2017 IEEE 17th International Conference on Bioinformatics and Bioengineering

使用 Apache Spark 进行流式分布式 DNA 序列比对

哈米德·穆斯塔克　　努曼·阿赫迈德　　扎伊德·阿尔-阿尔斯
代尔夫特理工大学计算机工程实验室，2628 CD，荷兰代尔夫特，
{H.Mushtaq, N.Ahmed, Z.Al-Ars}@tudelft.nl

摘要—下一代测序（NGS）技术产生的大量数据，通常每个实验达数百 GB，必须快速分析以生成有意义的变异结果。分析此类数据的第一步是将测序读段映射到人类基因组中的相应位置。进行此类序列比对最流行的工具之一是 Burrows-Wheeler 对齐器（BWA mem）。然而，BWA 程序的局限性在于它不能在集群上运行。在本文中，我们提出了 StreamBWA，这是一个新的框架，允许 BWA mem 程序以分布式方式在集群上运行，同时输入数据正在被流式传输到集群。它可以直接从压缩文件中处理输入数据，该文件位于本地文件系统或 URL 上。此外，StreamBWA 可以在分布式 BWA mem 任务仍在集群上执行时，同时开始组合这些任务的输出文件。实证评估表明，这种流式分布式方法比非流式方法快约 2 倍。 此外，我们的流式分布式方法比其他最先进的解决方案（如 SparkBWA）快 5 倍。StreamBWA 的源代码可在 https://github.com/HamidMushtaq/StreamBWA 公开获取。

I. I

DNA 分析用于识别 DNA 中的突变，这些突变表明对某些疾病的特定易感性。此类分析的第一步是将测序读段映射到人类基因组中的相应位置。一个非常流行的工具是 Burrows-Wheeler 对齐器（BWA mem）[Li13]。然而，BWA mem 只能在单个节点上使用多个线程运行。

现有的最先进工具，如 SparkBWA [Abuin16]，确实允许以分布式方式在集群上运行 BWA mem。然而，SparkBWA 需要数据在 HDFS（Hadoop 分布式文件系统）上可用。由于通常输入文件是以 gzip 格式提供的，这需要在将其上传到 HDFS 之前先解压缩文件。随后，这也减缓了 BWA 本身的执行速度，因为 HDFS 上的数据必须重新格式化为集群上运行的 BWA 程序任务的适当输入。最后，这些 BWA 任务生成的输出文件在最后是分别合并的，这也需要大量时间。

在本文中，我们提出了一种新的分布式框架，StreamBWA，它允许在集群上运行 BWA mem，同时输入数据直接从压缩文件中流式传输。该文件可以位于主节点上，也可以位于 URL 上，例如 https 或 ftp 站点。这消除了分别花费执行时间来下载文件和解压缩文件的需要。此外，由于主节点可以将数据流式传输到数据节点，因此将数据上传到 HDFS 的开销也可以被隐藏。一旦输出文件可用，主节点还可以并行地开始组合数据节点上运行的 BWA 任务的输出文件，进一步减少总体时间。实验结果表明，与 SparkBWA 相比，StreamBWA 在 4（+1 个主节点）的集群上对选定的数据集快了近 5 倍。

本文的贡献如下。

- 我们实现了 StreamBWA：一个在 Spark 集群上运行 BWA 的框架，其中输入数据并行流式传输到执行 BWA mem 任务的数据节点。
- StreamBWA 还能够以流式方式组合由 BWA 任务生成的输出文件。
- StreamBWA 通过无需重新格式化输入数据来运行 BWA 任务，从而提高了效率（与 SparkBWA 不同）。

本文结构如下。第二节讨论大数据技术、DNA 分析管道的不同阶段及相关工作。第三节介绍我们的流式方法。接下来，第四节讨论 StreamBWA 的实现。随后是第五节，展示评估结果和性能分析。最后，第六节总结全文。

II. B

在本节中，首先我们讨论可用于在可扩展计算基础设施上执行并行应用程序的大数据技术，然后我们讨论典型的 DNA 分析管道。最后，我们讨论相关工作。

A. 大数据技术

MapReduce [Dean08] 编程模型一直是处理需要在一个可扩展集群中的多个计算节点上进行处理的数据密集型计算管道的最常用的大数据方法之一。

该模型将计算分为两个阶段：映射和归约。在映射阶段，输入数据首先被格式化为键值 对，然后对所有这些对执行特定的映射函数，将输入映射到一个输出 对集合。映射函数以分布式方式执行，使用各种映射任务在集群节点的本地数据上运行。输出

