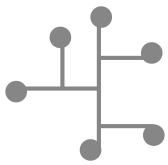


그래프 기반 분산처리 시스템 트리니티를 이용한 서열 정렬 알고리즘

SAG: Sequence Alignment Algorithm based on Graph with distributed system Trinity

이준수(Jun-Su Lee)[1] 여윤구(Yun-Ku Yeu)[2] 노홍찬(Hong-Chan Roh)[3]
윤영미(Young-Mi Yoon)[4] 박상현(Sang-Hyun Park)[5]

요 약

유전체학(Genomics)에서 서열정렬은 가장 널리 사용된다. 차세대 시퀀싱(Next Generation Sequencing) 기술이 발전하면서, 최근 서열 리드 데이터의 양이 급격하게 증가했다. 급증한 차세대 시퀀싱 데이터를 처리하기 위한 서열정렬 알고리즘이 많이 개발되었다. 하지만 서열정렬 알고리즘들은 반복서열(repeat), 변이(polymorphism)를 처리하기 위해 많은 계산량을 요구한다. 그렇기 때문에 기존 서열정렬 알고리즘은 처리량(throughput)과 정렬품질(quality)사이에 트레이드오프(trade-off)가 존재한다. 하지만 분산처리 시스템 Hadoop, Trinity에서 동작하는 정렬 알고리즘은 기존 싱글에서 동작하는 알고리즘에 비해 정렬 품질을 덜 희생하고, 더 높은 처리량을 얻을 수 있다. 본 논문에서는 Microsoft에서 제안한 그래프 기반 인-메모리(in-memory) 분산시스템 트리니티(Trinity)에서 동작하는 서열정렬 알고리즘 SAG(Sequence Alignment Algorithm based on Graph with Trinity)를 제안한다. 우리는 기존 참조 서열을 그래프 형태의 데이터로 변형 한 뒤, 그래프에서 연결 가능한 인접한 노드에 새로운 간선을 추가했다. 그리고 변이(polymorphism)를 허용하는 정렬을 수행하기 위해 서열조각들 사이의 조합을 통해 후보를 얻었다. 마지막으로 후보를 대상으로 glocal alignment를 수행해 최종적인 결과를 찾았다. 실험을 통해 SAG는 기존 Hadoop에서 동작하는 알고리즘과 비교했을 때 비슷하거나 더 좋은 정렬 품질조건과 동시에 상당히 높은 처리량을 얻었다. 또한 머신을 추가함으로써 더 좋은 처리량을 얻는 확장성을 입증하였다.

주제어: 서열정렬 알고리즘, 그래프, 분산처리시스템, 차세대 시퀀싱

---

1 연세대학교 컴퓨터과학과, 석사과정.
2 연세대학교 컴퓨터과학과, 박사과정.
3 연세대학교 컴퓨터과학과, 박사과정.
4 가천대학교 컴퓨터과학과, 교수.
5 연세대학교 컴퓨터과학과, 교수, 교신저자.
† 이 논문은 2012년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(2012R1A2A1A01010775).
+ 논문접수: 2014년 02월 24일, 심사완료: 2014년 03월 16일, 게재승인: 2014년 04월 02일.

Abstract

Sequence alignment is one of the widely used tools in genomics. Recently, after NGS(Next Generation Sequencing) technology was developed, the production of sequence read data increased dramatically. A number of sequence alignment algorithms have been developed for processing these NGS data. However, these algorithms are suffered from a trade-off between throughput and alignment quality, because there is a large computation cost for handling the repeat reads and polymorphism. On the contrary, alignment algorithms with distributed system such as Hadoop and Trinity can obtain better throughput without compromising alignment quality than existing algorithms on single machine. In this paper, we suggest SAG, sequence alignment algorithm based on graph with in-memory distributed system, Trinity proposed by Microsoft. We transformed reference sequence into a graph form, and added new edge between adjacent node having connection possibility on graph. And we performed combination of sequence fragments in order to candidates allowing polymorphism. Finally, we performed glocal alignment to find final results for the obtained candidates. Our experimental results show that SAG better throughput with same quality or better quality than existing algorithms with Hadoop. We have also proved scalability that we obtained better throughput by simply adding machines.

Keywords: sequence alignment algorithm, graph, distributed system, NGS (next generation sequencing)

1. 서 론

유전체(Genome)는 생명 현상의 정보를 담고 있는 가장 기존적인 물질이다. 유전체 내부의 유전자(gene)는 RNA와 단백질(protein)형태를 거쳐 생명체를 구성하거나 기능을 수행한다. 또한 유전체는 조상으로부터 유전 정보를 전달하고, 다양한 원인으로 발생한 변이(polymorphism)를 누적함으로써 생물의 진화에서 중요한 역할을 수행하기도 한다.

2003년 게놈 프로젝트(Genome project) 이후, 유전체 정보의 해석 기술은 급속도로 발전하였다. 초창기 인간의 전체 유전체를 해석하는데 천문학적인 금액과 시간이 소모되었지만, 이른바 차세대 시퀀싱(Next Generation Sequencing) 기술이 급속하게 발전함에 따라 해석에 필요한 시간과 금액도 급격히 감소하였다.

