PLOS ONE

研究文章

SparkBWA:加速对齐高通量DNA测序数据

José M. Abuín1☯*、 Juan C. Pichel1☯、 Tomás F. Pena1☯、 Jorge Amigo2,3☯

1 圣地亚哥大学信息技术研究中心 (CITIUS)
孔波斯特拉,圣地亚哥德孔波斯特拉,西班牙,2 Fundación Pública Galega de Medicina Xenómica (SERGAS),圣地亚哥德孔波斯特拉,西班牙,3 Grupo Medicina Xenómica,研究所
圣地亚哥德孔波斯特拉疗养院, 圣地亚哥德孔波斯特拉, 西班牙

☯ 这些作者对这项工作做出了同等贡献。 * josemanuel.abuin@usc.es

开放获取

引用:Abuín JM, Pichel JC, Pena TF, Amigo J (2016) SparkBWA:加速高通量DNA测序数据的比对。PLoS ONE 11(5): e0155461. doi:10.1371/journal.pone.0155461

编辑:Ruslan Kalendar,赫尔辛基大学,芬兰

收到日期:2016年3月30日

接受日期:2016 年 4 月 30 日

出版日期:2016年5月16日

数据可用性声明:SparkBWA 软件可在 GitHub 存储库https://github.com/citiususc/SparkBWA上获取。输入数据可从千人基因组计划(http://www.1000genomes.org) 获取。

资助:这项工作得到了 Ministryio de Economía y Competitividad (西班牙)的支持(http://www.mineco.gob.es)拨款 TIN2013-41129-P 和 TIN2014-54565-JIN。没有额外的外部
此项研究获得的资金。

竞争利益:作者声明不存在竞争利益。

抽象的

新一代测序 (NGS) 技术催生了海量基因组数据,这些数据需要分析和解读。这对 DNA 序列比对过程产生了巨大的影响,因为如今的比对过程需要将数十亿个小 DNA 序列映射到参考基因组上。因此,序列比对仍然是序列分析工作流程中最耗时的阶段。为了解决这个问题,最先进的比对器采用了并行化策略。然而,现有的解决方案可扩展性有限,且实现复杂。本文介绍了 SparkBWA,这是一种利用 Spark 等大数据技术来提升最广泛使用的比对器之一 Burrows-Wheeler 比对器 (BWA) 性能的新工具。SparkBWA 的设计采用了两个独立的软件层,无需修改原始 BWA 源代码,从而确保其与任何 BWA 版本(无论未来版本或旧版本)兼容。SparkBWA 在不同场景下进行了评估,在性能和可扩展性方面均表现出色。与其他基于BWA的并行比对工具的比较验证了我们方法的优势。最后,我们为NGS专业人员提供了一个直观灵活的API,以促进新工具的接受和采用。本文所述软件的源代码已公开发布,网址为https://github.com/citiususc/SparkBWA。使用 GPL3 许可证。

1 简介

现代DNA测序的历史始于35年前。这些年来,DNA测序能力和速度取得了惊人的增长,尤其是在新一代测序(NGS)和大规模并行测序出现之后。NGS带来了可用的测序数据量空前的爆炸式增长。例如,Illumina HiSeqX™ Ten等新型测序技术每次运行可产生多达60亿个序列读数。将这些数据比对到参考基因组上通常是序列分析工作流程的第一步。这个过程非常耗时,尽管最先进的

尽管比对软件的开发是为了高效处理大量DNA序列,但比对过程仍然是生物信息学分析的瓶颈。此外,NGS平台的快速发展,将测序能力推向了前所未有的水平。

为了应对这一挑战,我们建议利用大数据技术的并行架构来提升序列比对器的性能和可扩展性。这样,我们就能在合理的时间内处理海量测序数据。本文特别考虑使用 Apache Spark [1]作为大数据框架。Spark 是一个集群计算框架,它使用分布式内存抽象(称为弹性分布式数据集 (RDD))以容错方式支持内存和磁盘计算。RDD 可以显式地缓存在集群节点的内存中,并在多个类似 MapReduce 的并行操作中重用。

