대용량 데이터 처리를 위한 스토리지 I/O 기반 작업 스케쥴러

이준학

고성능바이오컴퓨팅팀
국가슈퍼컴퓨팅연구소
한국과학기술정보연구원

초록

최근 급속도로 발전하고 있는 DNA 시퀀싱 기술과 같이 현대 과학분야에서는 다양한 신기술로 개발된 실험장비로부터 가늠할수 없을 정도로 방대한 데이터가 생산되고 있다. 그러나 이러한 방대한 실험 데이터를 저장, 관리하고 동시에 분석을 수행하는 일은 그에 걸맞은 방대한 양의 컴퓨팅 자원과 메모리 그리고 대용량 데이터 스토리지가 필수적으로 필요하다. 현재 가장 많이 사용되고 있는 작업 관리 시스템은 많은 수의 분석 작업들을 효율적으로 관리하고 실행할 수 있는 환경을 제공하고 있다. 하지만 이러한 작업 관리 시스템들은 컴퓨팅 자원 즉, CPU의 효율적인 사용에 중심을 두고 있으며 대용량 실험 데이터 분석을 위한 Input/Output 혹은 메모리 관점에서 작업을 관리하는 기능을 제공하고 있지 못하다. 따라서 대용량의 실험 데이터를 하나의 컴퓨팅 시스템에서 동시에 실행하게 되는 경우 최대 스토리지 시스템의 I/O 대역폭보다 많은 I/O 요구가 발생하게 되며 이에 따라 전체적인 시스템의 성능저하를 야기하게 될 수 있다. 따라서 본 연구에서는 이러한 현상을 해결하기 위하여 스토리지 시스템의 I/O에 기반한 작업 스케쥴러를 디자인하고 개발하였다. 이 I/O 기반 스케쥴러를 국가슈퍼컴퓨팅연구소에서 운영중이 클러스터 시스템 중 하나의 시스템에 구성하였고 이를 기반으로 대용량 암 전장 유전체 분석연구에 활용하였다. The Cancer Genome Atlas에서 생성한 희귀 신장암 환자 50명의 샘플에 대한 분석을 진행한 결과 기존 스케쥴링 시스템의 경우 대부분의 작업이 멈추거나 상당히 오랜 시간이 걸리게 되는 데에 반해 제안한 스케쥴러를 사용한 결과 작업시간에 영향을 받지 않고 실행의 오류없이 모든 작업이 완료될 수 있었다.

Abstract

Recent advances in DNA sequencing technology have enabled Next Generation Sequencing (NGS) instruments to accelerate generating billions of DNA reads in a few days. However, the management of enormous NGS data and the concurrent analysis of these vast amount of data requires a great deal of computing power and memory as well as huge disk storage. Current popular job scheduling systems provide efficient ways for managing and scheduling vast amount of analysis based on the available computing resources but don't consider the maximum amount of Input/Output (I/O). Thus, when executing large number of genome analysis on a large scale cluster system, the maximum bandwidth of storage I/O is insufficient to utilize all computing resources so the analysis jobs are frequently suspended. Here we developed a disk I/O aware job submission scheduler to maximize disk I/O usage but not hampering previously running jobs due to the heavy disk I/O of a new job. And we constructed a cancer genome analysis pipeline by using our I/O aware scheduler and HPC resources in National Institute of Supercomputing and Networking (NISN) to overcome the obstacles of concurrent analysis for vast amount of NGS data. Based on our I/O aware job submission scheduler, we performed major genome analyses on over 50 case-control pairs of chromophobe renal cell carcinoma patients' whole genome samples sequenced by The

Cancer Genome Atlas (TCGA) and successfully completed all analysis jobs while maintaining no jobs to be suspended by I/O bottleneck.

서론

최근의 과학 기술 컴퓨팅은 데이터 중심의 컴퓨팅을 중심으로 급속도로 다각화되고 발전하고 있다. 기존에는 대부분의 과학 기술 컴퓨팅은 다수의 cpu를 활용하고 활용되는 cpu간의 고속 메세지 전송이 가능한 MPI 기반의 계산, 혹은 대용량 메모리를 활용하여 계산을 해야하는 분석 중심의 high performance computing이 주를 이루었다. 일 예로 유체역학 분야의 문제와 같이 수 천개에서 수 억개의 data point에 대해 인접 data point의 값을 계속적으로 참조 하면서 해당 data point의 값을 계산하는 미분방정식을 푸는 방식의 분석들이 이에 해당한다. 이러한 분석들의 대부분은 분석을 위한 모델의 정의, 그리고 해당 모델을 통해 문제를 해결하고자 하는 상황을 서술하는 파라메터들을 설정해 주는 방식을 통해 시뮬레이션을 수행하게 된다. 이러한 high performance computing 기반의 분석을 지원하기 위하여 최근까지 다양한 컴퓨팅 기술 및 작업 스케쥴링 기술들이 개발되어 high performance computing 기반의 분석 연구를 지원하고 있다. 특히, 현재 가장 많이 연구되었으며 많이 사용되고 있는 작업 스케쥴링 시스템들은 주어진 컴퓨팅 자원(CPU, 컴퓨팅 노드)들을 효율적으로 활용하여 작업들을 처리할 수 있는 다양한 방법들을 제공한다. 하지만 최근의 컴퓨팅의 추세는 단순히 많은 CPU와 메모리를 이용하는 분산 작업 뿐만 아니라 다양한 방법으로 수집, 생산되는 대용량의 데이터를 활용, 분석하는 작업들이 상당 부분을 차지하고 있다. 일 예로, 생물학 분야에서는 2000년대 후반에 개발된 병렬 시퀀싱 기술의 발달로 인해 하루에도 수 테라바이트의 데이터가 하나의 실험 장비에서 생산될 수 있으며 이렇게 생산된 데이터를 전부 활용하여야 연구를 수행하는 것이 가능하다.