2471-7819/17/31.00 ©2017 IEEE DOI
10.1109/BIBE.2017.00038

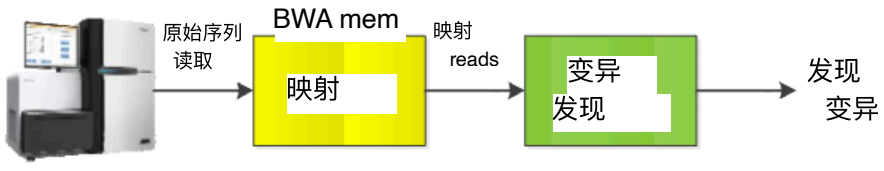

图 1。 典型的 DNA 分析和变异发现流程

随后由 reduce 任务处理，这些任务首先通过将属于同一键的所有值分组来洗牌数据。之后，reduce 任务从分组好的对中计算出一个输出。Apache Hadoop 是 MapReduce 编程模型的开放源代码实现的一个例子。

MapReduce 框架的一个缺点是它在两个阶段（map 和 reduce）之间存储所有生成数据在磁盘上，如果将多个 map/reduce 阶段串联起来，它会将每个阶段的输出存储在 HDFS 上。这会导致由于密集磁盘访问而产生显著的开销。另一个缺点是只允许 map 和 reduce 这两种转换。如果需要应用不同的转换，就必须通过修改 map 和 reduce 函数来完成，因此使用这个框架进行开发非常繁琐。

Apache Spark 是一个较新的处理大数据的框架，它解决了上面提到的 MapReduce 的缺点。首先，它允许比单纯的 map 和 reduce 更多的转换。它提供了诸如 join、cogroup、intersection、distinct 以及其他许多预定义操作。这使得程序开发比 MapReduce 框架更加容易。此外，每个转换的输出会保存在内存中，但仍然允许使用磁盘来保存那些不适合内存大小的数据。通过这种方式，Spark 避免了 MapReduce 框架中普遍存在的磁盘访问开销。另外，Spark 提供了一个编程接口，可以在 Scala、Python 和 Java 等多种编程语言中实现算法，这允许使用最适合问题的语言来编写程序。

B. DNA 分析流程

图 1 展示了一个典型的 DNA 分析和变异发现流程。该流程的输入数据集是原始测序读段，这些读段是从 DNA 测序机中获取的。由于测序机通常会将 DNA 进行高达 30 倍到 100 倍的超量采样，因此一个典型文件中的读段数量可能非常大，因此存储这些读段的文件体积也很大，对于人类基因组来说，通常在几百 GB 的范围内。目前用于存储这些读段的一种标准文件格式是 FASTQ 文件格式[Jones12]。

DNA 分析流程的第一步是 DNA 映射。在这一步中，原始读段被映射到参考基因组上。可以使用许多经典的对齐工具，例如 Bowtie2[Langmead12]或流行的 BWA mem。对于 StreamBWA，我们使用了未经修改的 BWA 程序，因为

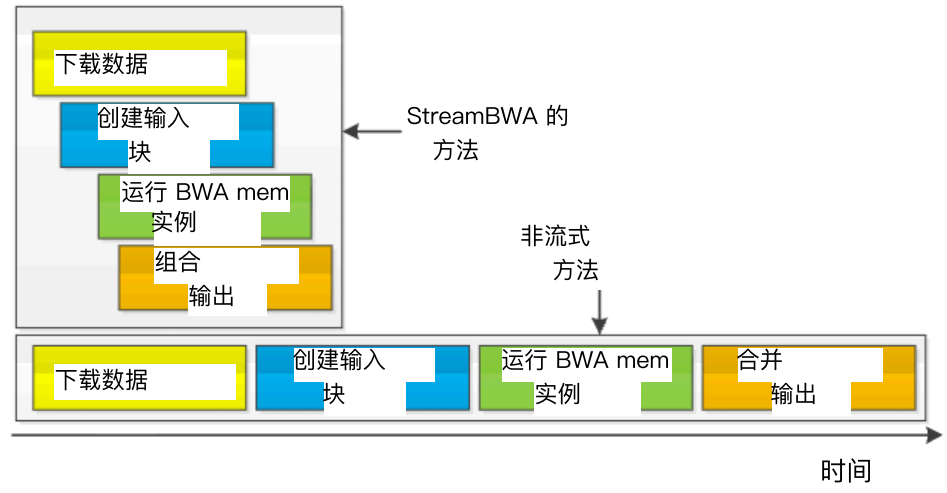

图 2。 流式处理与非流式处理方法的时序图

BWA 可执行文件由数据节点上的任务直接执行。这也使我们能够使用硬件加速的 BWA mem 实现，例如[Ahmed15]。使用 StreamBWA 时，输出文件（SAM 格式）也可以由主节点并行合并。这些文件还可以合并到单独的染色体中，甚至可以负载均衡的染色体区域。允许这种平衡输出的好处是，DNA 分析管道中的变异发现后续工具可以并行高效地处理这些负载均衡的区域。

