# 식물 염기 서열 분석 플랫폼의 개발

최동훈, 엄정호, 윤화묵, 최윤수, 이민호, 이원구, 송사광, 정한민
한국과학기술정보연구원 소프트웨어연구실
e-mail:choid@kisti.re.kr

# Development of Analysis Platform for Plant Sequence Assembly

Hoon Choi, Jungho Um, Hwa-mook Yoon, Yun-Soo Choi, Minho-Lee,
Won-Goo Lee, Sa-Kwang Song, Hanmin Jung
Dept. of Software Research, KISTI

요 약

식물 염기 서열 분석은 동물에 비해 유전체가 길고 데이터 양이 많아 특별한 전략을 요구하고 있다. 본 논문은 FCLA 전략을 지원하기 위한 데이터 분석 플랫폼의 개발에 대해 서술한다.

## 1. 서론

식물의 유전체는 동물의 유전체보다 더 길고 복잡한 구조를 가지고 있을 뿐만 아니라 반복 시퀀스가 많기 때문에 참조 서열(reference sequence)이 없는 식물 종의 *De novo* 어셈블리를 위해서는 더욱 주의를 기울여야 한다. 이러한 특징은 차세대 시퀀싱 플랫폼을 대상으로 최근 개발된 프로그램[1,2]이 있지만, FCLA[3]와 같은 다른 전략을 요구한다.

본 연구는 식물 염기 서열 분석을 위해 클라우드 컴퓨팅 기반의 식물 염기 서열 분석 플랫폼을 개발하고자 한다. 식물 종의 *De novo* 어셈블리를 위해 한국생명공학연구원에서 개발한 FCLA 전략을 채택하였으며, FCLA 전략은 선클러스터링에 의한 긴 Contig 생성, 후 Contig 어셈블리에 의한 참조 서열을 생성한다. 그리고 선클러스터링에 포함되어 있는 BLAST, Clustering, D2Clustering은 병렬 프로그램으로 구현되어, 사용자가 사용하기 쉽도록 클라우드 서비스로 제공된다. 이들 병렬 처리는 공개 소스 MapReduce[4] 프레임워크인 Hadoop[5]을 이용하여 수행된다.

## 2. 식물 염기 서열 분석 플랫폼의 요구 사항

식물의 시퀀스 데이터에 대한 드 노보 어셈블리는 FCLA 전략[3]을 따른다. 이 절에서는 FCLA 전략을 소개한다.

### 2.1 BLAST를 이용한 시퀀스의 파싱

(1) BLAST를 이용한 시퀀스 매칭 수행

차세대 시퀀싱 장비를 통해 얻어지는 짧은 시퀀스(short read)는 fasta 파일 형식으로 저장되어 있다. 식물 염기 서열의 정보를 추출, 조합하기 위해서는 이 짧은 시퀀스들을 길게 이어 붙여 긴 contig형태로 얻을 필요가 있다. 이를 위해 National Center for Biotechnology Information(NCBI)에서 제공하는 BLAST[6]라는 소프트웨어를 사용하여, fasta 파일에 저장된 시퀀스들을 서로 비교하여 매치 되는 시퀀스 쌍을 얻는다.

BLAST를 사용하여 서로 다른 두 시퀀스 집합에 포함된 두 시퀀스의 매치를 수행하기 위해서는 시퀀스 집합을 저장하고 있는 두 fasta파일 중에 하나의 파일이 formatdb형태로 변환되어야 하며, 이는 NCBI에서 제공하는 BLAST 패키지에 포함되어 있다. 변환된 formatdb 파일을 DB로 하고 나머지 시퀀스 집합을 저장하고 있는 fasta 파일을 쿼리로 하여 BLAST를 수행한다.

(2) BLAST 결과를 파싱

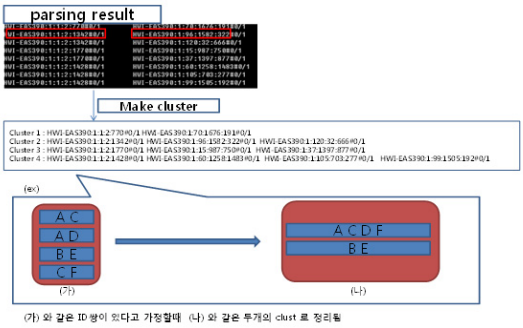
BLAST를 이용해 시퀀스 매치 결과는 query ID, subject ID, identity, coverag, q-start, q-end, s-start, s-end를 결과로 반환한다. query ID와 subject ID는 비교 대상이 되고 있는 두 개 짧은 시퀀스를 나타낸다. 각 시퀀스들은 각각 고유의 ID를 가지고 있으며 이는 전체 실험에서 하나의 시퀀스에 유일하다. identity는 일치율을 의미하며, 매치가 이루어진 시퀀스의 부분에서 얼마만큼의 비율로 일치를 이루고 있는지 나타낸다. coverage는 짧은 시퀀스 중에 몇 개의 글자가 매치되고 있는지를 나타내는 척도이다. Q_start, Q_end, S_start, S_end에서 Q는 쿼리로 들어간 fasta 파일에 속한 짧은 시퀀스를 의미하고 S는 formatDB로 들어간 짧은 시퀀스를 의미한다. start는 매치가 이루어지는 시퀀스의 시작 지점이며, end는 매치가 이루어지는 시퀀스의 끝 지점이다. identity와 coverage, Q_start, Q_end, S_start, S_end정보를 바탕으로 우리가 정말로 필요한 시퀀스 페어만을 골라낼 수 있도록