하지만 현재 개발되어 사용되고 있는 작업 스케쥴링 시스템들은 대용량 데이터를 처리하는 작업 수행에 있어 효과적이지 못한 단점을 가지고 있다. 기존 작업 스케쥴링 시스템들은 대부분 작업을 스케쥴링하는 데에 있어서 해당 작업이 사용하는 CPU의 수와 예상 작업 시간에 따른 큐 정책을 기반으로 작업을 컴퓨팅 노드에 분배하고 실행하게 된다. 이런 스케쥴링 시스템을 통해 대용량 데이터 처리 작업을 다수 실행하게 되는 경우, 데이터 처리 작업이 필요로 하는 CPU의 숫자가 많지 않으면 대용량의 데이터를 읽거나 쓰는 작업이 상대적으로 많은 숫자가 실행되게 되며 결국 스토리지 I/O의 병목현상을 초래하게 된다. 스토리지 I/O의 병목현상이 일어나게 되면 대용량 데이터를 다루는 작업뿐만 아니라 전체 컴퓨팅 시스템에서 실행되고 있는 대다수의 작업들의 실행에 영향을 미치게 되어 전체적인 효율 감소에 이르게 된다. 이는 각 노드당 활용 가능 메모리의 총 용량 및 전체 시스템의 네트워크 대역폭의 경우도 비슷한 현상을 초래하게 된다. 따라서 대용량 데이터 처리에 기반한 분석 작업을 효율적으로 처리하기 위해서는 기존의 CPU 기반의 작업 스케쥴링이 아닌 스토리지의 I/O에 기반한 작업 스케쥴링 기법이 필요하다.

본 연구에서는 스토리지의 I/O의 사용량에 기반한 작업 스케쥴러를 제안함으로써 스토리지의 I/O를 가능한한 최대로 사용하면서 신규 작업의 I/O 사용량으로 인해 기존 실행 중인 작업들의 실행에 영향을 끼치지 않는 방법을 제시한다. 본 연구에서 제시하는 스케쥴러는 기존 CPU 기반의 스케쥴러 상위의 메타 스케쥴러의 성격을 갖고 있으며 현재 스토리지 시스템의 I/O 상태를 모니터링하면서 신규 작업이 제출되

는 경우 해당 작업의 최대 I/O 사용 예측량과 스토리지 시스템의 최대 I/O 처리량을 비교하여 해당 작업을 실행할지 혹은 waiting queue에 기다리게 할 지를 선택하는 방식으로 작동하게 된다. 제안한 스토리지 I/O 기반의 작업 스케쥴러의 효용성을 보이기 위해 본 연구에서는 대용량 데이터 처리 연구 분야 중 하나인 암 유전체 분석 연구에 적용하였다. 그 결과 약 400GB의 암 유전체 데이터를 분석하는 약 100개의 작업을 시스템의 성능 저하 없이 성공적으로 마칠 수 있었다.

다음 절에서는 제안하는 스토리지 I/O 기반의 작업 스케쥴러에 대한 설명과 실제 암 유전체 데이터 분석에 적용한 사례를 설명하고 이후 결론을 맺는다.

본론

서론에서 설명한 것과 같이 본 연구에서는 이미 실행되고 있는 작업들에 대한 영향을 최소화 하면서 스토리지 시스템의 I/O 대역폭을 최대한 활용할 수 있는 방법으로 작업을 분배하는 메타 스케쥴러를 개발하였다. 개발한 스토리지 I/O 기반 작업 스케쥴러는 TORQUE를 기반으로 개발되었으며, 가장 많이 사용되는 파일 시스템 중 하나인 Network File System(NFS)와 높은 I/O 성능을 낼 수 있는 병렬 분산 파일시스템인 Lustre File System(LFS)에서 동작 할 수 있도록 각각 스케쥴러를 구성하였다.

Network File System을 위한 Storage I/O aware job scheduler

본 연구에서 제안하고 개발한 스토리지 I/O기반의 작업 스케쥴러는 작업이 주로 이루어지는 컴퓨팅 시스템의 스토리지의 사용 현황을 모니터링한다. 이 모니터링은 특정 주기를 가지고 이루어지게 되며 각 주기마다 사용된 스토리지의 input, output의 총량을 기준으로 현재 스토리지 시스템의 사용량을 계산하게 된다. 이렇게 계산된 스토리지 시스템의 사용량이 해당 스토리지 시스템이 견딜 수 있는 최대 I/O 사용량보다 작고 실행될 작업의 기대 I/O 사용량을 현재 스토리지 시스템의 I/O 사용량에 더했을때, 최대 스토리지 I/O 사용량보다 작은 경우 해당 작업을 실행해도 스토리지 시스템에 무리가 가지 않는다고 판단하여 PBS TORQUE 스케쥴러에 해당 작업을 제출하는 방식으로 작업을 실행하게 된다. 만약 실행하고자하는 작업의 기대 I/O 사용량이 스토리지 시스템의 최대 I/O 사용량을 넘어가게 된다면 해당 작업을 제출하지 않고 다음 스토리지 시스템 사용량 모니터링 시간까지 기다린 후에 같은 작업을 되풀이한다.

위와 같은 방식을 채택하고 있기 때문에 제안하는 I/O 기반 스케쥴러를 사용하기 위해서는 시스템 관리자는 해당 스토리지 시스템의 최대 I/O 사용 허용량을 알고있어야한다. 대략적인 스토리지 시스템의 최대 I/O 사용 허용량은 시스템 판매처에서 제공하는 하드웨어 사양서를 참고하거나 iozone과 같은 시스템 벤치마킹 툴을 사용하여 알아낼 수 있다.

제안한 I/O 기반 스케쥴러는 대상 스토리지 시스템의 사용현황을 모니터링하는 부분이 필수적이다. 이를 위해서 network file system을 위한 스케쥴러에서는 network file system을 설치하면 함께 설치되어 작동되는 'nfsstat'을 활용한다. 'nfsstat'을 실행하였을 때 얻을 수 있는 정보를 파싱하여 현재까지 실행된 input request, output request를 저장한 후에 특정 시간 이후 다시 실행하여 그 차이를 계산한다. 시스템에서 설정한 I/O request의 크기, nfsstat을 실행한 시간 차이 등을 이용하면 시간당 스토리지 I/O 사용량을 추정하여 계산할 수 있다.

마지막으로 사용자의 작업이 사용하는 기대 I/O 사용량은 스토리지 시스템의 최대 I/O 허용량을 추정하는 방법과 마찬가지로 다양한 프로파일링 툴들을 이용하여 미리 추정치를 계산하여야 한다.