C. 相关工作

近年来，提出了许多框架来解决 DNA 分析流程的可扩展性问题。虽然一些方法适用于整个流程并使用大数据可扩展性技术，如 SparkGA [Mushtaq17] 和 Halvade [Decap15]，但其他方法使用集成集群可扩展性技术，如 Churchill [Kelly15]。还有一些专注于 DNA 分析流程单个阶段的可扩展性框架，如 BigBWA [Abuin12] 和 SparkBWA。BigBWA 提出了一种基于 MapReduce 的大数据解决方案来运行 BWA mem，而在 SparkBWA 中，相同的作者使用内存中的 Apache Spark 框架来运行 BWA mem。在这项工作中，我们提供了一个针对 DNA 映射步骤的 Apache Spark 大数据可扩展性解决方案，类似于 SparkBWA。然而，我们的工具允许在输入文件仍在逐步流式传输时开始映射。这隐藏了大型 FASTQ 文件数据传输的高成本。类似地，使用我们的工具，输出 SAM 文件的组合也是在 DNA 映射执行的同时并行完成的。

III. 流式处理方法

图 2 展示了我们的流式方法如何隐藏与下载、解压缩、分发和合并文件相关的延迟。该图显示了我们的流式方法与非流式方法的时序对比。在非流式方法中，所有这些操作都是顺序执行的，从而引入了延迟，而 StreamBWA 能够并行执行所有这些操作。

图 3 展示了我们提出的流式方法的数据流。为了将 FASTQ 文件或两个配对的 FASTQ 文件分发到多个节点，我们需要首先将它们分解成数据块。这是通过我们构建的块处理程序完成的，其源代码可以在 https://github.com/HamidMushtaq/FastqChunker 找到。FASTQ 块处理程序在主节点上运行，可以从 URL（如 ftp 或 https 站点）或直接从文件读取数据。对于配对的 FASTQ 文件，块处理程序有一个选项可以将数据交错成单个块。此外，输入的 FASTQ 文件可以是未压缩的或 gzip 压缩的。FASTQ 块处理程序会持续从文件/URL 读取数据，解压缩数据，并在压缩块后上传块。块的压缩可以通过使用多个线程来完成。当 FASTQ 块处理程序上传一个块后，它还会上传一个相应的文件来通知 StreamBWA 程序该块已经准备好。一旦一个块被创建，如果所有资源没有被其他任务占用，StreamBWA 中的 BWA 任务将能够立即处理它。

当一个 BWA 任务输出一个 SAM 文件时，它还会上传一个相应的标志，通知 StreamBWA 的 SAM 文件组合部分，该部分的任务是将 SAM 文件组合成一个。StreamBWA 也可以为每个染色体创建一个 SAM 文件，甚至可以为染色体区域创建 SAM 文件。当 BWA 任务在 Spark 集群上运行时，StreamBWA 的 SAM 文件组合部分会并行运行。

由于 chunker 是一个独立的程序，它也可以与其他程序一起使用。例如，我们成功地将 chunker 工具与 SparkGA 程序结合使用，使得像 StreamBWA 一样，SparkGA 也能够实时处理数据块。

IV. I

我们的框架由两个工具组成。一个是 StreamBWA 工具，另一个是 chunker 程序。chunker 读取输入的 gzipped 文件，创建压缩的数据块并将它们上传到 HDFS。StreamBWA 实时读取这些数据块，并将生成的 SAM 文件组合成一个单一的输出文件，或者为每个染色体创建组合输出文件，或者按染色体区域分组创建文件，每个文件代表一个这样的染色体区域。接下来将讨论 chunker 和 StreamBWA。

A. 分块器

分块器工具使用 Scala 编写，并利用 Future 操作通过并行线程创建压缩块。在一次迭代中，它从本地存储或 URL 存储的文件中读取 N 个块，其中 N 是 Future（线程）的数量。读取这些 N 个块后，每个块都会分配给一个单独的线程。这样，N 个并行线程可以

