BIOINFORMATICS

Vol. 00 no. 00 2014
Pages 1–7

Cloudbreak: Accurate and Scalable Genomic Structural Variation Detection in the Cloud with MapReduce

Christopher W. Whelan[1], Jeffrey Tyner[2], Alberto L'Abbate[3], Clelia Tiziana Storlazzi,[3], Lucia Carbone[4], and Kemal Sönmez[1,5,∗∗]

[1]Institute on Development and Disability and Center for Spoken Language Understanding, Oregon Health & Science University (OHSU), Portland, OR, USA
[2] Program in Molecular and Cellular Biosciences, OHSU, Portland, OR, USA
[3] Department of Biology, University of Bari "Aldo Moro", Via G. Amendola 165/A, 70126, Bari, Italy
[4] Behavioral Neuroscience Department, OHSU, Portland, OR, USA
[5] Department of Medical Informatics & Clinical Epidemiology, OHSU, Portland, OR, USA

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Motivation: The detection of genomic structural variations (SV) remains a difficult challenge in analyzing sequencing data, and the growing size and number of sequenced genomes have rendered SV detection a bona fide big data problem. MapReduce is a proven, scalable solution for distributed computing on huge data sets.

Results: We describe an algorithmic framework for SV detection algorithms in MapReduce based on computing local genomic features, and use it to develop a deletion and insertion detection algorithm, Cloudbreak. On simulated and real data sets, Cloudbreak achieves accuracy improvements over popular SV detection algorithms, and genotypes variants from diploid samples. It provides dramatically shorter runtimes and the ability to scale to big data volumes on large compute clusters. Cloudbreak includes tools to set up and configure MapReduce (Hadoop) clusters on cloud services, enabling on-demand cluster computing.

Availability: Source code, binary releases, and user manual are available at `http://github.com/cwhelan/cloudbreak`

Supplementary information: Supplementary material is available at *Bioinformatics* online.

Contact: sonmezk@ohsu.edu

1 INTRODUCTION

Genomic structural variations (SVs) such as deletions, insertions, and inversions of DNA are widely prevalent in human populations and account for the majority of the bases that differ among normal human genomes (Mills *et al.*, 2011a; Conrad *et al.*, 2010). However, detection of SVs with current high-throughput sequencing technology remains a difficult problem, with limited concordance between available algorithms and high false discovery rates (Mills *et al.*, 2011a). Part of the problem stems from the fact that the signals indicating the presence of SVs are spread throughout large data sets, making the problem computationally intensive. As the volume of massively parallel sequencing data approaches "big data" scales, SV detection is becoming a time consuming component of genomics pipelines, and presents a significant challenge for research groups and clinical operations that may not be able to scale their computational infrastructure. Here we present a distributed software solution that is scalable and readily available on the cloud.

In other fields that have taken on the challenge of handling very large data sets, such as internet search, scalability has been addressed by frameworks that distribute processing to many compute nodes, each working on local copies of portions of the data. In particular, Google's MapReduce (Dean and Ghemawat, 2008) was designed to manage the storage and efficient processing of big data across clusters of commodity servers. Hadoop is an open source project of the Apache Foundation that provides an implementation of MapReduce. MapReduce and Hadoop allow efficient processing of large data sets by executing tasks on nodes that are as close as possible the data they require, minimizing network traffic and I/O contention. The Hadoop framework has been successfully applied to sequencing-related tasks including short read mapping (Schatz, 2009) and SNP calling (Langmead *et al.*, 2009).

Hadoop/MapReduce requires a specific programming model, however, which can make it difficult to design general-purpose algorithms for arbitrary sequencing problems like SV detection. MapReduce divides computation across a cluster into three phases. In the first phase, *mappers* developed by the application programmer examine small blocks of data and emit a set of $\langle key, value\rangle$ pairs for each block examined. The framework then sorts the output of the mappers by key, and aggregates all values that are associated with each key. Finally, the framework executes *reducers*, also created by the application developer, which process all of the values for a particular key and produce one or more outputs that summarize or aggregate those values.

SV detection algorithms use three main signals present in high-throughput sequencing data sets (Alkan *et al.*, 2011). Read-pair (RP) based methods use the distance between and orientation of the mappings of paired reads to identify the signatures of SVs (Campbell *et al.*, 2008; Chen *et al.*, 2009; Sindi *et al.*, 2009; Korbel *et al.*, 2009). Traditionally, this involves separating mappings into those that are *concordant* or *discordant*, where

∗to whom correspondence should be addressed discordant mappings deviate from the expected insert size or orientation, and then clustering the discordant mappings to find SVs supported by multiple discordantly mapped read pairs. Read-depth (RD) approaches consider the changing depth of coverage of concordantly mapped reads along the genome (Alkan *et al.*, 2009; Yoon *et al.*, 2009; Chiang *et al.*, 2009). Finally, split-read (SR) methods look for breakpoints by mapping portions of individual reads to different genomic locations (Wang *et al.*, 2011; Ye *et al.*, 2009).

Many RP methods consider only unambiguously discordantly mapped read pairs. Some approaches also include ambiguous mappings of discordant read pairs to improve sensitivity in repetitive regions of the genome (Hormozdiari *et al.*, 2009; Quinlan *et al.*, 2010). Several recent RP approaches have also considered concordant read pairs, either to integrate RD signals for improved accuracy (Sindi *et al.*, 2012; Michaelson and Sebat, 2012; Chiara *et al.*, 2012), or to eliminate the thresholds that separate concordant from discordant mappings and thus detect smaller events (Marschall *et al.*, 2012). Increasing the number of mappings considered, however, increases the computational burden of SV detection.