이 세가지 정보를 이용하여 다음과 같이 NFS에서 동작하는 스토리지 I/O 기반의 작업 스케쥴러를 구성할 수 있다. 다음 예는 암 전장 유전체 분석 툴 중의 하나인 TEA[4]를 실행하는 예제이다.

```
#!/bin/bash
SAMPLE_LIST=$1
filecontent=( `cat "$SAMPLE_LIST"`)


#TEA paths
TEA_BASE=/home/juneh/Tea
EXEC=$TEA_BASE/scripts/tea.pl
DIR=/cluster/data/scratch/TEA_run
```

```
TMPDIR=$DIR/tmp
SCRIPTDIR=$DIR/scripts
VA=$TEA_BASE/lib/assembly/hvir.collapsed.fa
RA=$TEA_BASE/lib/assembly/repeat.combined.div30.isize150.fa
BS=$TEA_BASE/lib/assembly/pylori.fa

#QSUB paths
QERROR=$DIR/qerror
QOUT=$DIR/qout

#IO query
INTERVAL=600
BSIZE=32
DATA=$(ssh biodata1 /usr/sbin/nfsstat --nfs --server | awk 'NR==5 {print}' |
     cut -d ' ' -f 1,3)
CREAD=$(echo $DATA | cut -d ' ' -f 1)
CWRITE=$(echo $DATA | cut -d ' ' -f 2)
IOTHREASHOLD=250

echo START$(date +%H:%M:%S)
for t in ${filecontent[@]}
do
        echo SAMPLE:$t
        DIRLIST=$(find $DIR -maxdepth 1 -type d -name "TCGA-*-$t-*")
        for d in $DIRLIST
        do
                while true
                do
                        sleep $INTERVAL
                        DATA=$(ssh biodata1 /usr/sbin/nfsstat --nfs --server
     | awk 'NR==5 {print}' | cut -d ' ' -f 1,3)
                        READ=$(echo $DATA | cut -d ' ' -f 1)
                        WRITE=$(echo $DATA | cut -d ' ' -f 2)
                        IOSTATUS=`echo "scale=3; $BSIZE * ( $READ - $CREAD +
     $WRITE - $CWRITE ) / 1024 / $INTERVAL" | bc`
                        CREAD=$READ
                        CWRITE=$WRITE
                        if [ $IOSTATUS -lt $IOTHREASHOLD ] then
                                break
                        fi
                done
                f=${d##*/}

                $EXEC tea -c hg19 -F -g $DIR/$f/bam/$f.bam -d $DIR -p 4 -P
     $TEA_BASE -r $RA -R ra -s $TMPDIR -f $f > $SCRIPTDIR/$f.ra.sh
                $EXEC tea -c hg19 -d $DIR -F -p 4 -P $TEA_BASE -r $VA -R va -
     s $TMPDIR -x -M 2 -o $f > $SCRIPTDIR/$f.va.sh
                FIRST=$(qsub -d $DIR -v f=$f -m e -M juneh@kisti.re.kr -l
     walltime=720:00:00,nodes=1:ppn=4:hugemem -N TEA${f:8:7}CBAM -e $QERROR
     -o $QOUT $DIR/cbam.sh)
```

```
                    qsub -W depend=afterok:$FIRST -d $DIR -m e -M
          juneh@kisti.re.kr -l walltime=720:00:00,nodes=1:ppn=4:hugemem -N TEA$
          {f:8:7}RA -e $QERROR -o $QOUT $DIR/scripts/$f.ra.sh
                    qsub -W depend=afterok:$FIRST -d $DIR -m e -M
          juneh@kisti.re.kr -l walltime=720:00:00,nodes=1:ppn=4:hugemem -N TEA$
          {f:8:7}VA -e $QERROR -o $QOUT $DIR/scripts/$f.va.sh
             done
    done
```

Lustre File System을 위한 Storage I/O aware job scheduler

앞에서 설명한 방법과 같이 NFS에서는 단순히 해당 스토리지 시스템의 I/O 허용량을 넘지 않는 선에서 작업을 실행하는 방법으로 스케쥴링을 하는 방식으로 스토리지 시스템을 효율적으로 사용하고 다른 기실행 작업의 속도 저하를 막을 수 있다. 하지만 스토리지 시스템의 고성능을 위해 널리 사용되고 있는 분산 병렬 파일 시스템과 같은 경우는 NFS의 경우와 같이 단순한 방법으로 작업을 스케쥴링하게 되는 경우 기존 스케쥴링 기법과 마찬가지로 스토리지 시스템의 병목현상을 피하지 못하는 경우가 발생할 수 있다. 이는 분산 병렬 파일 시스템의 특정 object storage에 분석 대상 파일들이 집중되어 저장되어 있는 경우 전체 분산 병렬 파일 시스템의 대역폭은 상대적으로 많이 여유가 있지만 파일들이 집중되어있는 object storage의 I/O 대역폭이 가득차게 되는 경우가 생길 수 있기 때문이다. 이러한 이유때문에 분산 병렬 파일 시스템에서 스토리지의 I/O를 기반으로 작업을 스케쥴링하기 위해서는 분석 대상인 파일들이 저장되어있는 object storage의 정보를 활용해야 효율적인 스케쥴링을 수행할 수 있다. 이와 같이 데이터의 분산 병렬 파일 시스템 상의 위치와 더불어 NFS의 경우와 같이 전체 분산 병렬 파일 시스템의 대역폭 및 현재 사용량, 그리고 분산 병렬 파일 시스템을 이루는 단위 스토리지의 대역폭 및 현재 사용량을 모두 종합적으로 이용하게 된다.

본 연구에서는 널리 사용되고 있는 분산 병렬 파일 시스템 중 하나인 Lustre file system(LFS)를 대상으로 스토리지 I/O 기반 작업 스케쥴러를 개발하였다. 위에서 설명한대로, LFS에서는 단위 스토리지를 Object Storage Server(OSS)와 OSS를 이루는 Object Storage Target(OST)로 나타낸다. 즉, 스토리지 I/O 기반 스케쥴러는 분석을 해야하는 파일들이 어떤 OSS와 OST에 속하는지 확인하고 파일이 속해있는 OSS와 OST의 대역폭 및 현재 사용량, 그리고 전에 LFS의 대역폭 및 현재 사용량을 전체적으로 파악하여 분석의 기대 I/O 사용량이 각각 OSS, OST, LFS의 대역폭을 초과하지 않는 선에서 작업을 실행하게 된다.