조건문을 설정하게 되는데 이것이 파싱 조건(parsing criteria)이다.

이처럼 나온 결과를 파싱 조건에 맞는 것들만 뽑아 시퀀스 ID 쌍의 형태로 뽑아준 결과가 파싱 결과로 저장된다. 시퀀스 ID 쌍을 제외한 다른 정보들은 이후에 더 이상 사용되지 않기 때문에 파싱 결과에서 제외된다.

2.2 시퀀스 집합 내에서 시퀀스의 클러스터링

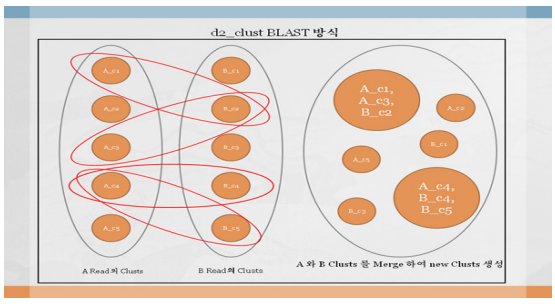
파싱 결과를 바탕으로 기준 이상의 유사도를 가지는 시퀀스 쌍들을 클러스터링한다. 예를 들어 (시퀀스 A, 시퀀스 C), (시퀀스 A, 시퀀스 D), (시퀀스 B, 시퀀스 E) 그리고 (시퀀스 C, 시퀀스 F)의 쌍이 순차적으로 입력으로 들어온다고 가정하면, 먼저 첫 번째 클러스터1에는 시퀀스 A와 그와 쌍을 이루는 시퀀스 C가 포함된다. 이어 다음 입력 (시퀀스 A, 시퀀스 D)을 받으면 시퀀스 A가 포함되고 있는 클러스터 1에 시퀀스 D가 포함된다. 첫 번째 입력과 같은 요령으로 시퀀스 B, 시퀀스 E가 두 번째 클러스터2에 포함되고, 네 번째 입력 (시퀀스 C, 시퀀스 F)에 대해서 시퀀스 C가 포함된 클러스터 1에 시퀀스 F가 포함된다. 이렇게 하여, 그림1과 같은 클러스터가 형성된다.




<그림1> 시퀀스 클러스터링

2.3 시퀀스 집합 간에 클러스터의 병합

두 개의 클러스터 집합 간에 연결되는 정보가 있는지를 보기 위하여 다시 BLAST 검색을 수행한다. 이때 두 결과 사이의 정보를 확인한 후 하나의 클러스터에 결합해, 더 길고 정확한 contig를 얻을 수 있다.




<그림2> 클러스터 병합

위 시퀀스 집합 내의 클러스터링 단계에서 사용한 BLAST와 마찬가지로 coverage가 50%(38bp) 이상이고 identity가 96%이상인 것만 사용 하여 "A" set의 클러스터 중 하나의 서열이라도 "B" set의 클러스터 멤버 시퀀스와 매치된 결과가 있다면 그림2와 같이 두 개의 클러스터를 병합하여 더욱 큰 클러스터를 생성한다.

## 3. 하둡을 이용한 플랫폼의 구현

2.1 BLAST를 이용한 시퀀스의 파싱

하둡 스트리밍(Hadoop streaming)을 이용하면 프로그램을 수정하지 않고 병렬 처리가 가능하다. UNIX 시스템에서 하둡 스트리밍을 이용하여 BLAST 프로그램을 실행할 경우 표1과 같이 mapper와 reducer를 수행하게 된다. BLAST 프로그램을 수행할 때, DB 파일로 사용할 formatdb 파일도 필요로 하므로, "-files"옵션을 이용하여 지정한다.