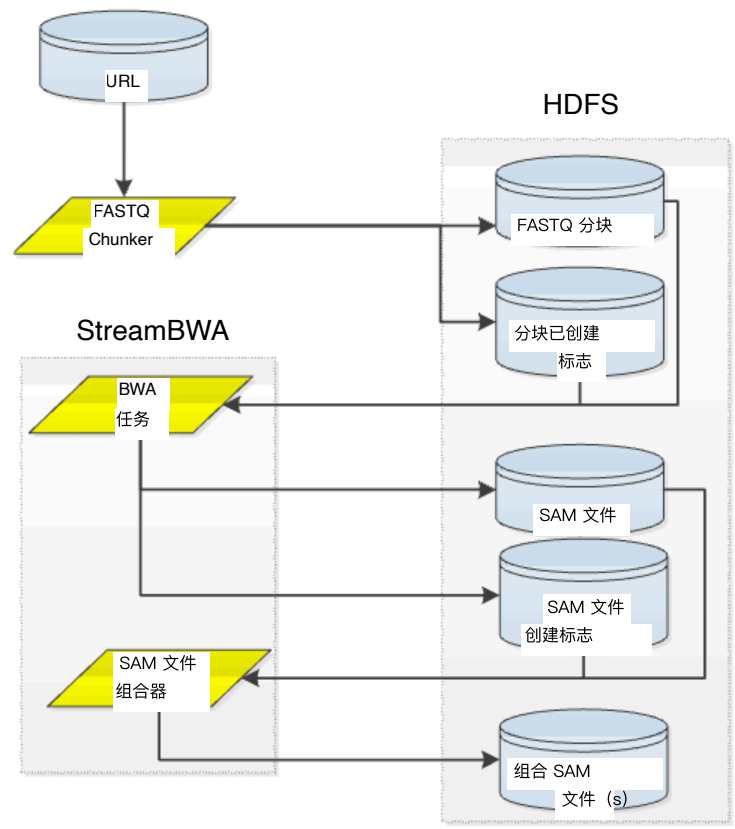

图 3。 所提出流式方法的数据流

压缩块并上传到 HDFS。Halvade 上传工具是一个类似的工具。然而，与 Halvade 上传工具不同，后者逐行读取数据，我们的工具一次从 gzip 压缩文件中读取一个未压缩的数据块，并在该块的末尾查找读取边界。也就是说，它检查最后一个读取结束的位置，获取到最后一个读取的数据，并将剩余部分放入一个缓冲区，我们称之为剩余缓冲区。然后从这个剩余缓冲区中的数据被追加到下一个块的数据中。逐块读取数据而不是逐行读取数据，是我们分块工具性能明显优于 Halvade 上传工具的原因，如评估结果所示。

上传压缩块到 HDFS 后，还会上传一个状态文件。这个状态文件只是一个带有 ID 的空文件，用于通知 StreamBWA 程序相应的块已经上传。ID 只是数字，从 0 开始。如果使用 N 个线程，首先会上传从 0 到 N-1 的块，然后是 N 到 N*2-1 的块，以此类推。这个分块程序在主节点上本地运行。当所有块都上传完毕后，还会发送一个哨兵文件，以通知所有文件已上传完毕。

用于配对端 FASTQ 文件分块的算法如算法 1 所示。这里，bArrArr1 和 bArrArr2 是类型为 ByteArray 的数组，ByteArray 是我们创建的类，用于存储每次迭代中读取的数据。这些元素一次性分配足够大的大小，以便可以反复使用，从而避免每次都创建新元素。反复使用 new 会增加内存和垃圾回收的开销。类似地，gis1 和 gis2 是类型为 GZInput 的对象，我们创建它们来读取 gzip 压缩数据。它不同于标准的 GZInputStream 类，因为它可以读取固定数量的未压缩字节。

算法 1：配对 FASTQ 分块

```
while !endReached do
    for i = 0 until nThreads do
        如果已到达末尾，则
            bArrArr1(i).setLen(0)
            bArrArr2(i).setLen(0)
            bytesRead(i) = -1
7 else
            bytesRead(i) = gis1.read(tmpBuf1)
            gis2.read(tmpBuf2)
10 if bytesRead(i) == -1 then
                endReached = true
                bArrArr1(i).copyFrom(leftOver1)
                bArrArr2(i).copyFrom(leftOver2)
14 else
                /* leftOver1 在第一次迭代时为 null
                                                    */
16 if leftOver1 == null then
                    leftOver1 = new ByteArray(bufSize)
                    leftOver2 = new ByteArray(bufSize)
                    bArr1.copy(tmpBuf1, 0, bytesRead(i))
                    bArr2.copy(tmpBuf2, 0, bytesRead(i))
21 else
                    bArr1.copy(leftOver1)
                    bArr1.app(tmpBuf1, 0, bytesRead(i))
                    bArr2.copy(leftOver2)
                    bArr2.app(tmpBuf2, 0, bytesRead(i))
                splitAtRB(bArr1, bArrArr1(i), leftOver1)
                splitAtRB(bArr2, bArrArr2(i), leftOver2)
28 if f(i) != null then
            Await.result(f(i), Duration.Inf)
30 gzOutStreams(i).synchronized {
                baFuture1(i).copy(bArrArr1(i))
                baFuture2(i).copy(bArrArr2(i))
            }
34 f(i) = Future {
                gzipOutStreams(i).synchronized {
                    写入到块(i)
                }
            }
    等待 Future 并关闭 GZOutSreams()
```