우선 스케쥴러는 분석에 사용할 데이터의 리스트를 입력으로 받는다. 이 데이터의 리스트를 lfs find 명령어와 lfs getstripe 명령어를 이용하여 각 데이터들의 LFS상에서의 위치 즉, 어떠한 OSS와 OST에 데이터가 위치하는지를 확인한다. LFS에서는 하나의 파일이 여러개의 OSS와 OST에 분산되어 저장하게 할 수도 있으며 하나의 OST에 저장하게 설정할 수 있다. 이는 각 분석의 특성에 따라 최적의 방법이 존재할 수 있는데 본 연구에서 타겟으로 하는 분석 방식, 즉, 각 데이터의 파일 사이즈가 크고 파일의 갯수가 많은 경우, 그리고 모든 데이터들을 동시에 분석해야하는 경우는 굳이 데이터를 분산하여 저장할 필요성이 없으므로 각 데이터는 striping을 하지 않고 저장하는 방식을 채택하였다. 따라서 각 데이터의 위치를 알아낼 수 있다. 이 정보를 기반으로 분석을 수행해야하는 파일들을 OST, OSS 별로 그룹을 짓는다. 이후 cerebro 모니터링 툴을 이용하여 각 OST, OSS의 I/O 사용량을 NFS의 경우와 마찬가지로 모니터링한다. cerobro로 부터 얻어진 각 OST, OSS의 I/O 사용량과 각 데이터 파일의 위치를 기반으로 각 OST 별로 NFS에서 수행한 것과 같이 OST, OSS의 I/O 사용량과 분석의 기대 I/O 사용량을 비교하면서 작업을 수행한다. 이때, 최대한 분석을 LFS에 분산시키기 위하여 각 OST를 round robin 방식으로 돌아가며 작

업 제출을 수행하게 된다. 또한 전체 LFS의 대역폭을 고려하여 해당 OST의 대역폭에 여유가 있더라도 전체 LFS의 대역폭이 가득차게 되면 작업 제출을 멈추고 일정시간 대기하게 된다. 이러한 방식으로 분산 병렬 파일 시스템에서도 대용량 데이터를 대량으로 분석하는 경우에도 시스템의 과부하를 막고 작업을 안전하게 완료할 수 있다.

아래는 전장유전체 데이터의 리스트를 받아 이를 위에서 설명한 방법 처럼 LFS의 사용량을 모니터링 하면서 작업을 스케쥴링하는 스케쥴링 스크립트이다.

```
#!/bin/bash
EMAIL="juneh@kisti.re.kr"
DIR=$( pwd )
QERROR=$DIR/qerror
QOUT=$DIR/qout

while getopts "l:v:f:" options; do
case $options in
  l)
        INPUTFILELIST=$OPTARG ;;
  v)
 VARNAME=$OPTARG ;;
  f)
 SCRIPT=$OPTARG ;;
esac
done

if [[ -z $VARNAME ]]; then
 VARNAME=BAM
fi


if [ $OPTIND == 1 ] || [ -z $SCRIPT ] ; then
        SELF=`basename $0`
        cat <<EOF
Usage:  $SELF [-l input file list][-v variableName] -f <script to submit>
                -l <input file list> : input file list
                -v <variable name> : variable name for script
EOF
        exit 1;
fi
shift $((OPTIND-1))


echo START$(date +%H:%M:%S)


#Gets obdid for each bam file
```

```
BAMSINOBD=.Bams_in_obd
if [[ -z $INPUTFILELIST ]]; then
        echo yes
        BAMS=$(lfs find . -type l -name '*.bam')
else
        echo no
        BAMS=$( cat $INPUTFILELIST )
fi
rm -f $BAMSINOBD*

for BAM in $BAMS; do
        TTT=($(lfs getstripe $(readlink $BAM) | sed -e 's/^[[:space:]]*//' |
     cut -d " " -f 1))
        echo $BAM >> $BAMSINOBD$(echo ${TTT[7]}| sed -e 's/^[ \t]*//' -e 's/[
     \t]*$//')
done
echo Number of BAMS in each OST
wc -l $BAMSINOBD*

INTERVAL=1
NETIOTHRESHOLD=126000 #MB/s 1/4 of Infiniband switch bandwidth
OSTIOTHRESHOLD=370 #MB/s 1/2 of Infiniband bandwidth * 1/3 OSTs
NUMOSTS=9 #9
CUROST=8
allDone () { FSIZE=$(stat -c %s $BAMSINOBD*); RETURN=0; for i in $FSIZE; do
     RETURN=$(echo "$RETURN + $i" | bc); done; echo $RETURN; }
nextOST () { while true; do if [ $(echo "$CUROST + 1" | bc) -eq $NUMOSTS ];
     then CUROST=0; else CUROST=$(echo "$CUROST + 1" | bc); fi; if [ $(stat -
     c %s $BAMSINOBD$CUROST) -gt 0 ] || [ $( allDone ) -eq 0 ]; then break;
     fi; done; }
curostfilesize () { FSIZE=$(stat -c %s $BAMSINOBD$CUROST); echo $FSIZE; }

IFS=";
"
CREAD=($(cerebro-stat -m lmt_ost | cut -d ';' -f 10,25,40))
CWRITE=($(cerebro-stat -m lmt_ost | cut -d ';' -f 11,26,41))

while true
do
        #Gather current status of each osts
        sleep $INTERVAL
        READ=($(cerebro-stat -m lmt_ost | cut -d ';' -f 10,25,40))
        WRITE=($(cerebro-stat -m lmt_ost | cut -d ';' -f 11,26,41))
        echo -n $(date +%H:%M:%S)
        READP=0
        for ((i=0; i<${#CREAD[@]}; i++))
        do
                READP=$(($READP + $(echo "${READ[i]} - ${CREAD[i]}" | bc)))
        done
        WRITEP=0
        for ((i=0; i<${#CWRITE[@]}; i++))
```

```
        do
                WRITEP=$(($WRITEP + $(echo "${WRITE[i]} - ${CWRITE[i]}" |
    bc)))
        done
        NETIO=$(echo "scale=3; ($READP + $WRITEP) / 1024 / 1024 / $INTERVAL"
    | bc)
echo -n " Total:$NETIO MB/s OST-$CUROST:$(bc <<< "scale=3; (($
    {READ[$CUROST]} + ${WRITE[$CUROST]} - ${CREAD[$CUROST]} - $
    {CWRITE[$CUROST]}) / 1024 / 1024 / $INTERVAL)")MB/s"

#IF NETIO is smaller than the threshold
        if [ $(bc <<< "$NETIO < $NETIOTHRESHOLD") -eq 1 ]; then
                #Check wheather CUROST is not busy and CUROST file is not
    empty
                CUROSTFILESIZE=$( curostfilesize )
                ISOSTFREE=$(bc <<< "scale=3; ((${READ[$CUROST]} + $
    {WRITE[$CUROST]} - ${CREAD[$CUROST]} - ${CWRITE[$CUROST]}) / 1024 / 1024
    / $INTERVAL) < $OSTIOTHRESHOLD")
                if [ $ISOSTFREE -eq 1 ] && [ $CUROSTFILESIZE -ne 0 ]; then
                        #Check whether resource is free
                        FREEHMEM=$(pbsnodes :hugemem | grep "state =" | grep
    free | wc -l)
                        FREEBLAST=$(pbsnodes :blast | grep "state =" | grep
    free | wc -l)
                        if [ $FREEHMEM -gt 0 ] ; then

    RLIST="walltime=720:00:00,nodes=1:ppn=4:hugemem"
                                echo ' hugemem is free'
                        elif [ $FREEBLAST -gt 0 ] ; then

    RLIST="walltime=720:00:00,nodes=1:ppn=4:blast"
                                echo ' blast is free'
                        else
                                echo ' IO is ok but not engough computing
    resources. Wait for another $INTERVAL sec'
                                continue
                        fi
                        #submit the job and remove bam file from CUROST file
                        BAM=$( head $BAMSINOBD$CUROST -n 1 )

                        echo "qsub -v $VARNAME=$BAM -d $DIR -m e -M $EMAIL -l
    $RLIST -e $QERROR -o $QOUT $SCRIPT"

                        echo OBD$CUROST:$BAM >> submitted.txt
                        sed -i 1d $BAMSINOBD$CUROST
                        nextOST
                else #when CUROST is busy or CUROST file is empty
                        echo " OST-$CUROST IO is busy or no bam in the OST-
    $CUROST"
                        nextOST
                fi
```

