클라우드 컴퓨팅 기반의 병렬 CNV 검출 알고리즘

홍상균*, 윤지희*, 이은주**
*한림대학교 컴퓨터공학과
**한림대학교 전자공학과
e-mail:hongsk@hallym.ac.kr

Parallel CNV detection algorithm based on Cloud Computing

Sang-Kyoon Hong*, Jee-Hee Lee*, Un-Joo Lee**
*Dept of Computer Engineering, Hallym University
**Dept of Electronic Engineering, Hallym University

요 약

시퀀싱 기술의 발달로 최근에는 비교적 저렴한 비용으로 개인의 유전체 시퀀싱 데이터를 산출할 수 있게 되었다. 하지만 이를 기반으로 하는 기존의 분석 방법은 매우 고가의 컴퓨팅 환경을 요구하기 때문에 분석을 위한 비용이 매우 높은 문제가 있다. 본 논문에서 클라우드 컴퓨팅 환경의 병렬 CNV 검출알고리즘을 제안한다. 제안하는 방법은 모양 기반의 CNV 검출 알고리즘인 CNV_shape을 MapReduce 기법으로 개발한 것으로 시퀀싱 데이터를 레퍼런스 서열에 매핑한 결과로부터 리드 커버리지 (read coverage)를 계산하여 커버리지가 감소하거나 증가하는 일정 길이 이상의 영역을 검출하는 방법이다. 클라우드 컴퓨팅 환경에 적용하고 노드의 밸런싱 유지를 위한 방법으로 파티셔닝 기법을 사용하였다. 또한 실 데이터를 이용한 실험을 통해 제안하는 방법의 효율적 데이터 처리를 보인다.

1. 서론

최근 시퀀싱 기술의 발달로 개인의 유전체 서열 정보를 기반으로 하는 질병 예측 및 치료의 연구 초석이 마련되었다. 2000년대 초 인간 게놈 프로젝트 (human genome project) [1]가 발표될 당시에는 유전체 분석 비용이 30억 달러 이상이 소요되었으나, 최근 시퀀싱 기술의 발달로 2012년 이후 1,000달러 수준의 비용으로 개인의 유전체 시퀀싱 (personalize sequencing)이 가능할 것으로 예측되고 있다 [2].

인간의 유전체는 약 30억 bp (basepair)의 크기를 가지는 DNA 사슬로 이루어져 있다. 서열 정보는 개개인의 차이가 있으며, 이러한 차이는 다양한 크기와 형태를 가지고 있다. 이러한 개인 간의 서열 정보의 차이를 유전적 구조 변이 (genetic structural variation)라고 한다. 유전적 구조 변이에는 다양한 종류가 있으며, 최근 SNP (Single Nucloetide Polymorphism)와 CNV (Copy Number Variation)에 대한 연구가 활발히 진행 중이다.

최근 차세대 시퀀싱 (next-generation sequencing)을 기반으로 하는 유전적 구조 변이에 관한 다양한 연구가 주목 받고 있다 [3]. 차세대 시퀀싱은 유전정보를 담고 있는 혈액 샘플로부터 서열 정보의 짧은 단편을 수 십bp에서 수 천 bp이상을 한번에 대량으로 읽어내는 방법이다. 이러한 서열 정보의 짧은 단편을 리드 (read)라 부른다. 차세대 시퀀싱 기술을 이용한 분석 방법은 비교적 높은 커버리지 (coverage)를 요구하기 때문에 데이터의 크기가 매우 크다는 문제가 있다 [4]. 이러한 데이터의 크기는 테라바이트 이상으로 대용량이며, 일반적인 컴퓨팅 환경에서는 데이터를 효과적으로 저장하고 처리하기 어렵다. 최근 바이오인포매틱스 분야에서는 대용량의 데이터 분석을 위해 클라우드 컴퓨팅 기술을 이용한 데이터 분석 및 소프트웨어 개발에 대한 연구가 진행되고 있다 [5, 6]. 하지만 이러한 연구는 시퀀싱 데이터의 매핑이나 SNP 연구에 국한되어 있는 상황이다. 특히 CNV와 같이 연속적인 영역에 대한 처리는 병렬 처리하기 어렵기 때문에 연구가 미비하다.