<표1> BLAST 및 파싱의 병렬 처리

```
$HADOOP_HOME/bin/hadoop jar
$HADOOP_HOME/hadoop-streaming.jar \
    -D mapreduce.job.reduces=24 -D
mapreduce.job.maps=24 \
    -input Input_file \
    -output Output_file \
    -mapper '$BLAST_HOME/bin/blastall -p blastn -d
DB_file -m 8' \
    -reducer filter.py
    -file $BLAST_HOME/bin/blastall \
    -file filter.py \
    -files hdfs://host:fs_port/user/DB_file.nhr#DB_file.nhr \
    -files hdfs://host:fs_port/user/DB_file.nin#DB_file.nin \
    -files hdfs://host:fs_port/user/DB_file.nsq#DB_file.nsq
```

이와 같이 mapper 단계에서 BLAST를 수행한 후, 매치되는 시퀀스 중 서열의 처음이나 끝이 파싱 조건을 만족하는 경우에만 클러스터로 저장하게 된다. 이러한 파싱 작업을 하둡 스트리밍의 reducer 단계에서 실행하는 것은 파일의 입력과 입력 파일의 파싱에 따르는 비용을 줄일 수 있는 장점이 있다. 이를 위해 파싱 프로그램을 표2와 같이 filtering.py을 파이썬으로 작성하여 reducer로 사용한다.

<표2> filter.py 코드

```
#!/usr/bin/python
# filter.py : Hadoop streaming의 reducer 함수로 사용
import sys

def read_mapper_output(file):      # file로부터 한줄 단위로
데이터를 넘겨주는 함수
  for line in file:
    yield line.strip()                    # line 양 옆의 white
```

```
space를 제거하여 하나씩 넘겨줌

def main():
    for data in read_mapper_output(sys.stdin):      # stdin으로부터 데이터를 받음
        try:
            # 입력 받은 라인을 탭 단위로 나눠주는 함수
            tokens = map(lambda x:x.strip(), data.split('\t'))
            QueryID = tokens[0].strip()
            SubjectID = tokens[1].strip()
               IDentity = float(tokens[2].strip())
            Coverage = int(tokens[3].strip())
            Qstart = int(tokens[6].strip())
            Qend = int(tokens[7].strip())
            Sstart = int(tokens[8].strip())
            Send = int(tokens[9].strip())
            # 각 조건을 검사한 후, 조건을 만족하는 ID만 출력
            if QueryID != SubjectID and IDentity >= 96 and Coverage >= 38:
                if Qstart != 1 and Qstart != 2 and Qstart != 75 and Qstart != 76 and\
                  Qend != 1 and Qend != 2 and Qend != 75 and Qend != 76 and \
                  Sstart != 1 and Sstart != 2 and Sstart != 75 and Sstart != 76 and \
                  Send != 1 and Send != 2 and Send != 75 and Send != 76:
                    pass
                else:
                    print "%s,%s"%(QueryID, SubjectID)  # key, value 값으로 리턴
            except ValueError:
                pass                # 에러 발생시 pass
            except IndexError:
                pass                # 에러 발생시 pass
if __name__ == "__main__":
    main()
```

### 3.2 클러스터링 서비스

시퀀스 ID를 클러스터링 하기 위해 그래프의 연결된 컴포넌트 찾는 알고리즘으로 해결하고자 하였다. 한 대의 컴퓨터에서 그래프에서 연결된 컴포넌트 찾는 알고리즘은 다수 존재하나, MapReduce 환경을 고려하여 작성된 알고리즘은 알려진 것이 별로 없다. MapReduce를 이용한 그래프 알고리즘의 병렬 처리[7]를 참조하여 본 과제의 문제를 해결하고자 하였다.

본 알고리즘은 크게 3단계의 과정을 거치게 된다. 첫 번째와 두 번째 단계는 zone 파일과 edge 파일을 이용해서 노드-노드의 관계를 zone-zone 관계로 바꾸는 작업이고 세 번째 단계는 zone-zone 관계에서 가장 작은 zone 값을 찾아서 나머지 zone 값을 가장 작은 zone 값으로 변경해주는 단계이다.

연결된 컴포넌트를 찾는 알고리즘은 초기 입력 파일로 edge 파일 뿐만 아니라 zone 파일도 필요하다. 이를 위해서 edge 파일로부터 zone 파일을 만드는 별도의 과정이 필요하다. 이를 구현한 map 함수와 reduce 함수는 각각 표3, 표4와 같다.

<표3> edge 파일로부터 zone 파일 생성: Map 함수

```
public void map(LongWritable key, Text value, Context context)
  throws IOException, InterruptedException {
    // 구분자를 기준으로 입력 문자열을 나눔
    String[] tokens = value.toString().split(",");
    context.write(new Text(tokens[0].trim()), new Text("|"+tokens[0].trim()));
    context.write(new Text(tokens[1].trim()), new Text("|"+tokens[1].trim()));
}
```

<표4> edge 파일로부터 zone 파일 생성: Reduce 함수

```
public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException
{
    Text nkey = new Text(), nvalue = new Text();
    Text value = values.iterator().next();
    context.write
(new Text(key.toString().trim()+","+value.toString().trim()),
                  new Text());
}
```

연결된 컴포넌트를 찾는 알고리즘의 구현 중에서 1단계의 map과 reduce 함수를 소개하면 표5, 표6과 같다. 이들은 입력으로 받은 edge 파일명 뒤에 "_zone-N" 의 접미어가 붙은 zone 파일을 생성하게 된다. N 값은 수행한 iterator 회수를 의미하고 제일 큰 값이 최종적으로 수행한 결과를 의미한다.