Our goal is to leverage the strengths of MapReduce to provide fast, accurate and readily scalable SV-detection pipelines. The main challenge in this endeavor is the need to separate logic into mappers and reducers, which makes it difficult to implement traditional RP-based SV detection approaches in MapReduce, particularly given the global clustering of paired end mappings at the heart of many RP approaches. MapReduce algorithms, by contrast, excel at conducting many independent calculations in parallel. In sequencing, for example, MapReduce based SNV-callers Crossbow (Langmead *et al.*, 2009) and GATK (McKenna *et al.*, 2010) perform independent calculations on partitions of the genome. SV approaches that are similarly based on local computations have been described: the RP-based SV callers MoDIL (Lee *et al.*, 2009) and forestSV (Michaelson and Sebat, 2012) compute features along the genome and then produce SV predictions from those features in a post-processing step. We will show that this strategy can be translated into the MapReduce architecture.

In this paper, we describe a framework for solving SV detection problems in Hadoop based on the computation of local genomic features from paired end mappings. We have used this framework to develop a software package, Cloudbreak, that discovers genomic deletions up to 25,000bp long, and short insertions. Cloudbreak computes local features based on modeling the distribution of insert sizes at each genomic location as a Gaussian Mixture Model (GMM), an idea first implemented in MoDIL (Lee *et al.*, 2009). The use of Hadoop enables scalable implementations of this class of algorithm. We characterize our algorithm's performance on a simulated data set, as well as a high coverage real data set from a normal individual and a low coverage cancer data set, and compare its performance to those of several popular methods.

Finally, and quite importantly from a practical point of view, our implementation of Cloudbreak provides the functionality to easily set up and configure Hadoop clusters on cloud service providers, making dynamically scalable distributed SV detection accessible to all whose computational needs demand it.

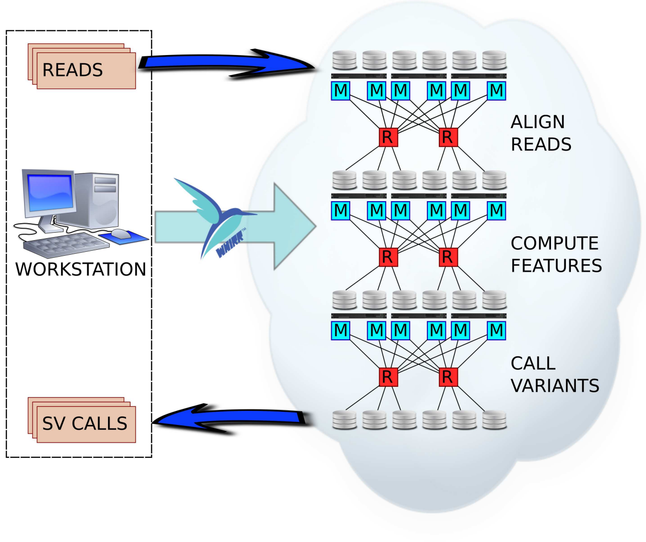

Fig. 1. An overview of the Cloudbreak workflow. Reads are first uploaded to a Hadoop cluster from local storage. Cloudbreak then executes three MapReduce jobs to process the data: 1) Mapping with sensitive settings. 2) Computation of features across the genome. 3) Calling structural variations based on the features computed in the previous step. Finally, SV predictions can be downloaded from the Hadoop cluster and examined. Cloudbreak can also use the Apache Whirr library to automatically provision Hadoop clusters on and deploy data to cloud providers such as Amazon Elastic Compute Cloud.

2 METHODS

2.1 A Framework for SV Detection in MapReduce

Our framework for SV detection in MapReduce divides processing into three distinct MapReduce jobs (Figure 1): a job that can align reads to the reference using a variety of mapping algorithms; a job that computes a set of features along the genome; and a job which calls structural variations based on those features.

The ALIGN READS job uses existing alignment tools to discover mapping locations for each read pair. Aligners can be executed to report multiple possible mappings for each read, or only the best possible mapping. Given a set of read pairs, each of which consists of a read pair identifier $rpid$ and two sets of sequence and quality scores $< s, q >$, each mapper aligns each pair end set $< s, q >$ in either single- or paired end mode and emits possible mapping locations under the $rpid$ key. Reducers then collect the alignments for each paired end, making them available under one key for the next job.

In the COMPUTE FEATURES job, we compute a set of features for each location in the genome. To begin, we tile the genome with small fixed-width, non-overlapping intervals. For the experiments reported below we use an interval size of 25bp (see Supplementary Materials for an exploration of the effects of different window sizes on accuracy and runtime). Let $L = \{l_1, l_2, \ldots, l_N\}$ be the set of intervals covering the entire genome. Let $R^1 = \{r^1_1, r^1_2, \ldots, r^1_M\}$ and $R^2 = \{r^2_1, r^2_2, \ldots, r^2_M\}$ be the input set of paired reads. Let $A^1 = \left\{a^1_{m,1}, a^1_{m,2}, \ldots, a^1_{m,K}\right\}$ and $A^2 = \left\{a^2_{m,1}, a^2_{m,2}, \ldots, a^2_{m,L}\right\}$ be the set of alignments for the left and right reads from read pair m. For any given pair of alignments of the two reads in a read pair, $a^1_{m,i}$ and $a^2_{m,j}$, let the ReadPairInfo $rpi_{m,i,j}$ be information about the pair relevant to detecting SVs, e.g. the fragment size implied by