본 연구에서는 클라우드 컴퓨팅 환경에서 차세대 시퀀싱 데이터를 이용한 효율적인 CNV 검출 방법을 제안한다. 제안하는 방법은 기 개발한 CNV_shape [7]을 MapReduce [8] 기반의 병렬 알고리즘으로 개발한 것으로 효과적인 병렬 처리와 연속적인 영역 처리를 위해 데이터의 파티셔닝과 파티션간 오버랩 영역을 포함하는 확장 파티션을 이용한다. 또한 실제 차세대 시퀀싱 데이터를 이용하여 제안하는 방식의 유용성을 실험을 통해 검증한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 관련 연구로서 클라우드 컴퓨팅 환경인 Hadoop [8]과 병렬 처리 기법인 MapReduce를 살펴본다. 제 3장에서는 제안하는 병렬 CNV 검출 알고리즘을 제시한다. 제 4장에서는 실험을 통해 제안하는 알고리즘의 유용성을 검증한다. 마지막으로 제 4장에서는 본 논문을 요약하고 결론을 내린다.

이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (No. 2010-0017194)

2. 관련 연구

2.1. Hadoop과 MapReduce 기법

클라우드 컴퓨팅은 네트워크상의 서로 다른 물리적인 위치에 존재하는 컴퓨터들을 가상화 기술로 통합하여 서비스 및 리소스를 제공하는 기술이다. 즉, 기존에 개개의 컴퓨터상에서 사용되던 프로그램들이나 데이터를 수많은 컴퓨터로 구성된 대규모 컴퓨팅 환경 상에 저장하고, PC나 소형 단말기로 필요한 작업을 수행하는 환경이다. 클라우드 컴퓨팅 환경은 수많은 컴퓨터 리소스로 구성되어져 있기 때문에 대용량의 데이터를 분산 저장할 수 있으며, 병렬 처리가 가능하기 때문에 빠르고 안정적인 컴퓨팅 환경을 제공한다 [8].

Hadoop은 클라우드 컴퓨팅 환경의 대표적인 프레임워크로, 더그 커팅이 개발한 오픈소스 기반의 분산 컴퓨팅 플랫폼이다. Hadoop에는 대용량의 데이터 처리를 위한 하둡 분산 파일 시스템 (HDFS; Hadoop Distributed File System)과 데이터베이스인 HBase 등의 다양한 오픈 소스 프로그램이 제공된다. 또한 분산/병렬 처리를 위한 MapReduce 기법을 지원한다.

MapReduce 기법은 클라우드 컴퓨팅 환경에서 분산/병렬 처리를 지원하기 위한 목적으로 개발된 프로그래밍 기법이다. MapReduce는 Map과 Reduce의 단계로 구성되어 있으며, 처리해야하는 작업을 보다 작은 단위로 세분화하여 처리하는 모델을 제공한다. Map과 Reduce에서는 데이터를 키와 값의 쌍으로 만들어 각 단계의 입출력으로 사용한다. Map 단계는 입력된 데이터에서 키와 값의 쌍으로 데이터를 산출한다. 즉, 연산의 처리 대상이 되는 데이터를 보다 작은 연산으로 처리할 수 있게 데이터를 바꾸는 단계이다. Reduce 단계는 Map 단계에서 생성된 데이터를 동일한 키 값을 가지는 값을 취합하여 처리하고 그 결과를 다시 키와 값의 쌍으로 출력한다.