本文介绍了 SparkBWA,这是一种将 Burrows-Wheeler 比对器 (BWA) [2]集成到 Spark 框架的新工具。BWA 是将序列读取映射到大型参考基因组的最广泛使用的比对工具之一。它包含三种不同的短读取比对算法。SparkBWA 的设计旨在满足三个要求。首先,SparkBWA 在性能和可扩展性方面应优于 BWA 和其他基于 BWA 的比对器。需要注意的是,BWA 有其自己的共享内存系统并行实现。第二个要求与 SparkBWA 保持兼容,使其与 BWA 的未来版本和旧版本兼容。由于 BWA 不断发展以包含新的功能和算法,因此 SparkBWA 必须与 BWA 版本无关。这是与其他基于 BWA 的现有工具的一个重要区别,因为其他现有工具需要修改 BWA 源代码。最后,NGS 专业人员需要一种能够高效执行序列比对的解决方案,并且完全隐藏实现细节。为此,SparkBWA 提供了一个简单灵活的 API 来处理与比对过程相关的所有方面。这样,生物信息学家只需专注于需要处理的科学问题。

本文对 SparkBWA 的性能和内存消耗进行了评估,并全面比较了 SparkBWA 与几种基于 BWA 的先进对齐器。这些工具利用 Pthreads、MPI 和 Hadoop 等不同的并行方法来提升 BWA 的性能。性能结果证明了我们方案的优势。

本研究结构如下:第 2 部分介绍本文背景。第 3 部分
讨论了相关工作。第 4 节详细介绍了 SparkBWA 的设计并介绍了其 API。
第5节介绍了为评估我们提案的行为和性能而进行的实验,并与其他基于BWA的工具进行了比较。最后,第6节阐述了本文得出的主要结论。

2 背景 2.1 MapReduce

编程模型MapReduce [3]是 Google 推出的一种编程

模型,用于在海量计算节点上处理和生成海量数据集。MapReduce 程序执行分为两个阶段:map 和 reduce。在该模型中,MapReduce 计算的输入和输出都是键值对列表。用户只需专注于实现 map 和 reduce 函数。在 map 阶段,map 工作线程以键值对列表作为输入,并生成一组中间输出键值对,这些中间输出键值对存储在中间存储(即文件或内存缓冲区)中。reduce 函数处理每个中间键及其关联的值列表,以生成最终的键值对数据集。通过这种方式,map 工作线程实现数据并行,而 reduce 工作线程执行并行归约。需要注意的是

并行化、资源管理、容错和其他相关问题由 MapReduce 运行时处理。

Apache Hadoop [4]是 MapReduce 编程模型最成功的开源实现。Hadoop 基本上由三层组成:数据存储层（HDFS Hadoop 分布式文件系统[5]）、资源管理器层（YARN 又一个资源协商器[6])和数据处理层（Hadoop MapReduce 框架）。HDFS 是一个面向块的文件系统,其理念是“一次写入,多次读取”是最高效的数据处理模式。因此,Hadoop 在需要单次 MapReduce 执行的高度并行应用程序中（假设 map 和 reduce 阶段之间的中间结果不大)表现出色,甚至对于需要少量顺序 MapReduce 执行的应用程序也表现出色[7]。需要注意的是,Hadoop 还可以高效地处理由一个或多个 map 函数组成的作业,方法是将多个 mapper 链接起来,后面跟着一个 reducer 函数,还可以选择链接零个或多个 map 函数,从而节省 map 阶段之间的磁盘 I/O 成本。对于更复杂的工作流程,应该使用Apache Oozie [8]或 Cascading [9] 等解决方案。

这些工作流管理器的主要缺点是,当必须使用 HDFS 存储中间数据时,性能会有所损失。例如,一个迭代算法可以表示为多个 MapReduce 作业的序列。由于不同的 MapReduce 作业无法直接共享数据,因此必须将中间结果写入磁盘,并在下一次迭代开始时再次从 HDFS 读取,这会导致性能下降。值得注意的是,即使算法的每次迭代也可能包含一个或多个 MapReduce 执行。

在这种情况下,性能方面的下降更加明显。

2.2 Apache Spark

Apache Spark 是一个集群计算框架,旨在克服 Hadoop 的限制,以支持迭代作业和交互式分析,最初由加州大学伯克利分校开发[1],现在由 Apache 软件基金会管理。