在构造函数中，我们可以指定块大小，然后 GZInputStream 每次会读取相同数量的字节（未压缩）。算法的工作方式是，在每次迭代中，我们为每个 Future 读取块，而 Future 的数量等于 nThreads。从每个 FASTQ 文件读取的数据会在最后一次读取时进行分割，以确保没有不完整的读取。最后剩余的读取会被放入剩余缓冲区。剩余缓冲区中的数据会被追加到下一次迭代的块的开头。之后，读取的数据会被复制到 baFuture1 和 baFuture2，这两个缓冲区被传递给相应的 Future。

一个块的数据然后在 Future 中交错，如算法 2 所示。当输出流中的数据大于允许的块大小时，它会被写入到块中。下次数据会被写入到另一个块中。

当未来的 i 正在交错数据并写入到一个块时，迭代 i 可以并行地从输入的 gzip 压缩的配对端 FASTQ 文件中获取数据。通过这种方式，当下一个 future i 的实例开始时，其输入数据可能已经准备好了。通过这种机制，我们能够完全或部分隐藏压缩块并上传到 HDFS 所需的时间。

算法 2：交错块的写入

```
如果 baFuture1(i).getLen() != 0 then
    content = interleave(baFuture1(i), baFuture2(i))
    gzOutStreams(i).write(content)
4 if gzOutStreams(i).getSize() > chunkSize then
        gzOutStreams(i).close()
        data = gzipOutStreams(ti).getByteArray
        writeChunkFile(chunkCtr(i) + ".fq.gz", data)
        writeStatusFile(chunkCtr(i))
        chunkCtr(i) = chunkCtr(i) + nThreads
        baos = new ByteArrayOutputStream
        gzOutStreams(i) = new GZIPOutputStream1(baos,
        compLevel)
```

由于分块器工具是一个独立的程序，而不是 StreamBWA 的一部分，这意味着它也可以与其他基因组管道工具一起使用。例如，我们已经成功地将分块器工具与我们的 SparkGA 工具接口，以在主节点上并行执行分块操作。

B. StreamBWA

StreamBWA 程序可以运行在客户端或者以集群模式运行。在客户端模式下，StreamBWA 的控制器部分（称为驱动程序）在主节点上运行，而执行程序在数据节点上运行。另一方面，在集群模式下，驱动程序也在其中一个数据节点上运行。

数据节点上的每个 BWA 任务被分配一个 ID。BWA 任务被调度为按 ID 从小到大依次执行。每个任务在执行前会等待对应的分块存在于 HDFS 中。这可以通过查看相应的状态文件来实现。一旦状态文件出现，任务就会获取对应的分块，解压缩并使用它执行 BWA。这意味着在初始阶段，第一组任务需要等待分块器完成分块。然而，当调度下一组任务时，分块器可能已经为它们准备好了分块，因为分块器在主节点上并行持续运行。如果出现一个 ID 大于分块总数的任务，该任务会通过查看分块器已上传的哨兵文件，而未生成具有该 ID 的文件，从而知道没有对应的分块可以处理。

如前所述，StreamBWA 程序还可以将输出分块合并为一个输出文件，或按染色体或染色体外区域分组为多个文件。此外，这些合并的输出文件可以存储在 HDFS 上，或存储在主节点的本地文件系统上。

表 I
C OMPARISON OF OUR CHUNKING UTILITY WITH H

TOOL (CL = 压缩级别)

程序	D1 分块-分块时间 (分钟)	D1 块大小 (GB)	D2 块 ing time (mins)	D2 块大小 (GB)
Halvade 上传工具（cl: 6） 53.5		74.7	20	28.4
StreamBWA (cl: 6)	39.5	74.7	13	28.4
StreamBWA (cl: 5)	33	77.5	12	29
StreamBWA (cl: 4)	32.5	79.2	12	29.5
StreamBWA (cl: 3)	32.5	80.5	12	30.4