2.2. MapReduce 기반의 분석 도구

대표적인 클라우드 컴퓨팅 기반의 연구로는 CloudBurst와 CrossBow 등이 있다. CloudBurst는 대표적인 서열 매핑 방법인 RMAP을 MapReduce 기법으로 개발한 것으로 레퍼런스 서열을 일정 길이의 시드 (seed)로 분할/생성하고 이와 동일한 크기로 리드를 분할하여 시드를 키로 하는 데이터를 산출한 뒤 시드와 동일한 리드의 단편을 모아서 리드의 단편을 레퍼런스 서열상에서 확장하여 매핑을 수행하는 방법이다. Crossbow는 서열 매핑 도구인 Bowtie와 SNP 추출 도구인 SOAPsnp를 결합하여 제작한 프로그램이다. Crossbow는 클라우드 환경 하에서 병렬로 수행되며, Bowtie를 통해 리드들을 매핑하고, SOAPsnp를 통해 SNP를 검출하여 나온 결과들을 최종 결과 값으로 산출한다. 그러나 이들 클라우드 컴퓨팅 기반의 연구는 시퀀싱 데이터의 매핑이나 SNP 검출과 같은 분야에 국한되어있다.

3. CNV_shape과 병렬 CNV 검출 알고리즘

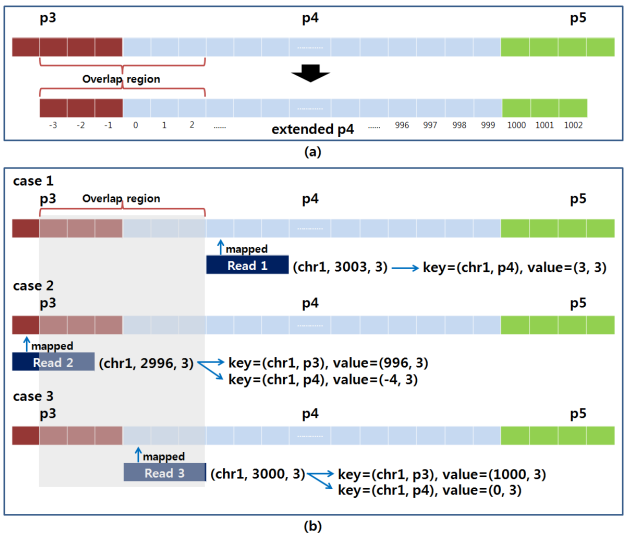
본 장에서는 기 연구된 CNV_shape을 설명하고, CNV_shape을 클라우드 컴퓨팅 환경으로 확장한 병렬 CNV 검출 알고리즘을 기술한다.

3.1. CNV_shape

본 연구실에서는 기존에 모양 기반의 CNV 영역 검출 모델인 CNV_shape을 제안한바 있다. 제안하는 모델에서는 리드의 커버리지 상에서의 CNV 영역의 이상적인 모양을 정의하고 이와 유사한 형태를 가지는 영역을 찾아 검출해내는 방식이다. 이상적인 CNV 영역은 커버리지 값이 평균 커버리지 값에 비하여 일정 크기 이상 증가 혹은 감소된 상태로 연속적으로 형태가 유지되고, 증가 혹은 감소된 영역의 커버리지 값이 비교적 안정되어 큰 변화를 나타내지 않는 영역이다. 제안한 알고리즘에서는 리드와 레퍼런스 서열에 포함된 에러의 영향을 줄이고 제안한 모양의 CNV를 검출하기 위하여 연속적인 쉬프팅 윈도우를 사용하는 mean shift, mean slope 변환을 수행한다. 두 가지 변환을 거친 데이터로부터 정의된 CNV 모양과 유사한 CNV 후보 영역을 검출해낸다. 다음, 검출된 CNV 후보 영역을 통계적, 경험론적 방법에 의하여 병합하여 최적의 CNV 영역을 보고한다.