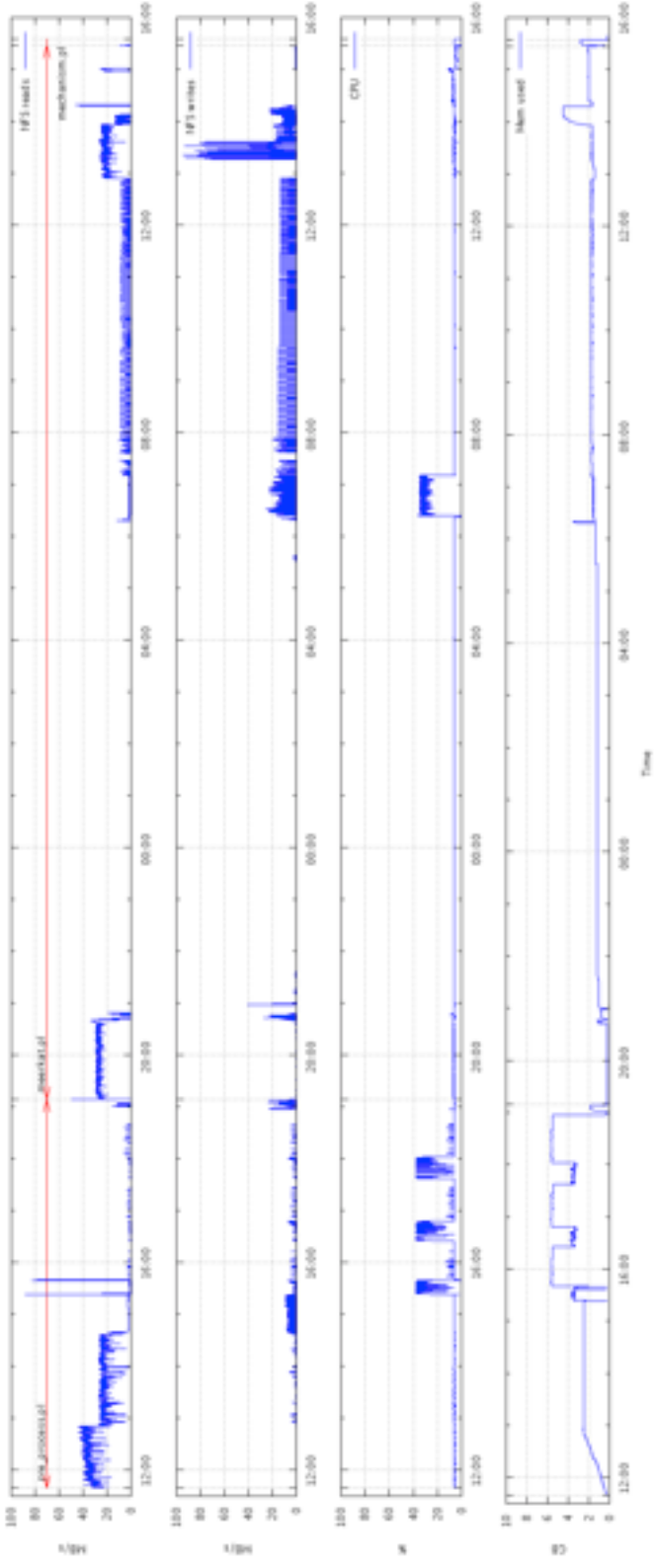
```
            else
                    echo " Infiniband is busy"
            fi
            CREAD=("${READ[@]}")
            CWRITE=("${WRITE[@]}")
     ISITOVER=$( allDone )
     if [ $ISITOVER -eq 0 ]; then
      break
     fi
    done
    rm -f $BAMSINOBD*
    echo Submission completed!
```