合并器部分使用 Scala Futures 实现，运行在 StreamBWA 的驱动部分。一旦某个任务完成 BWA 的执行，它就会将输出 SAM 文件上传到 HDFS。除此之外，它还会上传一个状态文件。合并器会持续查找这些状态文件，以知道何时合并这些输出 SAM 文件。为此，合并器维护一个 Set 数据结构，其中包含所有已合并 SAM 文件的 ID。通过比较状态文件的 ID 和 Set 中的 ID 的差异，合并器知道要合并哪些文件。它重复执行此步骤，直到所有文件都已合并。为了提高效率，可以使用多个 Future。当需要使用合并器时，每个运行 BWA 的任务都会输出一个压缩的 SAM 文件。通过使用多个 Future（线程），可以并行解压多个压缩的 SAM 文件。

V. 评估结果

我们在一个由 4 个数据节点+1 个主节点组成的 IBM Power7+集群上测试了结果。每个节点有两个插槽，每个插槽安装一个 Power7+处理器。每个节点总共有 16 个物理核心和 128GB 内存。每个 Power7+核心支持 4 路同时多线程。Spark 在该平台上通过 YARN 运行。尽管我们集群中的主节点也有 128GB 内存，但我们限制了主节点内存使用量不超过 16GB。这是因为，在典型场景中，主节点与数据节点相比功能较弱，因为主节点大部分时间都处于空闲状态。由于我们的框架利用了主节点的这种空闲状态，且没有增加额外成本，为了模拟这种情况，我们将主节点的内存使用量保持在 16GB 以下。

我们使用数据集 D1（来自[NA12878]的 NA12878D HiSeqX R1 和 NA12878D HiSeqX R2）和数据集 D2（来自[1000genomes]数据/HG00109/sequence read 文件夹的 ERR022066 1.filt 和 ERR022066 2.filt）测试并比较了我们的框架与其他解决方案。D1 的总未压缩大小为 272GB，而 D2 为 90GB。