3.2. 병렬 CNV 검출 알고리즘

MapReduce 환경에서 CNV 검출은 크게 2개의 스테이지로 구성된다. 첫 번째 스테이지에서는 리드 매핑 결과로부터 커버리지를 계산하고, 커버리지에서 mean shift와 mean slope 변환을 수행한다. 두 번째 스테이지에서는 첫 번째 스테이지에서 계산된 mean shift, mean slope 데이터를 콘티그 단위로 나누어 각 콘티그의 mean shift, mean slope의 분포를 계산하고 분포에 따른 임계값을 계산하여 CNV를 검출한다.



(그림 1) 확장 파티션과 파티셔닝 과정

첫 번째 스테이지에서는 리드 매핑 결과로부터 커버리지 및 mean shift, mean slope 변환을 수행한다. 병렬 수행 단위를 염색체 단위로 하는 경우, 염색체 크기가 매우 크기 때문에 주기억 장치의 제약이 있다. 또한 각 분산 노드가 염색체 길이에 따라 편중된 데이터를 처리할 확률이 높아지기 때문에 로드 밸런싱의 문제가 발생하게 된다. 이러한 문제를 해결하기 위하여 각 염색체의 영역을 일정 크기의 파티션으로 분할하여 모든 노드가 유사한 데이터 양을 처리하도록 한다. 하지만 각 파티션의 mean shift, mean slope를 계산하기 위해서는 각 파티션의 이전 파티

션과 이후 파티션에서 윈도우 크기에 비례한 데이터가 필요하다. 따라서 각 파티션은 이웃 파티션과 오버랩 될 수 있게 지정된 파티션 크기보다 윈도우 크기만큼 확장하여 영역을 지정하여야 한다.

다음의 그림 1은 윈도우 크기가 $w=6$이고 파티션 크기를 $p=1000$으로 하는 경우의 실제 파티션 영역 설정 과정을 도식화한 것이다. 파티션 p4에 대한 mean shift와 mean slope 변환을 계산하기 위해서는 파티션 경계 영역에서 윈도우 크기의 절반인 $w/2$의 영역을 이전 파티션과 이후 파티션에서 참조해야한다. 즉, 파티션 p4를 계산하기 위해 그림 1의 (a)와 같이 파티션 p3와 p5의 일부를 오버랩시켜 확장된 p4를 만들어야 한다. 따라서 매핑된 리드를 파티션 단위로 처리하기 위해서는 매핑된 위치가 어느 파티션에 해당하는지 계산하여 해당 파티션에서 계산될 수 있도록 키에 파티션 번호를 저장한다. 이때 각 파티션에는 오버랩되는 영역이 존재하기 때문에 오버랩되는 영역에 매핑되는 부분이 있는 리드의 경우 두 파티션 모두에 저장한다. 그림 1의 (b)는 리드의 매핑 위치에 따른 3가지 파티셔닝 과정을 보인다. Case 1은 매핑된 리드가 오버랩 영역에 속하지 않는 경우로 매핑 정보에서 염색체 번호에 파티션 번호를 추가하여 키로 사용하고 매핑된 위치를 파티션 내의 위치로 변경하여 매핑된 길이와 함께 값으로 저장한다. 그림의 예를 살펴보면 read 1은 염색체 1번의 3003번째에 매핑되었고 길이가 3이기 때문에 오버랩 영역에 속하지 않는다. 따라서 키로 염색체 1번과 파티션 번호 4를 지정하고, 값으로 파티션 4번상의 위치와 길이를 지정한다. Case 2와 3은 매핑된 리드가 오버랩 영역에 속하는 경우로 매핑된 리드 정보를 복사하여 p3와 p4의 두 파티션에 저장한다. 그림의 예를 살펴보면 read 2와 3은 각각 p3와 p4의 오버랩 영역에 매핑 되었기 때문에 각 리드마다 p3와 p4에 속하는 두 개의 정보를 생성한다.