<표5> 1단계 Map 함수

```
// 1단계 Map 함수
public void map(LongWritable key, Text value, Context context)
  throws IOException, InterruptedException {
    String[] tokens = value.toString().split(",");
    if(tokens.length == 2 && tokens[1].charAt(0) == '|') {
        // zone file 인 경우 - 한 개의 레코드를 넘겨줌
        context.write(new Text(tokens[0].trim()),
                  new Text(value.toString().trim()));
    } else {
        // edge file 인 경우 - 두 개의 레코드를 넘겨줌
```

```
            context.write(new Text(tokens[0].trim()),
                          new Text(value.toString().trim()));
            context.write(new Text(tokens[1].trim()),
                          new Text(value.toString().trim()));
        }
        context.progress();              // task의 progress 보고
}
```

<표6> 1단계 Reduce 함수

```
// 1단계 Reduce 함수
public void reduce(Text key, Iterable<Text> values,
Context context)
   throws IOException, InterruptedException {
      MarkableIterator<Text> mitr
       =                                              new
MarkableIterator<Text>(values.iterator());              //
MarkableIterator 선언
      Text zone = new Text();
      Text value = new Text();

      mitr.mark(); // MarkableIterator 마킹
      while(mitr.hasNext()) {
            value = mitr.next();
            if(value.toString().split(",")[1].charAt(0)      ==
'|') {
                  zone.set
(new Text(value.toString().split(",")[1].trim()));
            }
      }
      mitr.reset();      // MarkableIterator 초기화
      while(mitr.hasNext()) {
            value = mitr.next();
            if(value.toString().split(",")[1].charAt(0)      !=
'|') {
                  context.write(value, zone);
            }
      }
      context.progress();  // task의 progress 보고
}
```

3.3 D2Clustering 서비스

표7과 표8은 클러스터 간의 병합 알고리즘의 의사 코드(pseudo code)와 입력 파일이다.

<표7> D2Clustering 입력

```
- Original fasta File List
O_file1_path
O_file2_path
...
- New fasta File
new_file_path
- Original Cluster Path
cluster_path_inHDFS
```

<표8> D2Cluster 의사 코드

```
1. make format db for new fasta file
2. run blast new fasta file to new fasta formatdb
3. make zone file for the result of blast new fasta file
4. for each file x in Original fast file list
   4-1. run blast file x to new fasta formatdb
5. merge files from 4-1, 3 to orgininal cluster path
6. run cluster algorithm on the result of 5
```

## 4. 결론 및 향후 연구 방향

본 논문에서는 식물 염기 서열 분석을 위한 데이터의 분석 플랫폼을 하둡 기반으로 구축하였다. 개발 과정에서 생물학과 고성능 컴퓨팅에 대한 충분한 이해와 제공 가능한 기술에 대한 의견 교환을 통해 요구 사항 명세서를 작성하여 개발에 활용하였으며, 상호 요구 사항의 올바른 분석과 진행 상황의 점검 및 필요시 요구 사항 조정의 과정을 가지며 협력 융합 연구의 형태로 진행하였다.

개발된 플랫폼은 시범 개발 단계에 머무는 수준이지만 향후 지속적인 연구를 통해 더 큰 기술적, 경제적 효과를 기대할 수 있을 것이다. 또한 협력 융합 연구의 축적된 경험을 바탕으로, 현재 대용량 데이터 처리 플랫폼을 생물학뿐만 아니라 다양한 분야에 적용하여 활용 가능하도록 발전시킬 수 있을 것이다.

## 참고문헌

[1] J. Shendure, H. Ji. 2008. Next-generation DNA sequencing. Nat. Biotechno. 26:1135-1145

[2] E. R. Mardis. 2008. Next-generation DNA sequencing methods. *Annu. Rev. Genomics Hum. Genet.* 9:387-402

[3] 허철구, *클라우드 컴퓨팅을 이용한 유전자 서열 데이터의 contig assembly에 관한 연구*, 한국생명공학연구원, 2010

[4] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," OSDI 2004

[5] Hadoop. http://hadoop.apache.org/

[6] BLAST. http://blast.ncbi.nlm.nih.gov/Blast.cgi

[7] J. Cohen, "Graph twiddling in a MapReduce world," *Computing in Science and Engineering*, July/August 2009