암 전장 유전체 분석 연구 활용 사례

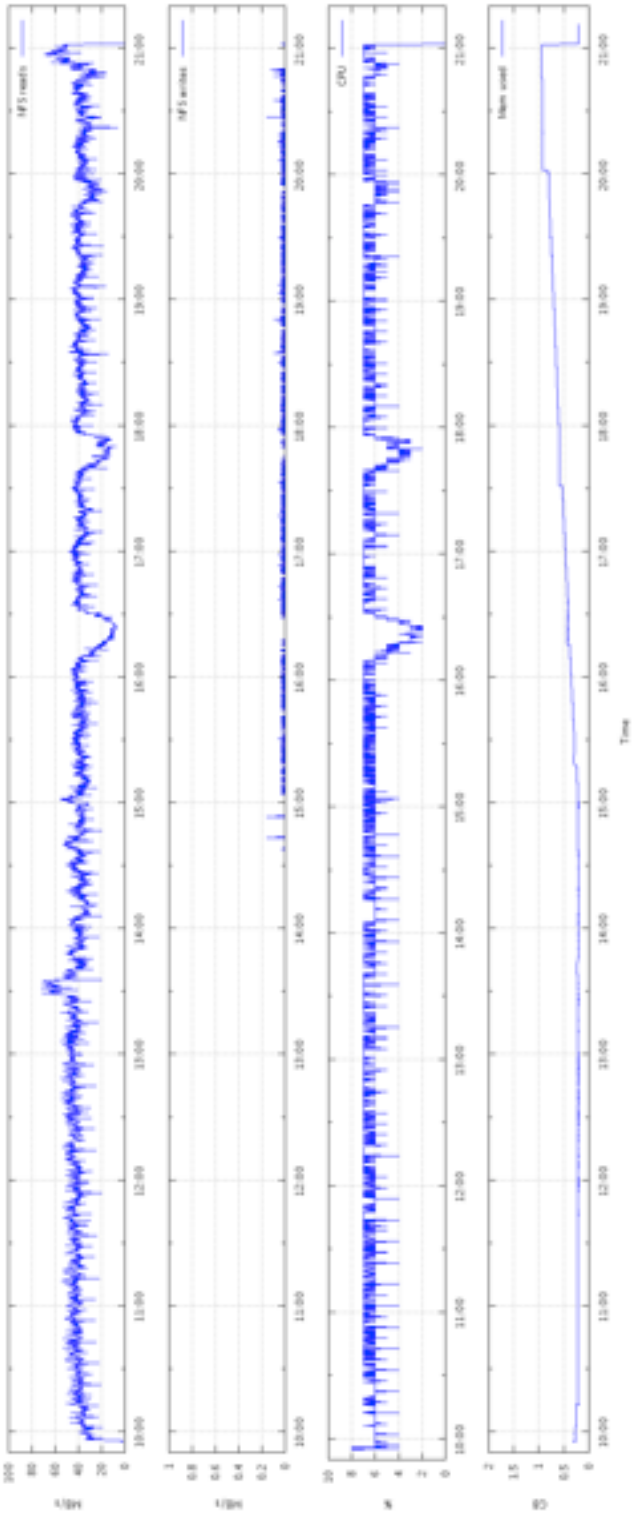
앞에서 설명한 스토리지 I/O 기반 작업 스케쥴러를 활용하기 위해 우리는 해당 스케쥴러를 국가슈퍼컴퓨팅연구소에서 운영 중인 HPC 자원에 설치하고 암 전장 유전체 분석 파이프라인에 적용하였다. 암 전장 유전체 분석의 특징을 설명하기 위해 진행한 분석 중 하나인 신장암 유전체 구조 변이 연구의 경우 연구의 특성상 한 명의 환자로부터 정상 세포 그리고 암 세포로부터 얻어진 데이터를 각각 보유하여야 연구를 수행할 수 있다. 따라서 총 50명 X 2(정상 세포, 암 세포)인 100개의 데이터를 분석하여 연구를 진행하여야 한다. 각 데이터 당 평균 300GBytes의 크기를 가지고 있으며 100 샘플에 대한 데이터의 총 크기는 약 32.7TByte로 확인 되었다. 전장 유전체 분석을 수행하기 위해서는 300GB의 데이터를 분석하는 작업을 100개 실행해야한다. 하지만 각 작업은 주로 300GB에 달하는 데이터를 고속으로 읽어들이는 작업을 기반으로 하고 있기 때문에 100개의 작업이 동시에 각기 다른 300GB의 데이터를 읽기 시작하면 스토리지의 대역폭을 넘게 되고 시스템의 전체적인 성능저하로 이어질 가능성이 있다.

앞 절에서 설명한대로 각 전장 유전체 분석 작업들의 기대 스토리지 I/O 사용량을 추정하기 위해 최신 전장 유전체 구조 분석 툴의 I/O 사용량을 프로파일링하였다. 그 결과는 다음그림과 같다.

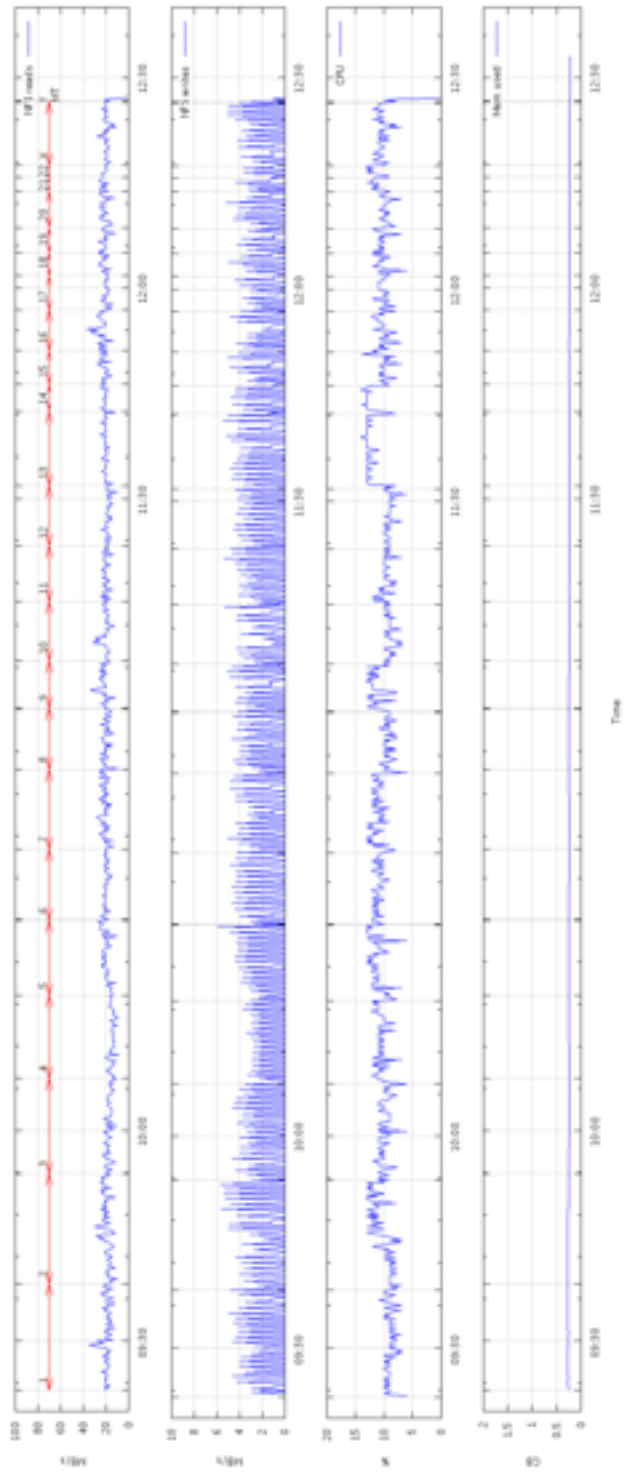
MEERKAT[1]

