제안하고 이를 ADAM을 기반으로 구현하였다. 제안하는 방법에서는 DNA 데이터들을 ADAM 파일 형식으로 변환하여 저장한다. 이를 통해 변환된 DNA 시퀀스 데이터를 빠르게 불러 올 수 있고 저장 공간을 효율적으로 사용할 수 있었다.

또한 SW 알고리즘 수행시 병렬성을 최대한 활용하기 위해서 분할 및 SW 실행 단계와 재계산 단계의 2단계로 구성되는 PSW를 제안하였다. 제안한 PSW 알고리즘을 ADAM에서 구현하고 성능평가를 수행하였다. 성능평가를 수행한 결과 PSW는 병렬성을 통해 실행시간을 ADAM의 SW에 비해서 절반 가까이 줄였다.

하지만, 이 논문에는 몇 가지 한계가 있다. 첫 번째는 확장성에 대한 검증이 추가로 필요하다. 이 논문에서는 실행시간을 줄이는데 초점을 두었지만, 확장성도 성능을 측정하는 중요한 요소이다. 향후에는 실험에 사용되는 노드의 수를 늘리고, 실험 데이터의 크기도 더 크게 늘려서 확장성에 대한 실험을 수행할 계획이다. 두 번째는 최신의 관련연구인 Spark-SW와의 성능비교가 되지 않은 점이다. 이 논문에서는 ADAM을 기반으로 제안하는 PSW를 구현하고 성능평가를 수행하였다. Spark-SW는 ADAM을 기반으로 하지 않아서 비교 평가를 하기 위해서는 PSW를 재 구현하거나 Spark-SW를 ADAM에 맞도록 일부 수정해야 하는 이유로 성능 비교를 수행하지 않았다. 향후 연구에서는 Spark-SW를 ADAM에 맞도록 수정하여 실행시간 및 확장성 측면의 성능 비교를 수행할 계획이다.

참 고 문 헌

[1] 복경수, 유재수, “빅데이터 활성화 정책 및 응용 사례,” 정보과학회지, 제32권, 제11호, pp.46-57, 2014.

[2] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster Computing with Working Sets,” Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, Vol.10, pp.10-10, 2010.

[3] M. Massie, F. Nothaft, C. Hartl, C. Kozanitis, A. Schumacher, A. D. Joseph, and D. A. Patterson, “Adam: Genomics Formats and Processing Patterns for Cloud Scale Computing,” University of California, Berkeley Technical Report, No. UCB/EECS-2013, 2013.

[4] Parquet, http://www.parquet.io.

[5] Avro, http://avro.apache.org.

[6] T. F. Smith and M. S. Waterman, “Identification of Common Molecular Subsequences,” Journal of Molecular Biology, Vol.147, No.1, pp.195-197, 1981.

[7] https://en.wikipedia.org/wiki/Smith-Waterman_algorithm

[8] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic Local Alignment Search Tool," Journal of Molecular Biology, Vol.215, No.3, pp.403-410, 1990.

[9] G. Zhao, C. Ling, and D. Sun, “SparkSW: Scalable Distributed Computing System for Large-scale Biological Sequence Alignment,” In 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp.845-852, 2015.

[10] D. R. Mathog, "Parallel BLAST on Split Databases," Bioinformatics Application Node, Vol.19, No.14, pp.1865-1866, 2003.

[11] 김동욱, 최한석 "그리드 컴퓨팅을 이용한 BLAST 성능개선 및 유전체 서열분석 시스템 구현," 한국콘텐츠학회논문지, 제10권, 제7호, pp.81-87, 2010.

[12] A. Julich, "Implementations BLAST for Parallel Computers," CABIOS, Vol.11, No.14, pp.3-6, 1995.

[13] V. Breton, E. Caron, F. Desprez, and G. L. Mahec, "BLAST Application with Data-Aware Desktop Grid Middleware," In Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp.284-291, 2009.

저 자 소 개

김 보 성(Bosung Kim) 준회원

- 2014년 2월 : 한국교통대학교 컴퓨터공학과(공학사)
- 2016년 2월 : 한국교통대학교 정보기술융합학과(공학석사)
- 2016년 6월 ~ 현재 : (주)이글루시큐리티 근무

<관심분야> : 데이터베이스, 빅 데이터, 스토리지 시스템, 시스템 보안 등

김 진 수(Jinsu Kim) 준회원

- 2015년 8월 : 한국교통대학교 컴퓨터공학과(공학사)
- 2015년 8월 ~ 현재 : 한국교통대학교 컴퓨터공학과(공학석사)

<관심분야> : 빅데이터, 스토리지 시스템, 분산/병렬처리 시스템 등

최 도 진(Dojin Choi) 준회원

- 2014년 2월 : 한국교통대학교 컴퓨터공학과(공학사)
- 2016년 2월 : 한국교통대학교 컴퓨터공학과(공학석사)
- 2016년 3월 ~ 현재 : 충북대학교 정보통신공학과 박사과정

<관심분야> : 연속 질의 처리, 그래프 스트림, 데이터베이스, 빅데이터

김 상 수(Sangsoo Kim) 정회원

- 1995년 1월 : 교보문고 전산실
- 1995년 2월 : 목포대학교 전산통계학과(이학사)
- 2000년 8월 : (주)가인정보기술
- 2006년 2월 : 전북대학교 컴퓨터공학과(공학석사)
- 2006년 8월 ~ 현재 : 한국산업인력공단 과정평가기준팀 근무

<관심분야> : 센서 네트워크, 의료 ICT, IOT 등

송 석 일(Seokil Song) 종신회원

- 1998년 2월 : 충북대학교 정보통신공학과(공학사)
- 2000년 2월 : 충북대학교 정보통신공학과(공학석사)
- 2003년 2월 : 충북대학교 정보통신공학과(공학박사)
- 2003년 7월 ~ 현재 : 한국교통대학교 컴퓨터공학과 교수

<관심분야> : 데이터베이스, 센서 네트워크, 스토리지 시스템 등