Algorithm 1 A framework for SV calling in MapReduce.

```
 1: job ALIGNMENT
 2:     function MAP(ReadPairId $rpid$, ReadId $r$, ReadSequence $s$,
                 ReadQuality $q$)
 3:         for all Alignments $a \in$ ALIGN$(s, q)$ do
 4:             EMIT(ReadPairId $rpid$, Alignment $a$)
 5:     function REDUCE(ReadPairId $rpid$, Alignments $a_{1,2,\ldots}$)
 6:         AlignmentPairList $ap \leftarrow$ VALIDALIGNMENTPAIRS$(a_{1,2,\ldots})$
 7:         EMIT(ReadPairId $rp$, AlignmentPairList $ap$)
 8: job COMPUTE SV FEATURES
 9:     function MAP(ReadPairId $rp$, AlignmentPairList $ap$)
10:         for all AlignmentPairs $\langle a_1, a_2 \rangle \in ap$ do
11:             for all GenomicLoci $l \in$ LOCI $(a_1, a_2)$ do
12:                 ReadPairInfo $rpi \leftarrow$
                         $\langle$InsertSize$(a_1, a_2)$, AlignScore$(a_1, a_2)\rangle$
13:                 EMIT(GenomicLocus $l$, ReadPairInfo $rpi$)
14:     function REDUCE(GenomicLocus $l$, ReadPairInfos $rpi_{1,2,\ldots}$)
15:         SVFeatures $\phi_l \leftarrow$
                 $\Phi$(InsertSizes $i_{1,2,\ldots}$, AlignmentScores $q_{1,2,\ldots}$)
16:         EMIT(GenomicLocus $l$, SVFeatures $\phi_l$)
17: job CALL SVS
18:     function MAP(GenomicLocus $l$, SVFeatures $\phi_l$)
19:         EMIT(Chromosome$(l)$, $< l, \phi_l >$)
20:     function REDUCE(Chromosome $c$,
                 GenomicLocus $l_{1,2,\ldots}, \phi_{1,2,\ldots}$)
21:         StructuralVariationCalls $svs_c \leftarrow$ POSTPROCESS $(\phi_{1,2,\ldots})$
```

the alignments and the likelihood that the alignments are correct. We then leave two functions to be implemented depending on the application:

$$\text{LOCI}\ : \langle a^1_{m,i}, a^2_{m,j} \rangle \rightarrow L_m \subseteq L$$

$$\Phi : \{\text{ReadPairInfo } rpi_{m,i,j}\} \rightarrow \mathbb{R}^N$$

The first function, LOCI, maps an alignment pair to a set of genomic locations to which it is relevant for SV detection; for example, the set of locations overlapped by the internal insert implied by the read alignments. We optimize this step by assuming that if there exist concordant mappings for a read pair, defined as those where the two alignments are in the proper orientation and with an insert size within three standard deviations of the expected library insert size, one of them is likely to be correct and therefore we do not consider any discordant alignments of the pair. The second function, Φ, maps a set of ReadPairInfos relevant to a given location to a set of real-valued vectors of features useful for SV detection.

Finally, the third MapReduce job, CALL VARIANTS, is responsible for making SV calls based on the features computed at each genomic location. It calls another application-specific function:

$$\text{POSTPROCESS} : \{\phi_1, \phi_2, \ldots, \phi_N\} \rightarrow \{\langle \text{SVType } s, l_{start}, l_{end} \rangle\}$$

This function maps the sets of features for related loci into a set of SV calls characterized by their type s (i.e Deletion, Insertion, etc.) and their breakpoint locations l_{start} and l_{end}. We parallelize this job in MapReduce by making calls for each chromosome in parallel, which we achieve by associating a location and its set of features to its chromosome in the map phase, and then making SV calls for one chromosome in each reduce task.

2.2 Cloudbreak: Detecting Deletions and Insertions

We have implemented an RP-based SV detection package, Cloudbreak, in the framework described above. Cloudbreak detects deletions of size 40bp to 25kb, and small insertions. We accomplish this by providing the following implementations of the user-defined functions described above:

LOCI: Because we are detecting deletions and short insertions, we map ReadPairInfos from each possible alignment to the genomic locations overlapped by the implied internal insert between the reads. For efficiency, we define a maximum detectable deletion size of 25,000bp, and therefore alignment pairs in which the ends are more than 25kb apart, or in the incorrect orientation, map to no genomic locations.

Φ: To compute features for each genomic location, we follow Lee *et al.* (2009), who observed that if all mappings are correct, the insert sizes implied by mappings which span a given genomic location should follow a Gaussian mixture model (GMM) whose parameters depend on whether a deletion or insertion is present at that locus (Supplementary Figure 1). Briefly: if there is no indel, the insert sizes implied by spanning alignment pairs should follow the distribution of actual fragment sizes in the sample, which is typically modeled as normally distributed with mean μ and standard deviation σ. If there is a homozygous deletion or insertion of length l at the location, μ should be shifted to $\mu + l$, while σ will remain constant. Finally, in the case of a heterozygous event, the distribution of insert sizes will follow a mixture of two normal distributions, one with mean μ, and the other with mean $\mu + l$, both with an unchanged standard deviation of σ, and mixing parameter α that describes the relative weights of the two components. Because the mean and standard deviation of the fragment sizes are selected by the experimenter and therefore known *a priori* (or at least easily estimated based on a sample of alignments), we only need to estimate the mean of the second component at each locus, and the mixing parameter α. We fit these parameters using an Expectation-Maximization algorithm (see Supplementary Material). To remove incorrect mappings, we use an adaptive quality cutoff for each genomic location and then perform an outlier-detection based noise reduction technique; these procedures also allow us to process multiple mappings for each read if they are reported by the aligner (see Supplementary Material). The features generated for each location l include the log-likelihood ratio of the filtered observed data points under the fit GMM to their likelihood under the distribution $N(\mu, \sigma)$, the final value of the mixing parameter α, and μ', the estimated mean of the second GMM component.