이러한 유전 정보 해석 기술의 발전과 함께 염기서열 정렬(sequence alignment) 알고리즘의 중요성 역시 부각되고 있다. 염기 서열 정렬 알고리즘은 유전체 재조립(genome re-sequencing), 유전체 변이의 탐색, TFBS(Transcription Factor Binding Site) 의 탐색, 그리고 새롭게 발견된 유전자나 단백질의 기능 탐색 등 생물학 전반에서 널리 사용되는 알고리즘이다.

차세대 시퀀싱 기술을 이용해 짧은 길이의 염기서열인 리드 형태로 대량 생산 할 수 있으며, 이 리드를 참조 유전체에 정렬하고 차이점을 비교함으로써 해석하고자 하는 염기 서열을 빠르고 저렴하게 할 수 있다. 이처럼 많은 리드의 생산량이 폭발적으로 증가하고 있기 때문에 이를 감당 할 수 있는 높은 처리량을 갖춘 서열 정렬 알고리즘이 필요하다.

서열 정렬 알고리즘은 두 개의 시퀀스 사이에 진화적인 관계 또는 기능, 구조적으로 유사한 지역(region)을 확인하는 알고리즘을 의미한다. 알고리즘으로서 갖추어야 할 조건은 크게 처리량(throughput)과 정렬품질(quality)로 압축 할 수 있다. 정렬 알고리즘은 싱글에서 동작하는 SOAP2[1], BWA[2], RMAP[3,4], 그리고 Bowtie[5] 등 다양하게 등장하였다. 하지만 서열정렬 알고리즘은 변이(polymorphism), 반복서열(repeat)을 허용하게 되면 계산량이 크기 때문에 기존 싱글머신에서 동작하는 알고리즘은 일정 정렬 품질을 포기하고, 높은 처리량을 얻기 위해 고유한 휴리스틱한(heuristic) 방법을 사용해 문제를 해결하였다. 하지만 컴퓨팅 파워가 발전하고, 다양한 분산처리 시스템 플랫폼이 등장함에 따라 정렬 품질을 덜 포기하고 더 높은 처리량을 얻을 수 있게 되었다.

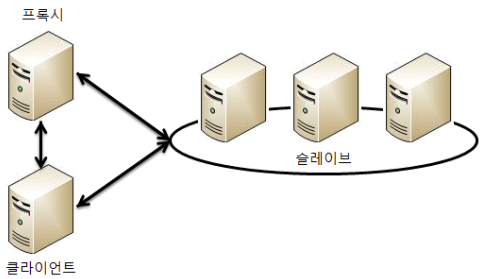
최근 Hadoop(http://hadoop.apache.org/), Trinity[6, 7] 등 다양한 분산시스템이 플랫폼이 개발되었다. Hadoop은 Google에서 제조된 오픈소스 프레임워크를 사용해 병렬 프로그래밍을 제공한다. 그러나 Hadoop은 Map/Reduce[8] 단계에서 중간 결과를 저장하기 때문에 Disk I/O가 발생한다. 그렇기 때문에 높은 처리량을 요구하는 알고리즘에는 적합하지 않다. 반대로 Trinity는 Microsoft에서 제공되는 분산처리 시스템으로 인-메모리 클러스터를 구성하고, 그래프 형태의 데이터를 병렬적으로 처리하면서 높은 처리량을 얻을 수 있기 때문에, 서열정렬 알고리즘과 같이 높은 처리량을 요구하는 알고리즘에 적합하다.

본 논문에서는 서열정렬 문제를 해결하기 위해 분산처리 시스템 Trinity에서 동작하는 SAG(Sequence Alignment based on Graph)를 제안한다. SAG는 총 2단계로 이루어져 있다. 첫 번째로 기존 참조 서열을 de-bruijn[9]과 유사한 그래프 형태의 데이터로 변환하고, 그래프를 각 슬레이브의 메모리 클러스터에 빌드 한다. 두 번째로 서열 알고리즘, SAG를 수행하기 위해, 각 리드에 대해 리드 그래프를 추출한 후, 그래프에서 연결 가능한 인접한 노드에 새로운 간선을 추가하였다. 그리고 변이(polymorphism)을 허용하는 정렬을 수행하기 위해 노드 사이의 조합을 통해 얻은 후보를 대상으로 glocal alignment를 수행해 최종적인 후보를 찾았다. 마지막으로 각 슬레이브에서 얻은 결과를 클라이언트(프록시)단계에서 종합하여 최종적인 결과를 얻었다.

본 논문은 다음과 같은 구성으로 이루어져 있다. 2장에서는 Trinity와 기존 서열정렬 알고리즘에 대한 간략한 설명을 하고, 3장에서는 본 논문에서 제시하고 있는 염기서열 정렬알고리즘, SAG에 대해서 설명하고, 4장에서는 실험 및 결과를 보여준다. 마지막으로 5장에서는 본 논문의 결론을 보여준다.

2. 관련연구

2.1 Trinity



[그림 1] Trinity 아키텍처.

[그림 1]은 Trinity를 간략하게 설명한 그림이다. Trinity는 노드의 역할에 따라 3개의 타입, 클라이언트, 슬레이브, 프록시로 나누어진다. 클라이언트는 슬레이브 또는 프록시로 메시지를 보내고, 최종 결과를 받는 역할을 한다. 슬레이브는 그래프 형태의 데이터를 저장 할 뿐만 아니라 실제 알고리즘이 동작하는 역할을 한다. 마지막으로 프록시는 슬레이브로부터 결과를 받고, 부분적인 결과를 통합해 최종적인 결과를 클라이언트에게 전송해준다.