Spark 采用主/从架构,包含一个中央协调器（驱动程序)和多个分布式工作器（执行器）。它引入了弹性分布式数据集 (RDD) 的概念[10],以容错方式支持内存和磁盘计算。RDD表示跨集群节点分区的只读对象集合,如果分区丢失,可以重建。用户可以显式地将 RDD 缓存到跨机器的内存中,并在多个类似 MapReduce 的并行操作中重用它。通过使用 RDD,程序员可以对数据执行迭代操作,而无需将中间结果写入磁盘。因此,Spark 非常适合机器学习等算法。

RDD 可以通过分发对象集合（例如列表或集合)或从 Hadoop 支持的任何存储源加载外部数据集来创建,这些存储源包括本地文件系统、HDFS、Cassandra [11]、 HBase [12]、 Parquet [13]等。对于创建的 RDD,Spark 支持两种类型的并行操作:转换和操作。转换是针对 RDD 的操作,会返回新的 RDD,例如 map、filter、join、groupByKey 等。生成的 RDD 默认存储在内存中,但 Spark 也支持在必要时将 RDD 写入磁盘。另一方面,操作是启动计算的操作,将结果返回给驱动程序或将其写入存储。例如 collect、count、take 等。需要注意的是,RDD 上的转换是惰性求值的,这意味着 Spark 在看到操作之前不会开始执行。

从高层次上讲,Spark 应用程序由一个驱动程序组成,该程序包含应用程序的主函数,并在集群上定义 RDD,然后对其应用转换和操作。Spark 程序会根据定义的转换和操作,隐式地创建

RDD,即操作的逻辑有向无环图 (DAG),由驱动程序转换为物理执行计划。然后,该执行计划会进行优化,例如合并多个 Map 转换,并将各个任务打包并准备发送到集群。驱动程序通过 SparkContext 连接到集群。执行器或工作进程负责在集群的每个节点上高效运行任务。

Apache Spark 提供了 Python 和 Scala 交互式 shell,允许用户与分布在多台机器的磁盘或内存中的数据进行交互。除了交互式运行之外,Spark 还可以链接到 Java、Scala 或 Python 应用程序中。最后,我们必须强调的是,Spark 可以在本地模式下运行,也可以在集群上以独立模式运行,或者使用 Mesos [14]或 YARN [6] 等集群管理器运行。

2.3 Burrows-Wheeler 矫正器（BWA）

Burrows-Wheeler 比对软件 (BWA) 是一款非常流行的开源软件,用于将序列读取映射到大型参考基因组。具体来说,它包含三种不同的算法:BWA-backtrack [2]、BWA-SW [15]和 BWA-MEM [16]。第一种算法专为 Illumina 短序列读取而设计,最长可达 100bp（碱基对）,而其他算法则侧重于较长的读取。

最新的 BWA-MEM 比 BWA-SW 更适合 70bp 或更长的读段,因为它速度更快、更准确。此外,在映射 100bp 或更长的读段时,BWA-MEM 也表现出比其他几款先进的读段比对工具更优的性能。

正如我们之前提到的,序列比对是一个非常耗时的过程。因此,BWA 有自己的并行实现,但它仅支持共享内存机器。因此,可扩展性受限于单个计算节点上可用的线程（核心）数量和内存。

虽然 BWA 可以读取未比对的 BAM [17]文件,但它通常接受 FASTQ 格式[18]作为输入,这是原始序列读取最常见的输出格式之一。它是一种纯文本格式,每四行描述一个序列或读取。图1显示了包含两个读取的示例。每个读取提供的信息包括:标识符（第一行）、序列（第二行)和读取的质量得分（第四行）。一个额外的字段,用符号“+”表示,用作数据和质量信息（第三行)之间的分隔符。BWA 能够使用单端读取（一个输入 FASTQ 文件)和双端读取（两个输入 FASTQ 文件）。当考虑双端读取时,可以获得对应于同一 DNA 片段两端的两个序列。这两个读取包含在不同的输入文件中,使用相同的标识符,并且在文件中的相对位置相同。这样,考虑到我们的例子,序列#2对应的对将位于另一个输入文件的第5行。另一方面,BWA的输出是一个SAM（序列比对/图谱） [17]文件,这是存储与参考序列比对结果的标准格式。例如,在进行变异发现分析时,这个SAM文件将进一步发挥作用。

```
1  @ERR000589.41 EAS139_45:5:1:2:111/1
2  CTTTCCTCCCTGCTTTCCTGGCCCCACCATTTCCAGGGAACATCTTGTCAT
3  +
4  3IIIIIIIIIIIII>1IIIFF9BG08E00I%IG+&?(4)%00646.C1#&(
5  @ERR000589.42 EAS139_45:5:1:2:1293/1
6  AGTTGTTAAAATCCAAGCCAATTAAGATAGTCTTATCTTTTTAAAAGAAAT
7  +
8  IIIIIGII.AIIII=?I9G-/II=+I=4?761BA2C9I+5A711+&>1$/I
```