POSTPROCESS: We convert our features along each chromosome to insertion and deletion calls by first extracting contiguous genomic loci where the log-likelihood ratio of the two models is greater than a given threshold. To eliminate noise we apply a median filter. We end regions when μ' changes by more than 2σ, and discard regions where the average value of μ' is less than μ or where the length of the region differs from μ' by more than μ.

An illustration of the Cloudbreak algorithm working on a simple example is shown in Supplementary Figure 2.

2.3 Implementation Details

Cloudbreak is a native Java Hadoop application. We deployed Cloudbreak on a 56-node cluster running the Cloudera CDH3 Hadoop distribution, version 0.20.2-cdh3u4. We use snappy compression for MapReduce data. Hadoop's distributed cache mechanism shares the executable files and indices needed for mapping tasks to the nodes in the cluster.

Cloudbreak's alignment job can run a variety of alignment tools that report reads in SAM format (Supplementary Materials). In the map phase, mappers align reads in either single-end or paired-end mode to the reference genome in parallel, outputting mapping locations as values under a key identifying the read pair. In the reduce phase, the framework combines the reported mapping locations for the two ends of each read pair. This job can also be skipped in favor of importing a pre-existing set of mappings directly into the Hadoop cluster.

2.4 Cloudbreak Genotyping

We used the parameters of the fit GMM to infer the genotype of each predicted variant. Assuming that our pipeline is capturing all relevant read mappings near the locus of the variant, the genotype should be indicated by the estimated parameter α, the mixing parameter that controls the weight of the two components in the GMM. By tuning on the simulated data, we determined that by setting a simple cutoff of .35 on the average value of α for each prediction, we were able to accurately (see Section 3.4) call the predicted variant homozygous or heterozygous. We used the same cutoff for deletion and insertion predictions.

2.5 Running Locally or in the Cloud

Cloudbreak can be executed on any Hadoop cluster; Hadoop abstracts away the details of cluster configuration, making distributed applications portable. In addition, our Cloudbreak implementation can leverage the Apache Whirr library to automatically create clusters with cloud service providers such as the Amazon Elastic Compute Cloud (EC2). This enables on demand provisioning of Hadoop clusters which can then be terminated when processing is complete, eliminating the need to invest in a standing cluster and allowing a model in which users can scale their computational infrastructure as their need for it varies over time. In addition, Whirr supports multiple cloud service providers, making it possible to leverage many public or private clouds and avoid vendor lock-in. Instructions and examples describing how to leverage cloud computing with Cloudbreak are available in the user manual and Supplementary Materials.

3 RESULTS

3.1 Tests with Simulated Data

We compared the performance of Cloudbreak for detecting deletions and insertions to a selection of popular tools: the RP method BreakDancer (Chen *et al.*, 2009), GASVPro, an RP method that integrates RD signals and ambiguous mappings (Sindi *et al.*, 2012), the SR method Pindel (Ye *et al.*, 2009), and the hybrid RP-SR method DELLY (Rausch *et al.*, 2012). DELLY produces two sets of calls, one based solely on RP signals, and the other based on RP calls that could be supported by SR evidence; we refer to these sets of calls as DELLY-RP and DELLY-SR. We also attempted to evaluate MoDIL on the same data. All of these methods detect deletions. Insertions can be detected by BreakDancer, Pindel, and MoDIL. See Supplementary Materials for details on how reads were aligned and each program was invoked. For all alignments we used BWA (Li and Durbin, 2009), although in testing Cloudbreak we have found that the choice of aligner, number of possible mapping locations reported, and whether the reads were aligned in paired-end or single-ended mode can have a variety of effects on the output of the algorithm (Supplementary Fig. 3).

There is no available test set of real Illumina sequencing data from a sample that has a complete annotation of SVs. Therefore, testing with simulated data is important to fully characterize an algorithm's performance characteristics. On the other hand, any simulated data should contain realistic SVs that follow patterns observed in real data. We took one of the most complete lists of SVs from an individual, the list of homozygous insertions and deletions from the genome of J. Craig Venter (Levy *et al.*, 2007), and used it to simulate a 30X read coverage data set for a diploid human Chromosome 2 with a mix of homozygous and heterozygous variants, with 100bp reads and a mean fragment size of 300bp (See Supplementary Material).

Figure 2 shows Receiver Operating Characteristic (ROC) curves of the performance of each algorithm at identifying variants larger than 40bp on the simulated data set, as well as the runtimes of the approaches tested, excluding alignment. See Supplementary Material for a description of how we identified correct predictions. All approaches show high specificity to deletions at stringent thresholds in this simulation. If specificity is relaxed slightly, Cloudbreak provides the highest levels of sensitivity, followed by DELLY. For insertions, Cloudbreak's performance is similar to Pindel, although Cloudbreak can achieve greater sensitivity at the same false positive rate by decreasing its threshold. Cloudbreak's runtime is half that of BreakDancer, the next fastest tool, processing the simulated data in under six minutes. (Of course, Cloudbreak uses many more CPUs as distributed algorithm. See Supplementary Material and Supplementary Table 1 for a discussion of runtimes and parallelization.) The output which we obtained from MoDIL did not have a threshold that could be varied to correlate with the trade-off between precision and recall and therefore it is not included in ROC curves; in addition, MoDIL ran for 52,547 seconds using 250 CPUs. Apart from the alignment phase, which is embarrassingly parallel, the feature generation job is the most computationally intensive part of the Cloudbreak workflow. To test its scalability we measured its runtime on Hadoop clusters made up of varying numbers of nodes and observed that linear speedups can be achieved in this portion of the algorithm by adding additional nodes until a point of diminishing returns is reached (Supplementary Fig. 4).