2.2 기존 서열정렬 알고리즘

차세대 시퀀싱 기술이 발전한 후, 많은 리드가 생산되었고, 그 데이터를 처리하기 위한 서열정렬 알고리즘 SOAP2[1], BWA[2], RMAP[3,4] 그리고 Bowtie[5] 등 여러 가지 알고리즘이 개발되었다. 그러나 진핵생물의 유전체에 광범위하게 분포하는 반복서열 (repeat)과 유전 과정과 돌연변이로 발생하는 치환(substitution), 삽입(insertion), 결손(deletion) 그리고 중복(duplication)으로 인해 최적의 정렬 결과를 찾기 위해서는 많은 계산량이 필요하였다. 이 때문에 기존의 알고리즘들은 최적의 정렬 결과를 찾는 대신, 고유의 휴리스틱을 적용하여 높은 처리량을 획득하였다. 그러나 이러한 접근 방법은 필연적으로 정확도와 처리량 사이에 트레이드오프(trade-off)를 갖게 된다. 그렇기 때문에 기존 알고리즘은 일정 정렬 품질을 희생하고 높은 처리량을 얻기 위해 노력했다. 하지만 컴퓨팅파워가 발전하고 분산처리 시스템이 등장함에 따라 더 이상 싱글머신이 아닌 다양한 방법으로 처리 할 수 있다.

최근 많이 사용되는 분산처리 시스템인 Hadoop을 사용해 서열정렬 문제를 해결하는 알고리즘, CloudBurst[10] 와 CloudAligner [11]가 존재한다. 두 알고리즘은 기존 싱글머신에서 동작하는 방법과는 다르게 분산처리 시스템 Hadoop을 사용해 병렬적으로 서열정렬을 수행하였다. CloudBurst와 CloudAligner는 seed-and-extend방법을 사용하는 RMAP과 유사한 방법으로 Map/Reduce를 사용하여 서열정렬 문제를 해결하였다. 그러나 두 알고리즘은 Map/Reduce 단계에서 중간결과를 저장하기 때문에 Disk I/O가 발생한다. Map/Reduce를 사용하는 Hadoop은 테라바이트 급 이상의 빅 데이터를 처리하는데 적합하기 때문에 기가바이트 급의 처리량을 요구하는 서열정렬 문제를 해결하기에는 적합하지 않다. 결과적으로 두 알고리즘은 Hadoop을 사용해 높은 처리량을 얻을 수 없었다.

3. 방법

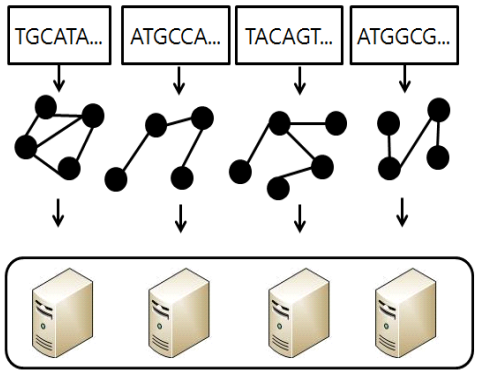
SAG(Sequence Alignment based on Graph)는 2개의 단계로 이루어져 있다. 첫 번째는 각 슬레이브는 기존 참조 서열 데이터를 그래프 형태의 데이터로

변환한 뒤 해당 데이터를 자신의 메모리에 빌드 한다(3.1). 두 번째로 클라이언트는 각 슬레이브에 리드 퀴리를 보내고, 각 슬레이브는 서열정렬 알고리즘, SAG를 각 리드 서열에 대해서 수행한다(3.2). 마지막으로 각 슬레이브의 결과를 클라이언트(프록시) 단계에서 통합한다.

3.1 그래프 배치

각 슬레이브에 그래프를 배치하기 위해, 우리는 2가지의 서브 단계를 진행한다. 첫 번째로는 각 슬레이브는 참조 서열을 나누어 그래프 형태로 변환시킨다(3.1.1). 두 번째로 자신에게 할당된 서열을 그래프 형태로 변환시키는 단계이다(3.1.2).

3.1.1 참조 서열 나누기



[그림 2] 참조 서열 나누기.