图 1.FASTQ 文件格式示例。

doi:10.1371/journal.pone.0155461.g001

3 相关工作

我们在文献中找到了几种基于 Burrows-Wheeler 比对器的有趣工具,它们利用并行和分布式架构来提升 BWA 性能。其中一些研究专注于 SparkBWA 等大数据技术,但它们都基于 Hadoop。例如 BigBWA [19]、 Halvade [20]和 SEAL [21]。BigBWA是作者最近开发的一款序列比对工具,与其他基于 BWA 的方法相比,它展现出了良好的性能和可扩展性。它的主要优势在于无需修改原始 BWA 源代码。SparkBWA 也具备这一特性,因此这两种工具都能与未来和旧版 BWA 版本兼容。

SEAL 使用 Pydoop [22],这是 MapReduce 编程模型的 Python 实现,运行在 Hadoop 之上。它允许用户用 Python 编写程序,并通过包装器调用 BWA 方法。SEAL 仅适用于 BWA 的特定修改版本。

由于 SEAL 基于 BWA 0.5 版本,因此它不支持用于较长读取的新 BWA-MEM 算法。

Halvade 也基于 Hadoop。它包含一个变异检测阶段,该阶段是 DNA 测序工作流程中序列比对之后的下一个阶段。Halvade 将 BWA 作为外部进程从映射器调用,如果任务超时参数配置不当,可能会导致 Hadoop 执行过程中出现超时。因此,需要事先了解应用程序的执行时间。需要注意的是,将超时参数设置为过高的值会导致实际超时检测出现问题,从而降低 Hadoop 容错机制的效率。为了解决这个问题,正如后续章节所述,SparkBWA 使用 Java 原生接口 (JNI) 来调用 BWA 方法。

另一种方法是将标准并行编程范式应用于 BWA。例如,pBWA [23]使用 MPI 将 BWA 并行化,以便在集群上执行比对。必须强调的是,与 SparkBWA 相比,pBWA 缺乏容错机制。此外,pBWA 和 SEAL 都不支持 BWA-MEM 算法。

有多种解决方案试图利用 GPU 的计算能力来提升 BWA 的性能。BarraCUDA [24] 就是一个例子,它基于 CUDA 编程模型。它需要修改 BWA 的 BWT（Burrows Wheeler 变换)比对核心,以利用 GPU 的大规模并行性。与支持 BWA 中所有算法的 SparkBWA 不同,BarraCUDA 仅支持用于短读取的 BWA-backtrack 算法。与线程版本的 BWA 相比,它的性能提升高达 2 倍。值得一提的是,由于最新版本的 BWA 中 BWT 数据结构的一些变化,BarraCUDA 仅兼容使用 BWA 0.5.x 版本生成的 BWT。其他重要的序列比对工具（非基于 BWA)也利用 GPU,包括 CUSHAW [25]、 SOAP3 [26]和 SOAP3-dp [27]。

一些研究人员致力于利用新的英特尔至强融核协处理器（英特尔集成众核架构,MIC)来加速比对过程。例如,基于BWA的mBWA [28]为至强融核协处理器实现了BWA回溯算法。mBWA允许同时使用主机CPU和协处理器来执行比对,其速度比BWA提高了5倍。 [29]中介绍了另一种MIC协处理器的解决方案。第三个利用MIC架构的比对器是MICA [30]。作者声称它比使用6核的线程化BWA快5倍。

请注意,与 SparkBWA 不同,此工具不是基于 BWA。

另一位研究人员利用 FPGA（现场可编程门阵列)中的细粒度并行性来提高几个短读比对器的性能,其中包括一些基于 BWT 的比对器[31–33]。

最后,最近的一项研究使用 Spark 来提高最著名的比对算法之一 Smith-Waterman 算法[34] 的性能。性能结果证明了 Spark 作为此类应用框架的潜力。