A. 分块工具的评估

表 I 比较了我们分块工具与 Halvade 上传工具的性能。这些结果也展示在图 4 中。表和图显示，对于

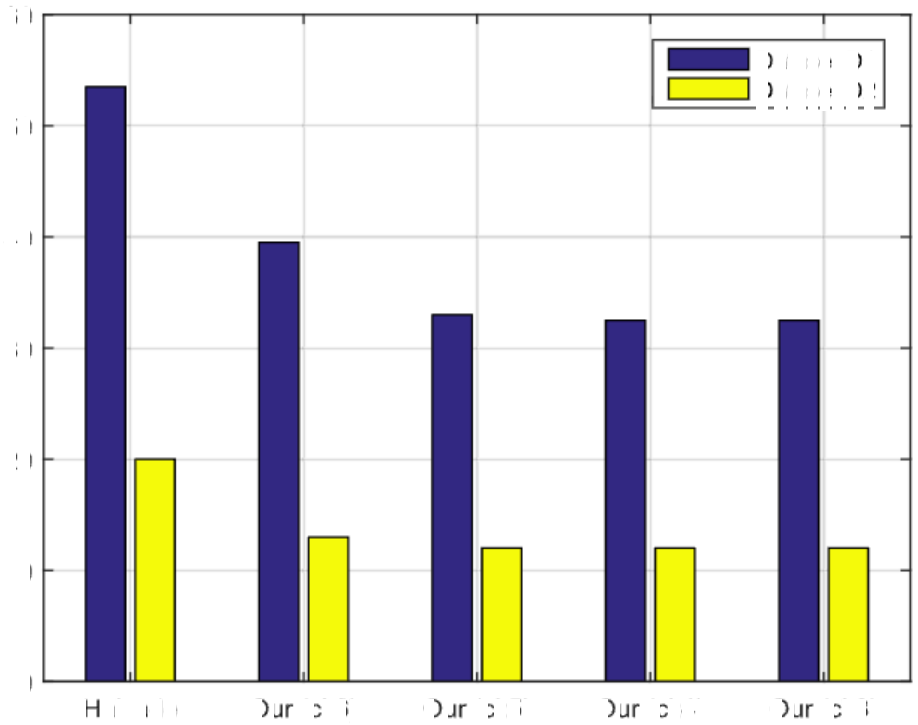

图 4. Halvade 的 chunker 工具与我们的 chunker 工具的性能比较（cl = 压缩级别）

表 II
C S BWA N - S BWA

AND S BWA

流程	D1、 大小 = 272GB（最小，% of total)	D2， size = 90GB (mins, % of total)
SparkBWA		
下载	45 (7%)	12.5 (12%)
解压缩	127 (20%)	30 (29%)
上传至 HDFS 19 (3%) 6 (6%) BWA	368 (59%) 33.5 (33%)	合并 70 (11%)
21 (20%)		
总计	629	103
非流式 SreamBWA		
下载	45 (18%)	12.5 (25%)
切分器	32.5 (13%)	12 (24%)
BWA	104 (41%)	5 (10%)
合并	70 (28%)	21 (42%)
总计	251.5	50.5
流式处理 StreamBWA		
分块器 ‖ BWA	105 (79%)	12 (65%)
分块器 ‖ BWA ‖ 合并 120 (91%)	15.5 (84%)	
下载 ‖ 分块器 ‖ BWA ‖ 合并	132.5	20

在两个数据集中，我们的分块工具比 Halvade 上传工具的性能高出最多 35%（在相同的压缩级别下）。这种性能提升的原因是输入数据的处理量减少。虽然 Halvade 逐行读取输入的 gzipped 数据，但我们的分块工具可以读取数据块，并在读取边界上正确标记它们。

结果还显示，通过将压缩级别从默认的 6 降低，我们获得了更高的速度提升，最高可达 65%，代价是文件稍大。这种速度提升是由于在较低压缩级别下 gzip 压缩速度更快，这显然是原因。我们无法在较低压缩级别下与 Halvade 上传工具进行比较，因为它只允许默认压缩级别。

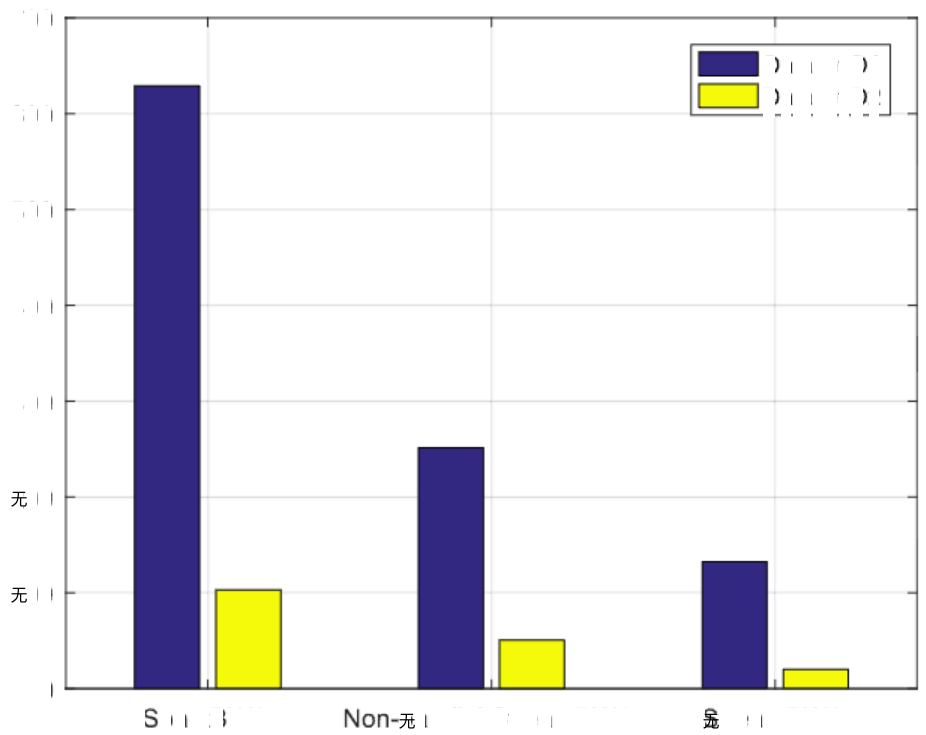

图 5. StreamBWA 与非并行方法及 SparkBWA 的比较

B. StreamBWA 与 SparkBWA 的比较

表 II 展示了 StreamBWA 与其非并行版本及 SparkBWA 的比较。每种解决方案的总时间以粗体显示，并在图 5 中绘制。结果表明 StreamBWA 比 SparkBWA 快约 5 倍。主要原因在于 StreamBWA 能够并行执行解压缩/上传、BWA 映射和输出合并，而 SparkBWA 必须按顺序执行这些操作。此外，即使数据已上传到 HDFS，SparkBWA 还必须将数据分组为输入文件以提供给 BWA 任务。为此，它必须首先通过昂贵的操作（如 join 和 sortByKey）创建 RDD。因此，即使 StreamBWA 的非并行版本也比 SparkBWA 快 2.5 倍，因为它不需要这一额外步骤。