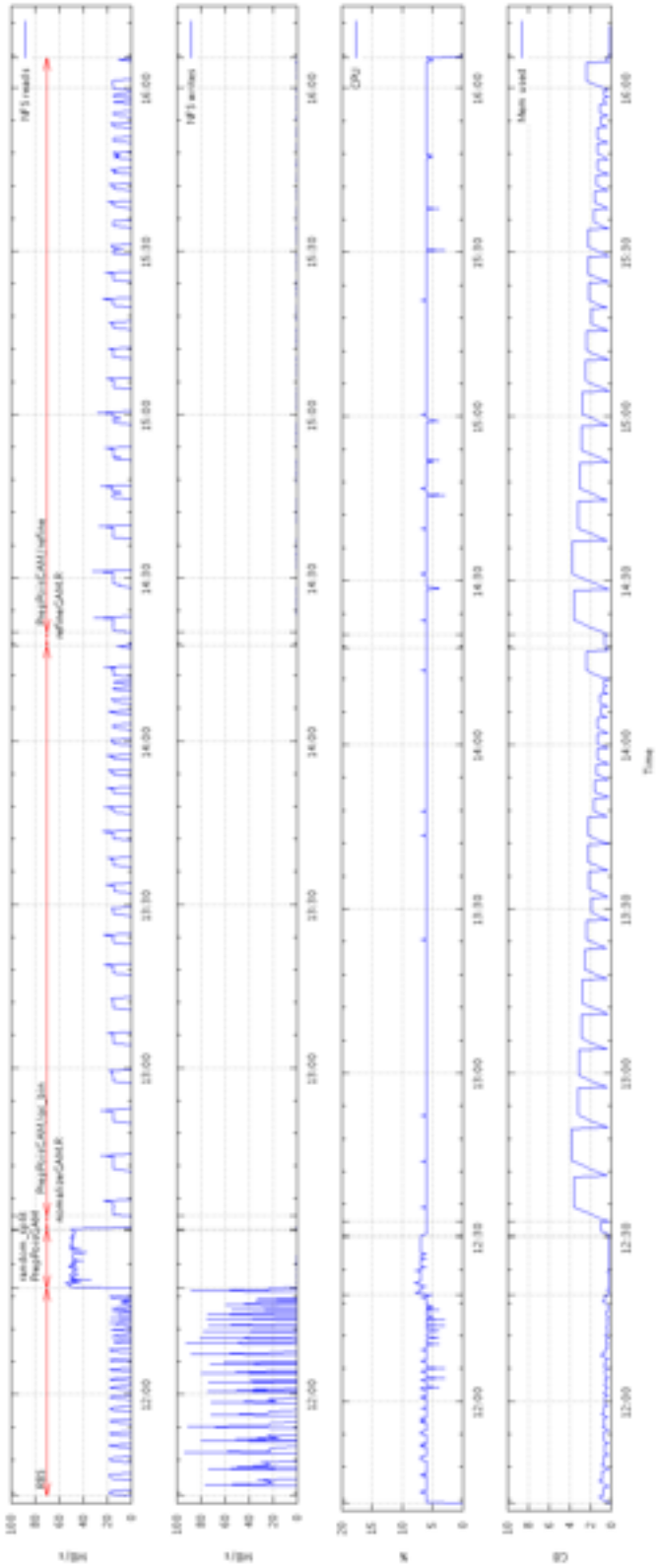


BREAKDANCER[2]



BIC-SEQ[3]





위 결과들은 NFS상에서 측정된 결과이다. 위 결과에서 볼 있듯이 대부분의 툴들은 약 50MB/s의 속도로 파일을 읽어들이는 작업을 초반에 주로 하게되며 간혹 상당히 높은 속도로 쓰는 작업을 수행하는 경우도 발생하게 된다. 측정된 기대 I/O 사용량에 완충 작용을 하기 위해 약 10MB/s의 여유를 두어 각 툴의 기대 I/O 사용량을 정하였다. 이렇게 정해진 기대 I/O 사용량을 이용하여 앞절에서 설명한 스토리지 I/O 기반 작업 스케쥴러를 실행한 결과 100개의 데이터를 실행하는 파이프라인이 서로 작업들의 충돌없이 완료되는 것을 확인할 수 있었다. NFS 상에서 최대 13개까지의 작업이 동시에 실행되었다. 즉, 최대 실행가능한 전장 유전체 분석 작업이 13개인데 반해 기존 CPU 기반의 스케쥴러를 사용하게 되면 각 작업이 활용하는 CPU의 갯수는 1~4개에 불과하기 때문에 거의 모든 100개의 작업이 동시에 실행되게 되고 스토리지의 병목현상이 발생하게 되는 것을 제안한 I/O 기반의 스케쥴러를 사용하게 됨으로써 방지할 수 있었다. 해당 연구의 결과는 Cancer Cell에 발표되었다. [5]