4 SparkBWA

本节介绍一个名为 SparkBWA 的新工具,它将 Burrows-Wheeler 对齐器集成到 Spark 框架中。如简介中所述,SparkBWA 的设计基于以下三个目标:

- 它应该在性能和
  可扩展性。
- 它应该与 BWA 版本无关,以确保其与未来或旧版 BWA 版本的兼容性。
- 应向 NGS 专业人员提供直观、灵活的 API,以方便
  表明新工具的接受度和采用度。

接下来,详细描述了SparkBWA的设计和实现,以及高层API的规范。

4.1 系统设计SparkBWA

工作流程包含三个主要阶段:RDD 创建、map 和 reduce 阶段。在第一个阶段,输入数据被准备好输入到 map 阶段,严格来说,比对过程在此进行。具体来说,RDD 是根据 FASTQ 输入文件创建的,这些文件存储在 HDFS 中。需要注意的是,在本文中,我们假设 HDFS 是分布式文件系统。这样,数据分布在各个计算节点上,以便在 map 阶段进行并行处理。FASTQ 文件格式中的读取标识符用作 RDD 中的键(参见图1 中的示例)。这样,从输入文件生成的键值对的形式如下:<read_id, read_content>,其中 read_content 包含 read_id 对应的序列的所有信息。这些 RDD 将在之后的 map 阶段使用。这种方法在单端读取(即只有一个 FASTQ 输入文件)时非常有效。

然而,SparkBWA 也应该支持双端读取。在这种情况下,将创建两个 RDD,每个输入文件一个,并分布在节点上。Spark 以这样一种方式分布 RDD,即不能保证两个 RDD 的第 i 个数据分割(分区)由同一个 Mapper 处理。这样,Mapper 就无法处理双端读取,因为它们始终位于两个 RDD 的同一个第 i 个数据分区中。这种行为可以在图 2(a) 所示示例的 RDD 创建阶段观察到。为了解决这个问题,提出了两种解决方案:

- 连接:此方法基于 Spark 连接操作,该操作是一种转换操作,通过将具有相同键的元素分组来将两个 RDD 合并在一起。该解决方案如图 2(a) 所示。由于两个输入文件中成对读取的键相同,因此连接操作后的结果将是一个唯一的 RDD,格式为:<read_id, Tuple<read_content1, read_content2>> (示例中为 RDDUNSORTED)。连接操作后生成的 RDD 不会保留 FASTQ 文件中读取的先前顺序。但这不会造成问题,因为映射器将彼此独立地处理成对读取。

  然而,Spark 提供了 sortByKey 转换,可以根据 RDD 记录的键对其进行排序。在示例中,应用此操作后创建的新 RDD 为RDDSORTED。必须强调的是,sortByKey 操作在内存消耗方面比较昂贵。