Choosing the correct threshold to set on the output of an SV calling algorithm can be difficult. The use of simulated data and ROC curves allows for some investigation of the performance characteristics of algorithms at varying thresholds. First, we characterized the predictions made by each algorithm at the threshold that gives them maximum sensitivity. The results are summarized in Table 1. MoDIL and Cloudbreak exhibited the greatest recall for deletions. Cloudbreak also has high precision at this threshold, and discovers many small variants. For insertions, Cloudbreak has the highest recall, although recall is low for all four approaches. Cloudbreak again identifies many small variants. Pindel is the only tool which can consistently identify large insertions, as insertions larger than the library insert size do not produce mapping signatures detectable by RP mapping. We also used the ROC curves to characterize algorithm performance when a low false discovery rate is required. Supplementary Table 2 shows the total number of deletions found when choosing a threshold that gives an FDR closest to 10% based on the ROC curve. At this more stringent cutoff, Cloudbreak identifies more deletions in every size category than any other tool. Insertion performance never reached an FDR of 10% for any threshold, so insertion predictions are not included in this table.

We designed Cloudbreak to optimize for sensitivity and specificity for SV events, as demonstrated in the ROC curves, as opposed to attempting to achieve high levels of breakpoint resolution. Supplementary Fig. 5 displays the breakpoint resolution of each of the methods tested on the simulated data set. Cloudbreak's resolution is lower than other methods because the independent calculation of features at each window does not allow the algorithm to keep track of the actual coordinates of the mapped reads from each pair, which other tools use to set bounds on the true breakpoint locations. In practice, RP-based predictions are typically validated using techniques such as targeted split read search, local assembly, or wet-lab methods, processes which refine their breakpoint predictions. Therefore, Cloudbreak attempts to maximize the accurate detection of the presence of SV events.

3.2 Tests with Biological Data

We downloaded a set of reads from Yoruban individual NA18507, experiment ERX009609, from the Sequence Read Archive. This sample was sequenced by Illumina Inc. on the Genome Analyzer II platform with 100bp paired end reads and a mean fragment size

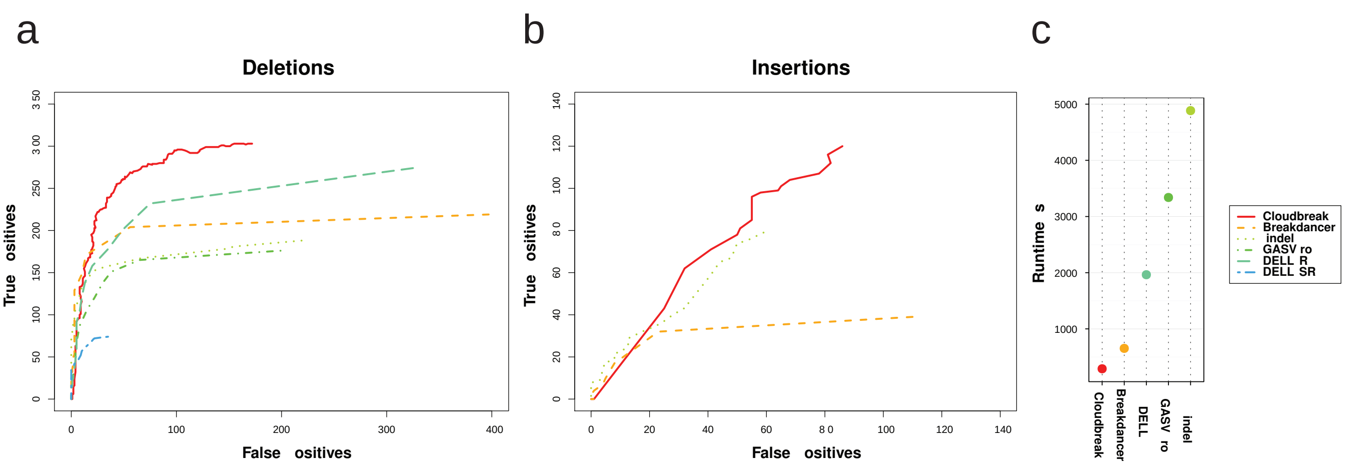

Fig. 2. Accuracy and runtime performance on a simulated data set. (a) Receiver Operating Characteristic (ROC) curves showing the specificity and sensitivity of each tool to deletions larger than 40bp on a simulated set of reads giving diploid coverage of 30X on human chromosome 2. Deletions from the Venter genome were randomly added to one or both haplotypes. Each point on a curve represents a different threshold on the confidence of predictions made by that tool. Thresholds vary by: Cloudbreak - likelihood ratio; BreakDancer, DELLY, GASVPro - number of supporting read pairs; Pindel - simple score. (b) ROC curves for insertion predictions. (c) Runtimes for each tool, not including alignment time, parallelized when possible (see Supplementary Material).

(minus adapters) of 300bp, with a standard deviation of 15bp, to a depth of approximately 37X coverage. To create a gold standard set of insertions and deletions to test against, we pooled annotated variants discovered by three previous studies on the same individual. These included data from the Human Genome Structural Variation Project reported by Kidd *et al.* (2008), a survey of small indels conducted by Mills *et al.* (2011b), and insertions and deletions from the merged call set of the phase 1 release of the 1000 Genomes Project (1000 Genomes Project Consortium *et al.*, 2012) which were genotyped as present in NA18507. We merged any overlapping calls of the same type into the region spanned by their unions. We were unable to run MoDIL on the whole-genome data set due to the estimated runtime and storage requirements.