알고리즘 1 : Calculate Coverage and mean shift/slope Transformations

Input : Mapping result MR. Window size w
Output : set of Coverage COV, set of mean shift transform MSH, set of mean slope transform MSL

function MAP(K1, V1, List<K2, V2>)

```
1.. part = calculatePartition(V1.chr, V1.pos);
2.. K2.set(V1.chr, part);
3.. V2.set(V1.pos, V1.len);
4.. List.add(K2, V2);
5.. if( OverlapTest(chr, pos) ) then
6..   K2.set(chr, Neighbor partition of part);
7..   V2.set(pos, len);
8..   List.add(K2, V2);
9.. return List of <K2, V2>;
```

function REDUCE(K2, List<V2>, List<K3, V3>)

```
1. Initialize Coverage cov;
2. CalculateCoverage(cov, List<V2>);
3. msh = MeanShiftTransform(cov);
4. msl = MeanSlopeTransform(cov);
5. for each idx from w/2 to length of cov - w do
6.   K3.set(K2.chr, K2.part, idx);
7.   V3.set(msh[idx], msl[idx]);
8.   List.add(K3, V3);
9. return List of <K3, V3>;
```

스테이지 1의 알고리즘은 <알고리즘 1>과 같다. Map 단계에서는 입력된 SOAP 매핑 결과를 키와 값의 쌍으로 읽어 들인다. 이때 파일내의 줄 번호가 키 K1이 되고 각 줄의 내용이 값 V1이 된다. 위에서 설명한 파티셔닝 과정에 따라 입력되는 매핑결과에서 커버리지 계산을 위해 매핑된 염색체 번호, 염색체 내에서의 매핑된 위치, 매핑된 리드의 길이 등의 정보를 읽어 키 K2와 값 V2의 데이터를 생성한다. Reduce 단계에서는 키 K2의 염색체 번호와 파티션 번호에 의해 데이터가 취합되며, 각 Reduce 함수들에 의해 각각의 파티션의 커버리지를 계산하고, 계산된 커버리지를 이용해 mean shift와 mean slope를 계산하여 저장한다. 이때 각 파티션에서 오버랩 영역에 의해 확장된 부분은 mean shift와 mean slope의 계산을 위해 중복 산출된 영역이기 때문에 각 파티션의 확장 영역을 제외하고 키 K3와 값 V3로 산출하여 저장된다. K3에는 염색체 번호, 파티션 번호, 파티션 내 위치가 저장되며, V3에는 각 위치의 mean shift와 mean slope 변환 값이 저장된다.

알고리즘 2 : CNV Detection

Input : Mean transform of Coverage MT, Contig info CI
Output : set of CNV regions CNV

function MAP(K1, V1, List<K2, V2>)

```
1. for each Chromosome chr, Mapped position pos, Mean
   shift value mshv, Mean slope value mslv in V1 do
2.   Contig = GetContigID(CI, chr, pos);
3.   K2.set(chr, Contig);
4.   V2.set(pos, mshv, mslv);
5.   List.add(K2, V2);
6. return List of <K2, V2>;
```

function REDUCE(K2, List<V2>, List<K3, V3>)

```
1. Initialize mean shift transform of coverage MSH, mean
   slope transform of coverage MSL
2. MSH.setValue(V2);
3. MSL.setValue(V2);
4. cCNV = DetectCNV(MSH, MSL);
5. CNVs = MergeCandidateCNV(cCNV);
6. for each CNV in CNVs do
7.   K3.set(K2);
8.   V3.set(CNV.pos, CNV.length, CNV.copy_type);
     List.add(K3, V3);
9. return List of <K3, V3>;
```

스테이지 2에서는 스테이지 1에서 계산된 mean shift, mean slope 데이터를 이용하여 CNV 영역을 검출한다. CNV 검출 과정에서는 각 콘티그의 모든 mean shift와 mean slope 변환 데이터가 필요하기 때문에 스테이지 1과 같이 로드 밸런싱을 위하여 파티셔닝 할 수 없다. 하지만 각 염색체는 보다 작은 콘티그 단위로 이루어지기 때문에 각 콘티그 단위로 처리하는 특성을 통해 노드 간 로드 밸런싱을 유지할 수 있다.