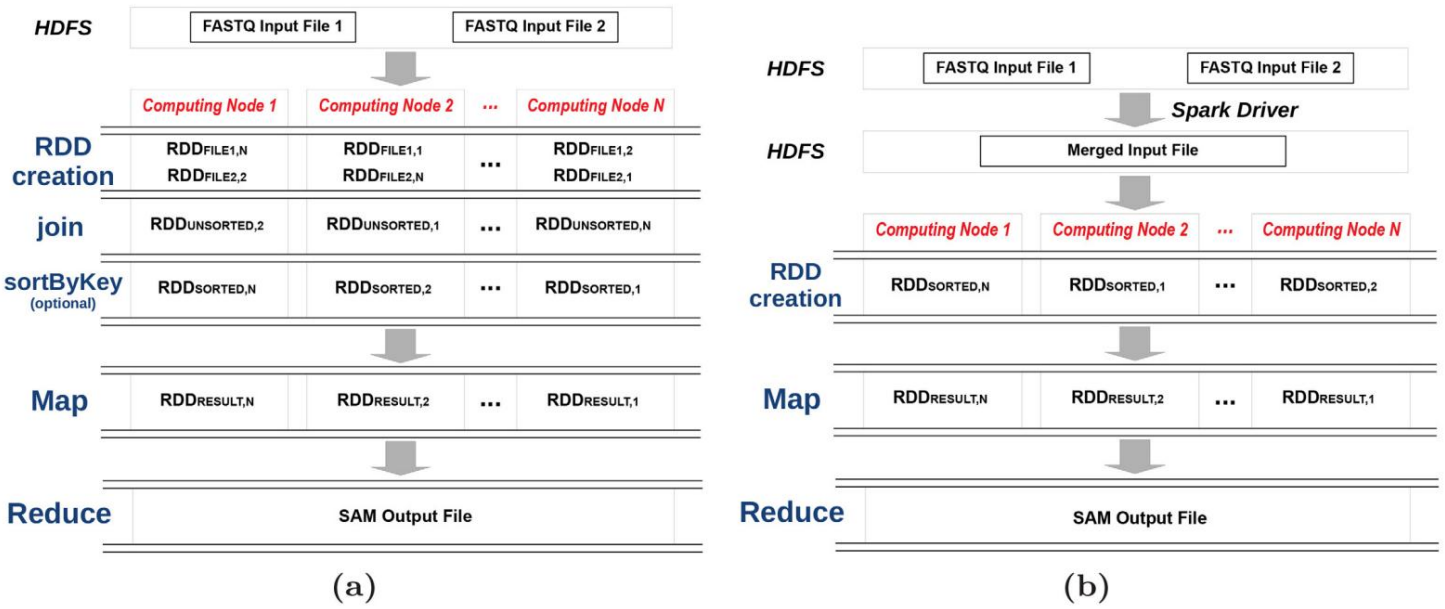

图 2. 使用 (a) Join 和 (b) SortHDFS 方法进行双端读取的 SparkBWA 工作流程。

doi:10.1371/journal.pone.0155461.g002

因此,此步骤在 SparkBWA 数据流中是可选的,如果用户想要获得排序的输出,则应专门启用它。

· SortHDFS:提出了一种新方法,以避免 join 和 sortByKey 操作
（见图 2(b)）。该解决方案可以被视为一个预处理阶段,需要读写 HDFS。这样,FASTQ 输入文件可以
通过以下方式直接访问:
Spark 驱动程序中的 HDFS Hadoop 库。双端读取（即那些
两个文件中具有相同标识符的文件)将合并为一个新的 HDFS 文件中的一条记录。
BWA 需要区分对中的两个序列,使用分隔符字符串来
方便 Mapper 中的后续解析过程。之后,创建一个 RDD
新文件（图中的 RDDSORTED ）。这样,键值对的格式为 <read_id, merged_content>。此解决方
案执行了几个耗时的 I/O 操作,但与 join & sortByKey 方法相比,节省了大量内存,因为我们

在第 5 节中进行说明。

一旦 RDD 可用,map 阶段就开始了。mapper 将应用序列比对
在 RDD 上调用 BWA 算法。然而,从 Spark 调用 BWA 并不简单
因为 BWA 源代码是用 C 语言编写的,而 Spark 只允许在 Scala、Java 中运行代码
或 Python。为了解决这个问题,SparkBWA 利用了 Java 原生接口
（JNI）,它允许合并用 C 和 C++ 等语言编写的本机代码
作为 Java 代码。

Map 阶段由两个独立的软件层设计。第一个软件层对应于 BWA 软件包,另一个软件层负责处理
RDD,将
将数据输入到 BWA 层,并从地图工作者那里收集部分结果。我们必须
强调映射器仅通过 JNI 调用 BWA 主函数。这
设计避免了对原始 BWA 源代码的任何修改,从而确保了 SparkBWA 与未来或旧版 BWA 版本的兼容性。
这样,我们的工具就与版本无关了。
关于BWA。请注意,此方法与BigBWA工具[19]中采用的方法类似。

双层设计的另一个优点是,对齐过程可以使用两个并行级别来执行。第一级对应于分布在集群中的 map 进程。在第二级中,每个单独的 map 进程都使用

多个线程,利用 BWA 并行实现共享内存
机器。我们将这种操作模式称为混合模式。此模式可以通过
用户通过 SparkBWA API。

另一方面,BWA 除了使用 FASTQ 文件之外,还使用参考基因组作为输入。所有映射器都需要完整的参考基因组,因此必须使用 NFS 在所有计算节点之间共享它,或者将其存储在所有节点的同一位置(例如,使用 Spark
广播变量)。

一旦映射阶段完成,SparkBWA 会在 HDFS 中为每个
启动 map 进程。最后,用户可以将所有输出合并到一个文件中,并选择执行额外的 reduce 阶段。