Figure 3 shows the performance of each algorithm on the NA18507 data set. All algorithms show far less specificity for the gold standard set than they did in the single chromosome simulation, although it is difficult to tell how much of the difference is due to the added complexity of real data and a whole genome, and how much is due to missing variants in the gold standard set that are actually present in the sample. For deletions, Cloudbreak is the best performer at the most stringent thresholds, and has the highest or second highest precision at higher sensitivity levels. Cloudbreak has comparable accuracy for insertions to other tools, and can identify the most variants at higher levels of sensitivity. Cloudbreak processes the sample in under 15 minutes on our cluster, more than six times as fast as the next fastest program, BreakDancer, even when BreakDancer is run in parallel for each chromosome on different nodes in the cluster.

Given the high number of novel predictions made by all tools at maximum sensitivity, we decided to characterize performance at more stringent thresholds. We examined the deletion predictions made by each algorithm using the same cutoffs that yielded a

		Prec.	Recall	F1	40-100bp	101-250bp	251-500bp	501-1000bp	> 1000bp
	Total Number				224	84	82	31	26
Deletions	Cloudbreak	0.638	**0.678**	**0.657**	**153** (9)	61 (0)	62 (0)	12 (0)	15 (0)
	BreakDancer	0.356	0.49	0.412	89 (0)	54 (0)	53 (0)	8 (0)	15 (0)
	GASVPro	0.146	0.432	0.218	83 (2)	32 (0)	55 (0)	8 (0)	15 (0)
	DELLY-RP	0.457	0.613	0.613	114 (3)	**68** (0)	**66** (0)	9 (1)	17 (0)
	DELLY-SR	**0.679**	0.166	0.266	0 (0)	3 (0)	49 (0)	6 (0)	16 (0)
	Pindel	0.462	0.421	0.44	96 (**11**)	24 (0)	48 (0)	5 (0)	15 (0)
	MoDIL	0.132	0.66	0.22	123 (6)	66 (**3**)	**66** (**11**)	**17** (**7**)	**23** (**8**)
	Total Number				199	83	79	21	21
Insertions	Cloudbreak	0.451	**0.305**	**0.364**	**79** (**32**)	**32** (**18**)	**11** (8)	1 (0)	0 (0)
	BreakDancer	0.262	0.0968	0.141	23 (5)	14 (5)	2 (1)	0 (0)	0 (0)
	Pindel	**0.572**	0.196	0.292	52 (25)	5 (1)	10 (**9**)	**3** (**2**)	**9** (**9**)
	MoDIL	0.186	0.0521	0.0814	14 (1)	4 (0)	1 (0)	2 (**2**)	0 (0)

Table 1. The number of simulated deletions and insertions in the 30X diploid chromosome 2 with Venter indels found by each tool at maximum sensitivity, as well as the number of those variants that were discovered exclusively by each tool (in parentheses). The total number of variants in each size class in the true set of deletions and insertions is shown in the first row of each section.

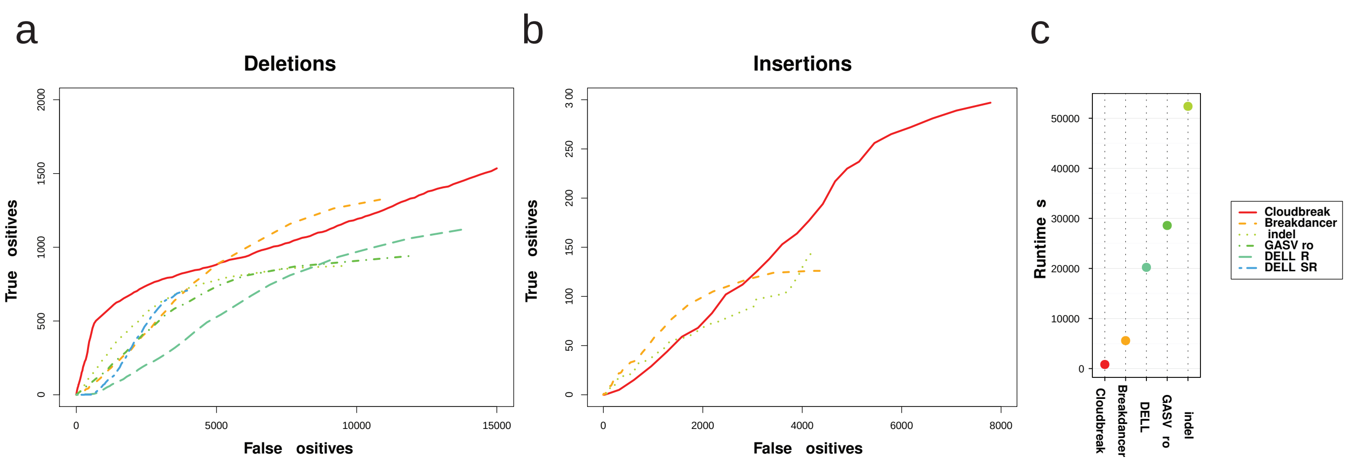

Fig. 3. Accuracy and performance on the 37X NA18507 sample. (a) ROC curve for deletion prediction performance, tested against the combined gold standard sets of deletions taken from Kidd *et al.* (2008), Mills *et al.* (2011b), and 1000 Genomes Project Consortium *et al.* (2012). (b) ROC curve for insertion prediction performance. (c) Runtimes for each tool, not including alignment time, parallelized when possible (see Supplementary Material).