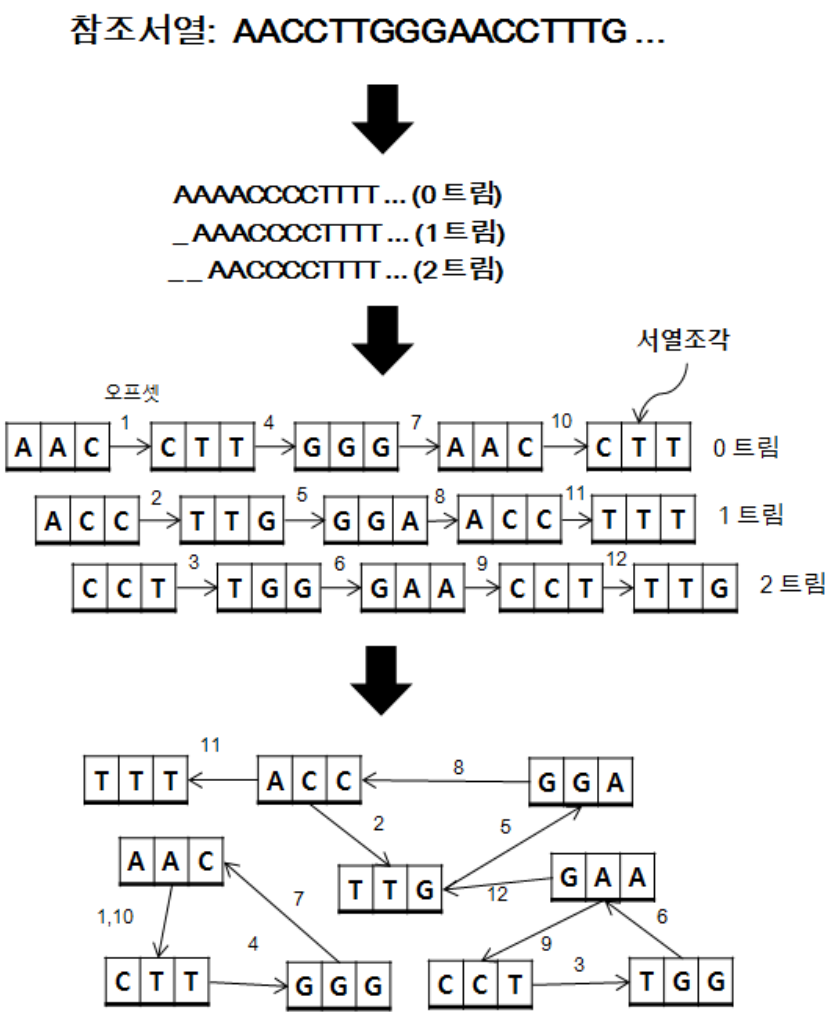
클라이언트로부터 메시지를 받으면, [그림 2]와 같이 모든 슬레이브는 전체시퀀스를 슬레이브 개수로 일정하게 나누어 각 슬레이브에 할당된 참조 서열을 부분을 자신의 메모리에 불러온다. 우리는 각 슬레이브에서 생성되는 그래프의 노드의 이웃을 줄이기 위해 각 염색체 별로 나누어 그래프를 구성하였다. 그 결과 노드의 개수는 늘어나지만, 노드가 보유하는 간선의 오프셋의 개수는 줄어들기 때문에 노드 사이의 비교연산을 줄일 수 있다. 우리는 많은 노드(인덱스)를 허용할 수 있는 Trinity의 장점을 사용하기 위해 이처럼 그래프를 구성하였다.

3.1.2 참조 그래프 생성

우리는 기존 나누어진 참조 서열을 de-bruijn과 유사한 그래프 형태의 데이터로 변환하였다. 그림 3은 참조 시퀀스를 그래프 형태 변환하는 과정을 나타낸 그림이다. 우리는 참조 서열을 *k*-1번 (*k*: 서열 조각의 길이) 트림(trim)하고, 다음 그래프 형태의 데이터로 변환하기 위해 트림된 서열을 *k*-mer크기의 서열 조각으로 나누었다. 여기서 *k*-mer는 길이가 *k*인 시퀀스를 의미하고, 그래프의 노드로서 각 슬레이브에서 빠르게 노드로 접근 할 수 있는 인덱스로서의 역할을 한다. 간선은 참조 서열에서 인접하는 *k*-mer 노드의 관계를 나타내고, 참조 서열에서의 위치 정보인 오프셋을 포함 하고 있다. 예를 들어, [그림 3]에서 AAC→CCT 간선은 1, 10의 오프셋을 가지고 있고, 1은 참조 서열에서 0~5의 위치를, 10은 9~14의 위치를 각각 나타내고 있다.

3.2 서열정렬

리드를 처리하기 위한 서열정렬 알고리즘, SAG는 총 2개의 서브 스텝으로 나누어져 있다. 첫 번째로 참조 그래프로부터 리드 그래프를 추출한다(3.2.1). 두 번째 후보를 얻기 위해 연속된 노드에 새로운 간선을 추가한다. 더 나아가 변이를 허용하는 정렬을 수행하기 위해 연속된 노드들의 조합을 통해 얻은 후보에 대해 glocal alignment를 수행해 최고의 점수를 갖고 있는 위치 정보에 대한 결과를 얻는다(3.2.2). 마지막으로 각 슬레이브는 얻어진 부분적인 결과를 클라이언트(프록시) 레벨로 전송을 하고, 클라이언트(프록시)에서는 부분적인 결과를 종합해 최종적인 정렬 결과를 얻는다.



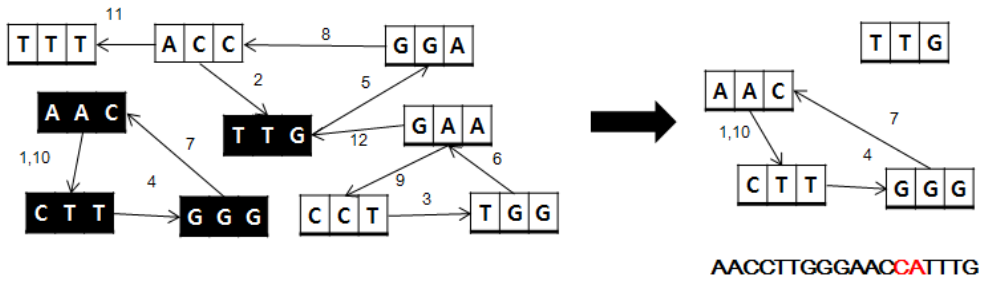
[그림 3] 기존 서열 정렬 데이터를 그래프 형태로 변환하는 과정.