4.2 SparkBWA API

SparkBWA 的要求之一是为生物信息学家提供一个简单而强大的
使用 Apache Spark 等大数据技术进行序列比对的方法。通过这种方式
目标明确,提供了一个基本的API。它允许NGS专业人员只专注于科学
问题,而 SparkBWA 的设计和实现细节对于
他们。

SparkBWA 可以通过 Spark shell(Scala)或控制台使用。表 1总结了 API
在 Shell 中设置 SparkBWA 选项的方法及其相应的控制台
参数。例如,可以选择数据分区的数量、RDD 的
创建的线程数,或每个映射器使用的线程数(混合模式)。

1. Spark Shell:Spark 附带一个交互式 shell,提供了一种简单的方式来学习
   Spark API,以及一个强大的交互式数据分析工具。它提供以下两种版本
   Scala(在 Java VM 上运行,因此是使用现有 Java 库的好方法)或
   Python。当前 SparkBWA 版本仅支持 Scala shell。
   以下是如何使用 Spark shell 中的 SparkBWA 执行比对的示例

表 1. 用于设置 SparkBWA 选项的 API 方法和控制台参数。

功能	默认	控制台参数	描述
设置使用Reducer(布尔值)	错误的	-r	使用 reducer 生成一个输出 SAM 文件。
设置分区号(整数)	汽车	无 |-分区 <数量>	默认情况下,数据会被分割成 HDFS 块大小的块。否则,输入数据分成 num 个分区。
设置快速排序(整数)	加入	无 |-排序 |-sorthdfs	设置双端读取的 RDD 创建方法:Join (0)、Join 和 sortByKey (1)或 SortHDFS (2)。
设置线程数(int)	1	-threads <数量>	如果 num > 1,则启用混合并行模式,这样每个 map 使用 num 个线程执行该进程。
设置算法(int)	BWA-MEM -mem |-aln |-bwasw -配对 |-单个		设置比对算法:BWA-MEM(0)、BWA-backtrack(1)、BWA-SW(2)
设置配对读取(布尔值)	配对		使用单端(一个 FASTQ 输入文件)或双端读取(两个 FASTQ 输入文件)。
设置索引路径(字符串)	–	-index <前缀>	设置参考基因组的路径(强制选项)。
设置输入路径(字符串)	–	位置	设置 FASTQ 输入文件的路径(在 HDFS 中)(单端和双端读取)。
设置输入路径2(字符串)	–	位置	设置第二个 FASTQ 输入文件的路径(在 HDFS 中)(强制选项双端读取)。
设置输出路径(字符串)	–	位置	设置输出 SAM 文件的存储位置(在 HDFS 中)。

doi:10.1371/journal.pone.0155461.t001

```
scala> var options = new BwaOptions();

scala> options.setInputPath("ERR000589_1.filt.fastq");
scala> options.setInputPath2("ERR000589_2.filt.fastq");

scala> options.setOutputPath("OutputSparkBWA.sam");
scala> options.setIndexPath("/opt/HumanBase/hg38");

scala> var newBwa = new BwaInterpreter(options, sc);
scala> var bwaRDD = newBwa.getDataRDD(); # Optional
scala> newBwa.runAlignment();
```

图 3. 从 Spark Shell（Scala)运行 SparkBWA 的示例。

doi:10.1371/journal.pone.0155461.g003

如图3所示。首先,用户应创建一个 BwaOptions 对象来指定执行 SparkBWA 所需的选项（第 1 行）。本例中仅设置了必需的选项（第 3-7 行）。其他选项请参见表1 。

指定选项后,应创建一个新的 BwaInterpreter（第 9 行）。此时,将根据前面 4.1 节中详述的实现,从输入文件创建 RDD。值得一提的是,RDD 的创建是惰性求值的,这意味着 Spark 在调用某个操作之前不会开始执行。例如,此操作可以是使用 getDataRDD 方法显式获取输入 RDD（第 10 行）。此方法非常有用,因为它允许用户将 Spark API 提供的所有转换和操作（以及用户定义的函数)应用于输入 RDD。需要注意的是,使用 getDataRDD 方法并非使用 SparkBWA 执行序列比对的必要条件。另一个触发 RDD 创建的操作是 runAlignment,它将执行完整的 SparkBWA 工作流,包括 map 和 reduce 阶段（第 11 行）。