한계점

현재 제안된 스케쥴링 기법은 단순히 분석 작업의 기대 I/O 사용량을 기반으로 스케쥴링을 하고 있다. 이러한 방법으로도 기존의 방법보다는 안정적인 작업 실행이 가능하였다. 하지만 그 이유는 실행한 분석 작업의 I/O 사용 프로파일이 특정 패턴을 가지고 있었기 때문이다. 대부분의 전장 유전체 분석 파이프라인의 I/O 사용 프로파일을 다시 확인해보면 분석 시작 시 빠른 속도로 데이터를 읽어들이는 작업을 수행하여 I/O를 많이 사용하지만 이후에는 I/O의 사용보다는 메모리와 CPU의 사용이 주가 된다. 즉, 초반의 I/O 집중 사용 기간과 I/O 사용이 상대적으로 덜한 부분으로 나뉠 수 있다는 것이 주로 사용한 분석 파이프라인의 특징이다. 하지만 I/O 사용 프로파일의 변화가 큰 경우 예를 들어 초반에 많은 I/O를 사용하다가 중간에는 I/O 사용량이 적었다가 다시 I/O 사용량이 많아지는 것과 같은 프로파일을 갖는 경우 두번째의 I/O 사용 구간의 실행 시간을 예측하기가 어려우므로 현재 제안된 스케쥴링 기법이 제대로 동작하지 못할 수 있다. 이러한 경우 단순히 I/O 사용량의 기대치를 가지고 스케쥴링하는 것이 아니라 해당 작업의 전체 실행 시간 및 전체 실행시간 동안의 I/O 사용량의 변화까지 모두 확인하고 스케쥴링이 되어야한다. 하지만 이러한 경우 스케쥴링의 복잡성이 높아질 수 있기 때문에 현 선행연구 격의 프로토타입에서는 관련 기능은 구현되지 않았다.

결론

생명공학분야의 DNA 시퀀싱 기술의 눈부신 발전을 필두로 다양한 분야의 신기술 실험장비에서부터 생산되는 대용량의 데이터는 현대 컴퓨팅 기술의 패러다임에 적지않은 변화를 가져오고 있다. 끝없이 생산되고 있는 방대한 양의 데이터를 얼마나 효율적으로 관리하고 분석할 것인지가 중요한 이슈로 자리잡았다. 하지만 지금까지 사용되오던 작업 관리 시스템들은 CPU를 최대한 효율적으로 활용하는데 초점을 두고 있었으며 따라서 방대한 양의 데이터를 효율적으로 분석하는 데에는 상대적으로 비효율적인 경우가 생기게 된다. 이러한 경우는 본론에서 논한 대로 상대적으로 컴퓨팅을 위한 CPU는 여유있는 반면 데이터 분석을 위한 스토리지 시스템의 대역폭이 모자라게 되는 경우 이러한 현상들이 발생되게 된다. 이 것은 지금까지 대부분의 컴퓨팅 센터들이 추구해오던 분석 방법들의 특징이 CPU 위주이기 때문에 CPU 위주로 시스템이 갖춰져 있기 때문이라고 볼 수 있다. 즉 현재의 시스템을 유지하면서 대용량 데이터 분석을 처리하기 위해서는 데이터 분석 작업을 기존의 스케쥴링 기법이 아닌 데이터 분석의 관점에서의 작업 스케쥴링이 필요한 것이다. 따라서 본 연구에서는 대용량 암 전장 유전체 데이터의 안정적이고 효율적인 분석을 위해 스토리지 I/O에 기반한 스케쥴링 기법에 대하여 논하였다. 제안된 스토리지 I/O 기반의 스케쥴링 기법을 적용한 결과 대용량 전장 유전체 분석을 안정적으로 완료할 수 있었다. 향후 분석이 필요한 데이터의 양은 계속적으로 늘어날 것으로 예상되며 제안한 방법과 같은 혹은 더 진보한 스케쥴링 기법이 필수적일 것으로 생각된다.

참고문헌

[1] L. Yang, L. J. Luquette, N. Gehlenborg, R. Xi, P. S. Haseley, C.-H. Hsieh, C. Zhang, X. Ren, A. Protopopov, L. Chin, R. Kucherlapati, C. Lee, and P. J. Park, "Diverse mechanisms of somatic structural variations in human cancer genomes." Cell, vol. 153, no. 4, pp. 919–929, May 2013.

[2] K. Chen, J. W. Wallis, M. D. McLellan, D. E. Larson, J. M. Kalicki, C. S. Pohl, S. D. McGrath, M. C. Wendl, Q. Zhang, D. P. Locke, X. Shi, R. S. Fulton, T. J. Ley, R. K. Wilson, L. Ding, and E. R. Mardis, "BreakDancer: an algorithm for high-resolution mapping of genomic structural variation," Nature Methods, vol. 6, no. 9, pp. 677–681, Sep. 2009.

[3] R. Xi, A. G. Hadjipanayis, L. J. Luquette, T.-M. Kim, E. Lee, J. Zhang, M. D. Johnson, D. M. Muzny, D. A. Wheeler, R. A. Gibbs, R. Kucherlapati, and P. J. Park, "Copy number variation detection in whole-genome sequencing data using the Bayesian information criterion." Proceedings of the National Academy of Sciences of the United States of America, vol. 108, no. 46, pp. E1128–36, Nov. 2011.

[4] E. Lee, R. Iskow, L. Yang, O. Gokcumen, P. Haseley, L. J. Luquette, J. G. Lohr, C. C. Harris, L. Ding, R. K. Wilson, D. A. Wheeler, R. A. Gibbs, R. Kucherlapati, C. Lee, P. V. Kharchenko, P. J. Park, and The Cancer Genome Atlas Re- search Network, "Landscape of Somatic Retrotransposition in Human Cancers," Science (New York, N.Y.), vol. 337, no. 6097, pp. 967–971, Aug. 2012.

[5] C.F.Davis,C.J.Ricketts,M.Wang,L.Yang,A.D.Cherniack, H. Shen, C. Buhay, H. Kang, S. C. Kim, C. C. Fahey, K. E. Hacker, G. Bhanot, D. A. Gordenin, A. Chu, P. H. Gunaratne, M. Biehl, S. Seth, B. A. Kaipparettu, C. A. Bristow, L. A. Donehower, E. M. Wallen, A. B. Smith, S. K. Tickoo, P. Tamboli, V. Reuter, L. S. Schmidt, J. J. Hsieh, T. K. Choueiri, A. A. Hakimi, The Cancer Genome Atlas Research Network, L. Chin, M. Meyerson, R. Kucherlapati, W.-Y. Park, A. G. Robertson, P. W. Laird, E. P. Henske, D. J. Kwiatkowski, P. J. Park, M. Morgan, B. Shuch, D. Muzny, D. A. Wheeler, W. M. Linehan, R. A. Gibbs, W. K. Rathmell, C. J. Creighton, and Cancer Genome Atlas Research Network, "The somatic genomic landscape of chromophobe renal cell carcinoma." Cancer cell, vol. 26, no. 3, pp. 319–330, Sep. 2014.