3.2.1 리드 그래프 생성

참조 그래프를 만드는 방법과 유사한 방법을 통해 리드 그래프를 생성한다. 입력 값으로 들어온 리드 서열에 대해서 참조 서열과 동일한 *k*-mer단위로 조각을 내고, 각 *k*-mer 서열 조각에 상응하는 서열을 갖고 있는 노드를 참조 그래프에서 추출한다. 간선은 참조 그래프에 있는 *k*-mer 노드 사이에 있는 간선을 가져와 리드 그래프의 노드사이를 연결시킨다. [그림 4]는 3.1에서 참조 서열'AACCTTGGGAACCTTTG...'로 얻은 참조 그래프에서 리드서열 'AACCTTGGGAACCATTTG'을 이용해 리드 그래프를 추출하는 그림을 보여주고 있다.

3.2.2 SAG 알고리즘

SAG알고리즘은 추출된 리드 그래프 내에서 이웃한 간선의 오프셋 사이에 연결 가능성을 비교한다. 간선

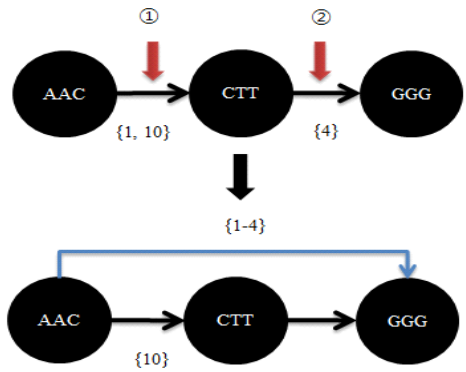


[그림 4] 참조서열 그래프에서 리드 그래프 추출: *왼쪽 그래프는 참조그래프, 오른쪽 그래프는 참조 그래프로부터 추출된 리드 그래프를 나타낸다. 간선 위에 있는 정보는 참조서열에서의 위치를 나타내는 오프셋 정보이다. 빨간 서열 C와 A는 변이(polymorphism)을 의미하고, 참조 그래프에 있는 검정색 노드는 리드 그래프에 상응하는 노드를 나타낸다.*

사이에 연결 가능성이 있으면 [그림 5]와 같이 노드 사이에 새로운 인터벌 간선을 추가하고, 기존 오프셋을 제거하였다. 예를 들어 이웃한 간선 ①번과 ②번 간선의 오프셋 간에 연결 가능성을 비교 하였다. {1, 10}과 {4} 사이에 오프셋 1, 4의 연결 가능성이 존재하기 때문에 우리는 AAC→GGG로 가는 인터벌(interval) 간선을 추가하고 {1-4} 오프셋을 해당 간선에 추가한다. 이때 추가된 오프셋은 기존 간선에서 제거한다. 이렇게 추가된 인터벌 간선은 후보를 찾는 조합 과정의 계산을 줄인다.

우리는 연속된 서열 조각을 찾는 과정에서 발견하지 못한 에러를 허용하는 후보를 찾기 위해 연속된 서열 조각 사이에 조합을 통해 후보를 얻었다. 예를 들어 [그림 6]에서 ①, ②, ③ 은 인터벌(interval) 간선으로 연결된 노드들을 의미한다. 노드들 사이에 존재하는 변이를 허용하는 정렬을 수행하기 위해 ①-②, ①-③ 그리고 ②-③을 대상으로 연결 가능성을 조사한다.

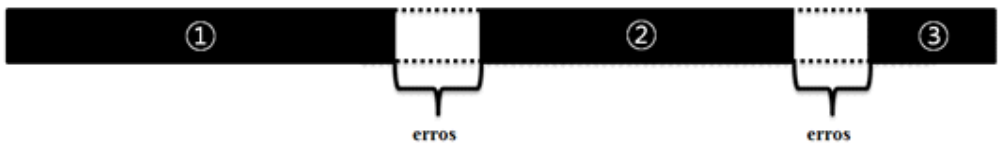
우리는 조합을 통해 얻어낸 후보를 대상으로 가장 높은 점수를 갖는 최고의 후보를 얻기 위해 glocal alignment를 수행한다. glocal alignment는 global alignment[12] 와 local alignment[13]를 합친 알고리즘을 의미한다. 이 두 개의 알고리즘은 동적 프로그래밍(dynamic programming)을 사용해 서열정렬을



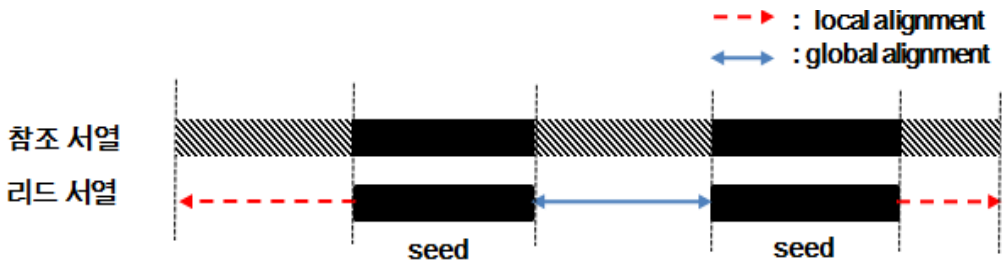
[그림 5] 인터벌(interval) 간선 추가 과정: *검정색 동그라미는 시드(정확한 위치에 정렬된 서열)를 의미한다. 빨간색 화살표는 비교하는 간선을 가리키고 있고, 간선 사이에는 오프셋 정보가 포함되어 있다. 파란색 선은 새롭게 추가되는 인터벌(interval) 간선으로 기존 오프셋을 추가해 새로운 오프셋{1–4}을 포함하고 있다.*

실시한다.