10% FDR on the simulated chromosome 2 data set, adjusted proportionally for the difference in coverage from 30X to 37X. For insertions, we used the maximum sensitivity thresholds for each tool due to the high observed FDRs in the simulated data. Precision and recall at these thresholds, as well as the performance of each algorithm at predicting variants of each size class, is shown in Table 2. For deletions, Cloudbreak has the greatest sensitivity of any tool, identifying the most variants in each size class. Pindel exhibits the highest precision with respect to the gold standard set. For insertions, Pindel again has the highest precision at maximum sensitivity, while Cloudbreak has by far the highest recall.

3.3 Performance on a Low-Coverage Cancer Data Set

We also tested Cloudbreak on a data set obtained from a patient with acute myeloid leukemia (AML). This data consisted of 76bp paired end reads with a mean insert size of 285bp and standard deviation of 50bp, yielding sequence coverage of 5X and physical coverage of 8X. Using a pipeline consisting of Novoalign, BreakDancer, and a set of custom scripts for filtering and annotating candidate SVs, we had previously identified a set of variants present in this sample and validated several using PCR, including 8 deletions. Cloudbreak was able to identify all of the validated deletions, showing that it is still sensitive even when using lower coverage data with a greater variance of insert sizes. The variants identified include deletions in the gene *CTDSPL*/*RBPS3*, an AML tumor suppressor (Zheng *et al.*, 2012), and NBEAL1, a gene up-regulated in some cancers (Chen *et al.*, 2004). We are currently investigating these SVs to determine their functional impact on this patient.

3.4 Genotyping Deletions

Because Cloudbreak explicitly models zygosity in its feature generation algorithm, it can predict the genotypes of identified variants. We tested this on both the simulated and NA18507 data sets. For the NA18507 data set, we considered the deletions from the 1000 Genomes Project, which had been genotyped using the population-scale SV detection algorithm Genome STRiP (Handsaker *et al.*, 2011). Cloudbreak was able to achieve 92.7% and 95.9% accuracy in predicting the genotype of the deletions it detected at our 10% FDR threshold in the simulated and real data sets, respectively. Supplementary Table 3 shows confusion matrices for the two samples using this classifier.

4 DISCUSSION

Over the next few years, due to advances in sequencing technology, genomics data are expected to increase in volume by several orders of magnitude, expanding into the realm referred to as “big data”. In addition, the usage of existing information will increase drastically as research in genomics grows and translational applications are developed, and data is reprocessed and integrated into new pipelines. In order to capitalize on this emerging wealth of genome data, novel computational solutions that are capable of scaling with the increasing number and size of these data sets will have to be developed.

Among big data infrastructures, MapReduce is emerging as a standard framework for distributing computation across compute clusters. In this paper, we introduced a novel conceptual framework for SV detection algorithms in MapReduce, based on computing local genomic features. This framework provides a scalable basis for developing SV detection algorithms, as demonstrated by our development of an algorithm for detecting insertions and deletions based on fitting a GMM to the distribution of mapped insert sizes spanning each genomic location.

On simulated and real data sets, our approach exhibits high accuracy when compared to popular SV detection algorithms that run on traditional clusters and servers. Detection of insertions and deletions is an important area of research; Mills *et al.* (2011b) recently identified over 220 coding deletions in a survey of a large number of individuals, and they note that such variants are likely to cause phenotypic variation in humans.

In addition to delivering high accuracy and fast runtimes for a RP-based SV detection tool, our approach offers a basis for developing a variety of SV algorithms can run in MapReduce pipelines with

		Prec.	Recall	F1	40-100bp	101-250bp	251-500bp	501-1000bp	> 1000bp
	Total Number				7,462	240	232	147	540
Deletions	Cloudbreak	0.0943	**0.17**	0.121	**573 (277)**	**176 (30)**	**197 (18)**	**121 (6)**	**399 (24)**
	BreakDancer	0.137	0.123	**0.13**	261 (29)	136 (3)	178 (0)	114 (0)	371 (0)
	GASVPro	0.147	0.0474	0.0717	120 (21)	40 (2)	85 (0)	36 (0)	128 (0)
	DELLY-RP	0.0931	0.1	0.0965	143 (6)	128 (3)	167 (1)	103 (0)	323 (1)
	DELLY-SR	0.153	0.0485	0.0736	0 (0)	26 (0)	123 (0)	66 (0)	203 (0)
	Pindel	**0.179**	0.0748	0.106	149 (8)	61 (0)	149 (0)	69 (1)	217 (0)
Insertions	Total Number				536	114	45	1	0
	Cloudbreak	0.0323	**0.455**	0.0604	**265 (104)**	**49 (24)**	3 (1)	0 (0)	0 (0)
	BreakDancer	0.0281	0.181	0.0487	97 (10)	27 (5)	2 (1)	0 (0)	0 (0)
	Pindel	**0.0387**	0.239	**0.0666**	144 (45)	14 (7)	**7 (6)**	**1 (1)**	0 (0)

Table 2. The precision and recall with respect to the gold standard set of deletions and insertions for each tool on the NA18507 data, as well as the number of variants found in each size class found. Exclusive predictions are in parentheses. For deletions, the same cutoffs were used as for the simulated data as in Supplementary Table 2, adjusted for the difference in coverage from 30X to 37X. For insertions, the maximum sensitivity cutoff was used.

the power to process vast amounts of data in a cloud or commodity server setting. With the advent of cloud service providers such as Amazon EC2, it is becoming easy to instantiate on-demand Hadoop compute clusters. Having computational approaches that can harness this capability will become increasingly important for researchers or clinicians who need to analyze increasing amounts of sequencing data.