VI. C

在这篇论文中，我们介绍了 StreamBWA，一个开源工具（可在 https://github.com/HamidMushtaq/StreamBWA 获取），它可用于在基于 Spark 的集群上执行 Burrows-Wheeler 对齐器（BWA mem）算法。StreamBWA 不仅能够在集群上以分布式方式运行 BWA mem 任务，还能实时将数据流传输到运行在数据节点上的 BWA mem 任务。此外，它还能以流式方式将 BWA mem 任务的输出文件合并为一个或多个 SAM 文件。结果表明，这种流式分布式方法比非流式方法快约 2 倍。此外，由于不像 SparkBWA 等其他最先进方法那样，需要以特定方式格式化数据，即使数据上传到 HDFS 后也是如此，StreamBWA 在将数据上传到 HDFS 之前就进行格式化，因此进一步节省了时间。由于这个原因，即使是 StreamBWA 的非流式版本也比 SparkBWA 快高达 2.5 倍。

在选定的数据集上，流式版本的速度大约是 5 倍。最后，与 SparkBWA 不同，除了允许输出合并到一个文件中，StreamBWA 还允许按染色体或甚至负载均衡的染色体区域对输出数据进行分组。将输出安排在这样的染色体区域中，有助于优化 DNA 分析管道后续工具的性能。

R

[Ahmed15] N. Ahmed, 等, ”异构硬件/软件
“BWA-MEM DNA 比对算法的加速”，ICCAD’15，第 240-246 页，2015 年。

[Auwera13] G.A. van der Auwera, 等人, ”从 FASTQ 数据到
高置信度变异检测：《基因组分析工具包最佳实践流程》，生物信息学当前协议，43:11.10.1-11.10.33，2013。

[Decap15] D. Decap, J. Reumers, C. Herzeel, P. Costanza and J. Fostier, ”Halvade: 可扩展的序列分析使用 MapReduce”, Bioinformatics, btv179v2-btv179, 2015. [Bio] http://www.bioplanet.com/gcat [Gusfield97] D. Gusfield. 1997. 字符串、树和序列上的算法. 剑桥大学出版社, 剑桥, 英国. [Picard] https://broadinstitute.github.io/picard [NA12878] http://allseq.com/knowledge-bank/1000-genome/get-your-1000-genome-test-data-set [1000genomes] ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase3 [Dean08] J. Dean 和 S. Ghemawat, ”MapReduce: Simplified
“在大型集群上的数据处理”，ACM 通信，第 51 卷，第 1 期，2008 年。

[Zaharia10] M. Zaharia 等，“Spark：基于工作集的集群计算”，HotCloud’10，USENIX 协会。[Abuin16] J.M. Abuin, J.C. Pichel, T.F. Pena 和 J. Amigo，
“SparkBWA：加速高通量 DNA 测序数据的比对”，PLoS ONE 第 11 卷第 5 期，e0155461，2016 年。

[Jones12] D.C. Jones, W.L. Ruzzo, X. Peng 和 M.G. Katze,
“通过高效从头组装辅助下一代测序读段压缩”，核酸研究，2012 年。

[Kelly15] B.J. 凯利等，《Churchill：一种超快速、确定性...
一种针对临床和人群规模基因组学中人类遗传变异发现的特性、高度可扩展且平衡的并行化策略”，基因组生物学，第 16 卷，第 6 期，2015 年。

[Li13] H. Li, ”对序列读取、克隆序列和
使用 BWA-MEM 组装 contigs”，arXiv:1303.3997 [qbio.GN], 2013.

[Mushtaq15] H. Mushtaq 和 Z. Al-Ars，“基于集群的 Apache
“使用 Apache Spark 实现的 GATK DNA 分析管道”，
IEEE 国际生物信息学与生物医学会议 (BIBM), pp. 1471-1477, 2015.

[Mushtaq17] H. Mushtaq, 等，《SparkGA：一个 Spark 框架
用于成本效益高、快速准确的规模化 DNA 分析”，ACM 生物信息学、计算生物学与健康信息学会议 (ACM-BCB)，第 148-157 页，2017 年。

[Langmead12] B. Langmead 和 S.L. Salzberg, ”Fast gapped-read
与 Bowtie 2 的比对”，Nature Methods，第 9 卷，第 4 期，第 357-359 页，2012 年。

[Abuin12] J.M. Abun, J.C. Pichel, T.F. Pena 和 J. Amigo,
“BigBWA：将 Burrows-Wheeler 对齐器应用于大数据技术，生物信息学”，第 31 卷，第 24 期，第 4003-4005 页，2015 年。