[그림 7]은 glocal alignment의 예제를 보여주고 있다. glocal alignment는 변이(polymorphism)를 포함하는 서열 조각의 위치에 따라 global alignment와 local alignment를 수행한다. global alignment는 시드 (정확하게 정렬된 서열) 사이에 있는 서열 조



[그림 6] 서열 조각 사이에 조합: *검정색 사각형은 이전 인터벌 간선을 추가한 연속된 서열 조각을 의미한다. 즉, 정확하게 매치된 위치를 말한다. 검정색 사각형 사이는 변이(polymorphism)으로 인해 연속되지 못한 서열 조각을 말한다. 해당 위치에는 갭(gap), 미스매치(mismatch)과 같은 에러가 포함되어 있다.*



[그림 7] 삽입, 삭제 그리고 치환과 같은 변이(polymorphism)가 포함된 서열 조각에 대해서 glocal alignment를 수행: *사선 사각형은 갭(gap), 미스매치(mismatch)에 의해 매치되지 않은 구간을 의미한다. 빨간색 점선 화살표와 파란색 양두의 화살표는 local alignment와 global alignment를 각각 의미한다.*

각에 대해서 정렬 할 때 사용하고, local alignment는 맨 앞에 위치한 시드의 앞부분과, 맨 뒤에 위치한 시드의 뒷부분을 정렬할 때 사용한다. 우리는 이처럼 후보마다 점수를 계산해 가장 높은 점수를 갖는 후보를 최고의 후보로 선택했다.

4. 실험 결과 및 토론

4.1 실험 환경

우리는 SAG, CloudBurst 그리고 CloudAligner의 처리량과 정렬품질을 측정하기 위해, Trinity 슬레이브 6개, 클라이언트 1개를 구성하고, 6개의 Hadoop 클러스터를 각각 구성하였다. 각 머신은 1TB HDD, 32GB RAM 그리고 3.40GHz Intel R i73770의 성능을 갖고 있고, Trinity에서 운영체제는 64-bit Window Server 2008 R2 Enterprise, Hadoop에서는 Ubuntu 12.04.2를 각각 사용하였다.

4.2 데이터, 측정 값 설명

우리는 SAM tool package[14] (http://samtools.sourceforge.net/)에 포함되어 있는 dwgsim 프로그램을 사용해 human genome (hg18)의 염색체 21번으로부터 fastq 포맷의 길이 100bp인 10만개, 50만개의 리드 데이터를 생성하였다.

우리는 각 알고리즘의 성능을 측정하기 위해 실행시간과 정확도를 측정했다. 실행시간은 처리량(throughput)을 측정하기 위해 사용했다. 실행시간은 정렬시간, Network I/O 그리고 Disk I/O 시간을 포함한 총 수행시간을 의미한다. 정확도를 계산하기 위해, Recall과 Precision을 사용했다. 두 값을 계산하는 식은 아래와 같다:

$$Recall = \frac{mc}{mc + mu + mi}$$

$$\mathrm{P}recision = \frac{mc}{mc + mi}$$

*mc*는 맵핑(mapping)이 되어야 하는 위치에 정확하게 맵핑되는 리드의 개수를 의미한다. *mi*는 맵핑이 되어야 하는 위치에 정확하게 맵핑되지 않은 리드의 개수를 의미한다. *mu*는 맵핑이 되면 안 되는 위치에 맵핑이 되는 리드의 개수를 의미한다.

4.3 비교 결과 분석

우리는 분산처리 시스템 Trinity에서 동작하는 SAG의 성능을 평가하기 위해 기존 Hadoop에서 동작하는 알고리즘 CloudBurst v1.1.0, CloudAligner v1.9를 사용해 비교했다.

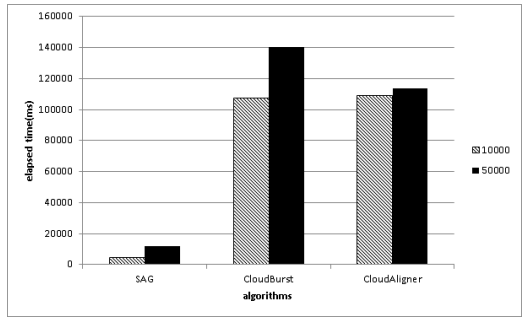
우리는 SAG의 처리량을 비교하기 위해 두 비교 알고리즘과 비슷하거나 더 높은 정렬 품질을 가질 때의 처리속도를 비교했다. 비교 알고리즘 CloudBurst와 CloudAligner는 허용하는 에러의 개수가 늘어날수록 정확한 결과를 보여주는 반면, 낮은 처리량을 보여준다. 그렇기 때문에 세 알고리즘이 비슷한 Recall과 Precision이 0.97이 나오는 에러 개수를 허용할 때를 대상으로 처리량을 비교했다. [표 1]은 세 알고리즘의 정렬 품질의 값을 정리한 내용이다.

[표 1] 정렬 품질 비교.