스테이지 2의 알고리즘은 <알고리즘 2>에 보인다. Map 단계에서는 스테이지 1의 출력으로부터 염색체별로 계산되어진 mean shift, mean slope 데이터를 콘티그 단위로 나눈다. 키 K2는 염색체 번호와 콘티그 ID를 사용한다. 값 V2는 각 콘티그 영역에서의 위치 값과 mean shift, mean slope 변환 값을 사용한다. 이때 콘티그 분할을 위

해 별도로 입력된 콘티그 정보를 이용한다. Reduce 단계에서는 각 콘티그 단위로 취합된 mean shift, mean slope 데이터를 CNV_shape의 알고리즘으로 작성된 DetectCNV() 함수를 이용하여 각각의 분포를 계산하고 임계값을 산출한 뒤, 임계값에 따라 모양 기반의 CNV 후보를 검출한다. CNV 후보로부터 각 영역을 병합 및 보정의 작업을 수행하여 최종 검출된 CNV 영역을 저장한다. 최종 출력인 키 K3는 염색체 번호와 콘티그 ID를 사용하고, 값 V3는 검출된 CNV의 위치와 길이, copy loss와 copy gain 정보를 사용한다.

4. 실험 결과

본 장에서는 제안하는 방법론의 효용성을 실험을 통해 검증한다. 4.1절에서는 실험 환경 및 방법에 대하여 기술하고, 4.2절에서는 실험 결과를 분석, 평가한다.

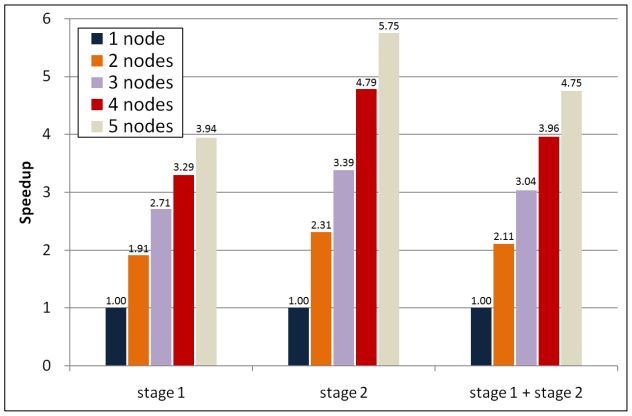
4.1. 실험 방법

본 실험에서는 클라우드 컴퓨팅 환경에서 실 데이터를 사용하여 다수의 노드를 사용하여 성능을 평가하였다. 실험 데이터는 1000 genome project (http://www.1000genomes.org)에서 다운로드한 NA10851의 시퀀싱 데이터를 NCBI Build 36.3의 레퍼런스 서열에 SOAP2 [9]로 매핑한 결과에서 염색체 1, 2, 3번의 데이터를 사용하였으며, 데이터의 크기는 약 16.5 GB이다.

실험에 사용된 클라우드 환경으로는 CentOS 5.5와 Hadoop 0.20.2를 사용하였고, 실험에 사용된 각 노드는 2 GB의 주기억 장치와 200 GB의 보조 기억 장치, Intel Pentium IV 3 GHz의 중앙 처리 장치를 가지며, 개발 언어로는 JAVA를 사용하였다.

4.2. 실험 결과 및 평가

본 실험에서는 클라우드 컴퓨팅 환경에 참여하는 노드 수의 증가에 따른 성능 향상을 실험하였다. 노드의 수는 단일 노드부터 5개의 노드까지 증가하며 실험하였으며, 제안하는 알고리즘의 각 스테이지와 전체 수행 시간을 측정하였다.