5 CONCLUSION

ACKNOWLEDGEMENT

We thank Izhak Shafran at the Center for Spoken Language Understanding for advice, support, and shared computational resources, and Bob Handsaker for helpful conversations.

Funding: C.T.S. and A.L.A. are supported by the Italian Association for Cancer Research (AIRC). Sequencing of the AML sample was supported by the Oregon Medical Research Foundation.

REFERENCES

1000 Genomes Project Consortium, *et al.* (2012). An integrated map of genetic variation from 1,092 human genomes. *Nature*, **491**(7422), 56–65.

Alkan, C., *et al.* (2009). Personalized copy number and segmental duplication maps using next-generation sequencing. *Nat. Genet.*, **41**(10), 1061–1067.

Alkan, C., *et al.* (2011). Genome structural variation discovery and genotyping. *Nat. Rev. Genet.*, **12**(5), 363–376.

Campbell, P. J., *et al.* (2008). Identification of somatically acquired rearrangements in cancer using genome-wide massively parallel paired-end sequencing. *Nat. Genet.*, **40**(6), 722–729.

Chen, J., *et al.* (2004). Identification and characterization of NBEAL1, a novel human neurobeachin-like 1 protein gene from fetal brain, which is up regulated in glioma. *Mol Brain Res*, **125**(1-2), 147–155.

Chen, K., *et al.* (2009). BreakDancer: an algorithm for high-resolution mapping of genomic structural variation. *Nat. Methods*, **6**(9), 677–681.

Chiang, D. Y., *et al.* (2009). High-resolution mapping of copy-number alterations with massively parallel sequencing. *Nat. Methods*, **6**(1), 99–103.

Chiara, M., *et al.* (2012). SVM^2: an improved paired-end-based tool for the detection of small genomic structural variations using high-throughput single-genome resequencing data. *Nucleic Acids Res.*, **40**(18), e145.

Conrad, D. F., *et al.* (2010). Origins and functional impact of copy number variation in the human genome. *Nature*, **464**(7289), 704–712.

Dean, J. and Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Commun. ACM*, **51**(1), 107–113.

Handsaker, R. E., *et al.* (2011). Discovery and genotyping of genome structural polymorphism by sequencing on a population scale. *Nat. Genet.*, **43**(3), 269–276.

Hormozdiari, F., *et al.* (2009). Combinatorial algorithms for structural variation detection in high-throughput sequenced genomes. *Genome Res.*, **19**(7), 1270–1278.

Kidd, J. M., *et al.* (2008). Mapping and sequencing of structural variation from eight human genomes. *Nature*, **453**(7191), 56–64.

Korbel, J. O., *et al.* (2009). PEMer: a computational framework with simulation-based error models for inferring genomic structural variants from massive paired-end sequencing data. *Genome Biol.*, **10**(2), R23.

Langmead, B., *et al.* (2009). Searching for SNPs with cloud computing. *Genome Biol.*, **10**(11), R134.

Lee, S., *et al.* (2009). MoDIL: detecting small indels from clone-end sequencing with mixtures of distributions. *Nat. Methods*, **6**(7), 473–474.

Levy, S., *et al.* (2007). The diploid genome sequence of an individual human. *PLoS Biol.*, **5**(10), e254.

Li, H. and Durbin, R. (2009). Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, **25**(14), 1754–1760.

Marschall, T., *et al.* (2012). CLEVER: clique-enumerating variant finder. *Bioinformatics*, **28**(22), 2875–2882.

McKenna, A., *et al.* (2010). The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res.*, **20**(9), 1297–1303.

Michaelson, J. J. and Sebat, J. (2012). forestSV: structural variant discovery through statistical learning. *Nat. Methods*, **9**(8), 819–821.

Mills, R. E., *et al.* (2011a). Mapping copy number variation by population-scale genome sequencing. *Nature*, **470**(7332), 59–65.

Mills, R. E., *et al.* (2011b). Natural genetic variation caused by small insertions and deletions in the human genome. *Genome Res.*, **21**(6), 830–839.

Quinlan, A. R., *et al.* (2010). Genome-wide mapping and assembly of structural variant breakpoints in the mouse genome. *Genome Res.*, **20**(5), 623–635.

Rausch, T., *et al.* (2012). DELLY: structural variant discovery by integrated paired-end and split-read analysis. *Bioinformatics*, **28**(18), i333–i339.

Schatz, M. C. (2009). CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics*, **25**(11), 1363–1369.

Sindi, S., *et al.* (2009). A geometric approach for classification and comparison of structural variants. *Bioinformatics*, **25**(12), i222–30.

Sindi, S. S., *et al.* (2012). An integrative probabilistic model for identification of structural variation in sequencing data. *Genome Biol.*, **13**(3), R22.

Wang, J., *et al.* (2011). CREST maps somatic structural variation in cancer genomes with base-pair resolution. *Nat. Methods*, **8**(8), 652–654.

Ye, K., *et al.* (2009). Pindel: a pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads. *Bioinformatics*, **25**(21), 2865–2871.

Yoon, S., *et al.* (2009). Sensitive and accurate detection of copy number variants using read depth of coverage. *Genome Res.*, **19**(9), 1586–1592.

Zheng, Y.-S., *et al.* (2012). MiR-100 regulates cell differentiation and survival by targeting RBSP3, a phosphatase-like tumor suppressor in acute myeloid leukemia. *Oncogene*, **31**(1), 80–92.