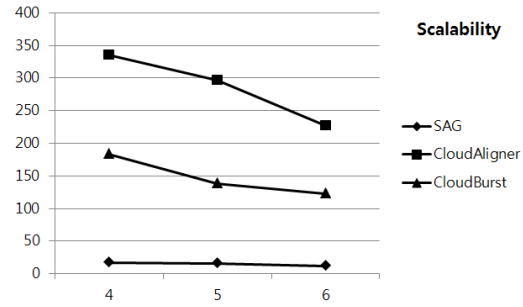
알고리즘	Recall	Precision
SAG	0.97	0.97
CloudBurst	0.61	0.97
CloudAligner	0.97	0.97

SAG는 CloudBurst와 비교했을 때 0.36의 더 높은 Recall과 동일한 Precision, 0.97값을 얻었고, CloudAligner와 비교했을 때는 동일한 Recall과 Precision, 0.97값을 보여주고 있다. 우리는 최종적으로 위와 같이 동일하거나 높은 정렬 품질을 보장할 때의 처리량을 비교했다.

[그림 8]: Trinity에서 동작하는 SAG는 Hadoop에서 동작하는 CloudBurst와 CloudAligner와 비교했을 때 처리량이 최소 9배에서 최대 25배의 높은 값을 보여주고 있다. Hadoop기반 서열정렬 알고리즘은 Map/Reduce 플랫폼을 사용하기 때문에 중간 결과를 저장하는 Disk I/O에 영향을 받는다. 그렇기 때문에 서열정렬 문제와 같이 높은 처리량을 요구하는 알고리즘은 적합하지 않다. 결과적으로 분산처리 시스템, Trinity에서 동작하는 SAG는 동일하거나 더 높은 정렬 품질에서 더 높은 처리량을 얻었다.

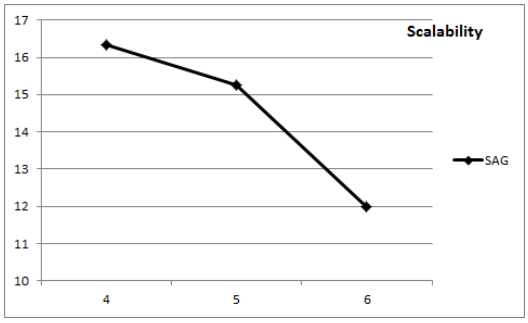


[그림 8] SAG, CloudBurst 그리고 CloudAligner의 처리 속도 비교: *x축은 알고리즘의 종류를 의미하고, y축은 수행시간을 각각 의미한다.*



[그림 9] 머신 개수에 따른 각 알고리즘의 처리 속도 비교.

[그림 9]: 마지막으로 분산처리 시스템 기반 알고리즘의 중요한 확장성을 입증하기 위해 슬레이브의 개수를 4개에서 6개로 증가시키면서 각 알고리즘의 처리 속도를 비교했다. SAG는 두 알고리즘과 비교했을 때 처리량의 차이가 크기 때문에 따로 [그림 10]에 표시했다. 결과적으로 노드의 개수를 추가함으로써 처리속도가 감소하는 것을 볼 수 있었다.



[그림 10] SAG의 머신 개수에 따른 처리속도.

5. 결론

차세대 시퀀싱(Next Generation Sequencing) 기술이 발전한 후에, 서열 데이터의 생산이 급증했다. 그렇기 때문에 급증한 서열 데이터를 처리하기 위해 많은 서열정렬 알고리즘이 등장했다. 하지만 서열정렬 알고리즘은 변이(polymorphism), 반복서열(repeat)과 많은 데이터양을 처리해야하기 때문에 처리량과 정렬품질 사이에 트레이드오프가 존재한다. 하지만 컴퓨팅 파워가 증가하고, 분산처리 시스템이 등장했기 때문에 더 이상 싱글머신에서 처리 할 필요가 없어졌다.

최근 Hadoop은 가장 널리 사용되는 분산처리 시스템이다. 하지만 Map/Reduce 플랫폼을 사용하기 때문에 중간결과를 저장하기 때문에 Disk I/O가 발생하고, 결과적으로 서열 정렬 문제를 처리할 때 높은 처리량을 얻기가 힘들다. 반면 Trinity는 인-메모리(in-memory)기반 클러스터이기 때문에 빠른 처리가 가능하고, 결과적으로 Hadoop보다 더 좋은 처리량을 얻을 수 있다. 이러한 이유로 우리는 Trinity를 사용해 서열정렬 문제를 해결하였다.

본 논문에서 분산처리시스템, Trinity에서 병렬적으로 차세대시퀀스를 처리할 수 있는 염기서열정렬 알고리즘, SAG를 제안했다. SAG는 첫 번째로 기존 참조 서열데이터를 de-bruijn과 유사한 그래프 형태의 데이터로 변환 하였다. 두 번째로 각 리드에 대해서 리드 그래프를 추출 하고, 추출된 리드 그래프 내에 연결 가능한 인접한 간선 사이에 추가적으로 새로운 인터벌 간선을 추가하였다. 변이(polymorphism)을 허용하는 정렬을 수행하기 위해 우리는 연결된 서열 조각 사이에 조합을 수행했고, 이 과정이서 얻은 후보를 대상으로 glocal alignment를 통해 최고의 후보를 얻었다. 마지막으로 각 슬레이브에서 최고의 후보를 클라이언트(프록시)로 전송해 부분적인 결과를 종합해 최종적인 정렬 결과를 찾는다.