(그림 2) 노드 증가에 따른 연산 속도 증가

그림 2는 노드 수의 증가에 따라 단일 노드의 연산 속도의 상대적인 성능 향상을 그래프로 표현한 것이다. 실험은 스테이지 1과 2의 수행 시간을 따로 측정하였으며, 두 스테이지가 수행되는 시간의 합을 알고리즘 전체의 수행 시간으로 산출하였다. 그래프를 보면 전체적인 성능은 노드의 증가에 따라 일정하게 늘어남을 알 수 있다. 특히 단일 노드에 비해 하나 이상의 노드가 추가 되었을 경우 성능 향상의 폭이 큼을 알 수 있다. 각 스테이지 별로 살펴보면, 스테이지 1의 경우에는 노드의 증가에 따라 일정한 수준으로 성능이 향상되는 것을 알 수 있다. 반면에 스테이지 2는 노드 증가에 따른 성능의 향상 폭이 이론적인 성능 향상 폭을 넘어서고 있다. 이는 각 Reduce가 CNV를 검출하는 과정에서 각 콘티그의 전체 데이터를 가지고 처리해야하기 때문에 동시에 처리해야하는 데이터양이 많아서 노드가 적을수록 각 노드의 부담이 크기 때문이다. 이러한 데이터가 분산되면서 스테이지 1에 비하여 큰 성능 향상을 나타내는 것이다. 하지만 노드 수를 3, 5로 증가시킨 경우 2, 4에 비해 향상 폭이 낮음을 알 수 있다. 이는 콘티그들의 길이가 짧게는 수십 kbp에서 길게는 약 100 Mbp로 다양하기 때문에 각 노드가 처리하는 데이터양이 다소 차이가 나면서 발생하는 문제다.

5. 결론 및 향후연구

본 논문에서는 클라우드 컴퓨팅 환경에서 CNV를 검출하는 방법론을 제안하였다. 제안하는 방법론은 모양 기반의 CNV 검출 알고리즘인 CNV_shape을 MapReduce 기법을 적용하여 병렬화한 알고리즘으로 MapReduce 기법에 최적화하기 위하여 검출 대상이 되는 리드 커버리지를 파티셔닝하고 오버랩 영역을 적용하였다. 또한 제안한 방법론의 효용성을 보이기 위해 실 데이터를 사용한 성능 분석 실험을 수행하였다. 금후 실험을 통해 발견된 문제를 해결하고 보다 다양한 실험을 통한 성능 개선과 대규모 클라우드 컴퓨팅 환경에서 다수의 개인 시퀀싱 데이터의 유전체 전체 영역의 실험을 수행할 예정이다.

참고문헌

[1] http://www.genome.gov/10002192

[2] F. S. Robert, "The Race for the $1000 Genome," SCIENCE, Vol. 311, pp. 1544-1546, 2006.

[3] J. Kim et al., "A highly annotated whole-genome sequence of a Korean individual," Nature, Vol. 460, No. 7258, pp. 1011-1015, 2009.

[4] S. Yoon et al., "Sensitive and accurate detection of copy number variants using read depth of coverage," Genome Research, Vol. 19, pp. 1586-1592, 2009.

[5] M. C. Schatz, "CloudBurst: Highly Sensitive Read Mappling with MapReduce," Bioinformatics, Vol .25, No. 11, pp. 1363-1369, 2009.

[6] Langmead et al., "Searching for SNPs with cloud computing," Genome Biology, Vol. 10, No. 11, pp. R134.1-10, 2009.

[7] S. Hong et al., "Shape-based retrieval of CNV regions in read coverage data," International Conference on Bioinformatics, 2010.

[8] T. White, "Hadoop: The Definitive Guide," O'REILLY, 2009.

[9] R. Li et al., "SOAP2: an improved ultrafast tool for short read alignment," Bioinformatics, Vol. 25, No. 15, pp. 1966-1967, 2009.