우리는 SAG의 처리량과 정렬품질을 평가하기 위해 기존 Hadoop에서 동작하는 알고리즘과 비교했다. 실험 결과 SAG는 비슷하거나 더 높은 정렬 품질에서 상당히 높은 처리량을 보여주었다. 또한 머신의 개수가 늘어날수록 처리 속도가 늘어나는 확장성 또한 입증하였다. 결과적으로 Hadoop이 아닌 분산처리 시스템, Trinity를 사용해 높은 처리량을 얻는 서열 정렬알고리즘 SAG를 제안했다.

참 고 문 헌

[1] R.Li et al, “SOAP2: an improved ultrafast tool for short read alignment”, Bioinformatics Application note, vol.25, no.15, pp. 1966-1967, 2009.

[2] H. Li and R. Durbin, “Fast and accurate short read alignment with Burrows-Wheeler transform,” Bioinformatics, vol. 25, no. 14, pp. 1754-1760, 2009.

[3] Andrew D.Smith, Wen-Yu Chung, Emily Hodges et al, “Updates to the RMAP short-read mapping software“. Bioinformatics, vol. 25, no. 21, pp 2841-2842, 2009.

[4] Andrew D Smith, Zhenyu Xuan, Michael Q Zhang “Using quality scores and longer reads improves accuracy of Solexa read mapping”. MBC Bioinformatics, 9:128, 2008.

[5] B. Langmead, C. Trapnell, M. Pop, and S.L. Salzberg, “Ultrafast and memory-efficient alignment of short DNA sequences to the human genome,” Genome Biology, 10:R25, 2009.

[6] B Shao, H Wang, Y Li “Trinity: A distributed graph engine on a memory cloud.” ACM” SIGMOD, pp. 505-516, 2013.

[7] Trinity Manual by Microsoft Research Asia Copyright © 2012 Microsoft Corporation.

[8] Dean J, Ghemawat S “MapReduce:simplified data processing on large clusters.” Commun ACM 51:107-113, 2008.

[9] N. G. “de Bruijn. A combinatorial problem”, Proc. Kon. Ned. Akad. Wetensch., vol. 49, pp.758-764, 1946.

[10] Schatz M, “CloudBurst: highly sensitive read mapping with MapReduce” Bioinformatics 25(11):1363, 2009.

[11] Tung Nguyen, Weisong Shi, “CloudAligner: A fast and full-featured MapReduce based tool for sequence mapping”. BMC Research Notes 4:171 i54–62. doi:10.1093/bioinformatics/btg1005. PMID 12855437. 2011.

[12] Needleman, S. B. & Wunsch, “A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins” C. D, J. Mol. Biol. 48, 443453. 1970.

[13] T.F. Smith and M.S. Waterman, "Identification of Common Molecular Subsequences," Journal of Molecular Biology vol. 147, pp. 195–197, 1981. gy, vol. 10, issue 3, Article R25, Mar. 2009.

[14] H. Li et al, “The Sequence alignment/map (SAM) format and SAM tools,” bioinformatics, vol. 25, no. 16, pp. 2078-9, June 2009.

이 준 수

2012년 경기대학교 컴퓨터과학과 졸업 (학사)
2012년~현재 연세대학교 컴퓨터과학과 석사과정
관심분야: 바이오인포매틱스, 데이터마이닝, 분산처리시스템.

여 윤 구

2009년 연세대학교 컴퓨터과학과 졸업 (학사)
2011년 연세대학교 컴퓨터과학과 졸업 (석사)
2011년~현재 연세대학교 컴퓨터과학과 박사과정
관심분야: 바이오인포매틱스, 데이터마이닝, 데이터베이스 시스템.

노 홍 찬

2006년 연세대학교 컴퓨터과학과 졸업 (학사)
2008년 연세대학교 컴퓨터과학과 졸업 (공학석사)
2008년~현재 연세대학교 컴퓨터과학과 박사과정
관심분야: 데이터베이스 시스템, 플래시 SSD.

윤 영 미

1981년 서울대학교 자연과학대학 졸업 (학사)
1983년 오하이오 주립대학 수학과 (학사수료)
1987년 스탠포드대학교 컴퓨터과학과 졸업(이하석사)
2008년 연세대학교 컴퓨터과학과 졸업(공학박사)
1987년 5월~1993년 5월 IntelliGenetics Inc.. California, USA Software Engineer
1995년 2월~현재 가천대학교 컴퓨터공학과 교수

관심분야: 데이터베이스 시스템, 데이터마이닝, 바이오인포매틱스, 소셜데이터 마이닝.

박 상 현

1989년 서울대학교 컴퓨터공학과 졸업 (학사)
1991년 서울대학교 대학원 컴퓨터공학과(공학석사)
2001년 UCLA 대학원 컴퓨터과학과(공학박사)
1991년~1996년 대우통신 연구원
2001년~2002년 IBM T. J. Watson Research Center Post-Doctoral Fellow
2002년~2003년 포항공과대학교 컴퓨터공학과 조교수
2003년~2006년 연세대학교 컴퓨터과학과 조교수
2006년~2011년 연세대학교 컴퓨터과학과 부교수
2011년~현재 연세대학교 컴퓨터과학과 교수
관심분야: 데이터베이스, 데이터마이닝, 바이오인포매틱스, 적응적 저장장치 시스템, 플래쉬메모리 인덱스, SSD.