2. 控制台:也可以从控制台运行 SparkBWA,即使用 spark-
提交命令。示例如图4所示。spark -submit 提供了多种选项,允许用户控制应用程序特定运行的具体细节（第 2-6 行）。在本例中,用户还需要将 SparkBWA 选项作为参数传递给 Spark（第 7-11 行）。表1详细列出了 SparkBWA 支持的所有标志。

```
spark-submit
 --class SparkBWA
 --master yarn-client          # Connect to a YARN cluster
 --num-executors 16            # Number of worker processes
 --archives bwa.zip            # BWA library
 SparkBWA.jar                  # SparkBWA tool
 -partitions 16                # Data partitions
 -index /opt/HumanBase/hg38    # Reference genome
 ERR000589_1.filt.fastq        # Input file 1
 ERR000589_2.filt.fastq        # Input file 2
 OutputSparkBWA.sam            # Output file
```

图 4.从控制台运行 SparkBWA 的示例。

doi:10.1371/journal.pone.0155461.g004

因此,SparkBWA 提供了一个简单灵活的界面,用户可以
只需在 Spark shell 中编写几行代码即可执行序列比对,或者使用
控制台中的标准 spark-submit 工具。

5 评估

本节将从性能、可扩展性和内存消耗等方面对 SparkBWA 进行评估。首先,提供实验设置的完整描述。接下来,
本文详细分析了 SparkBWA,特别关注 RDD 的创建及其不同的操作模式(常规和混合)。最后,为了验证我们的方案,我们还与几种基于 BWA 的对齐器进行了比较。

5.1 实验设置

SparkBWA 使用千人基因组计划[35] 的数据进行测试。输入数据集的主要特征如表2 所示。读取次数是指

与参考基因组比对的序列。读取长度用
碱基对的数量(bp)。

由于比对可以针对单端或双端读取进行,因此需要确定在评估过程中使用哪一种。由于双端DNA测序读取在包含重复序列的DNA区域之间提供了更优的比对结果

读取,这是本文所考虑的。这样,每个数据集由两个
FASTQ 文件。

实验在六节点集群上进行。每个节点由四个 AMD
Opteron 6262HE 处理器(4×16 核),配备 256 GiB 内存(即每个核 4 GiB)。
节点通过 10GbE 网络连接。使用的 Hadoop 和 Spark 版本为 2.7.1
和 1.5.2,运行在 CentOS 6.7 平台上。OpenMPI 4.4.7 用于
需要 MPI 的实验。集群配置分配了大约 11 GiB 的内存
每个 YARN 容器(map 和 reduce 进程)最多可以同时执行 22 个容器。这种内存配置允许每个 SparkBWA

容器执行一个 BWA 进程,包括存储引用所需的内存
基因组索引。需要注意的是,集群中的主节点也用作计算节点。

SparkBWA 的行为与几种最先进的基于 BWA 的对齐器进行了比较。在
具体来说,我们考虑了表3中详述的工具。这些工具的简要描述如下:
在第 3 节中提供。pBWA 和 SEAL 仅支持 BWA-backtrack 算法,因为
两者都基于 BWA 0.5 版(2009)。为了公平起见,SparkBWA
同样使用 BWA 版本 0.5 获得了 BWA-backtrack 算法的性能结果。
在 BWA-MEM 的情况下,评估了三种不同的对齐器:BigBWA、Halvade 和 BWA
(共享内存线程版本)。对于 BWA-MEM 性能评估,最新的
使用本文撰写时可用的 BWA 版本(版本 0.7.12,12 月
2014)。我们必须强调,本节中显示的所有时间结果均按以下方式计算:
二十次处决的平均值(算术平均值)。

表 2. 千人基因组计划输入数据集的主要特征。

标签	姓名	读取次数	读取长度(bp)	大小(GiB)
D1	NA12750/ERR000589	12×106	51	3.4
D2	HG00096/SRR062634	24.1×106	100	11.8
D3	150140/SRR642648	98.8×106	100	48.3

doi:10.1371/journal.pone.0155461.t002

表 3. 评估的算法和基于 BWA 的对齐器。

算法	工具	并行化技术
BWA回溯	pBWA [23]	MPI
	密封[21]	Hadoop
	SparkBWA	火花
BWA-MEM	布瓦[16]	P线程
	BigBWA [19]	Hadoop
	哈尔瓦德[20]	Hadoop
	SparkBWA	火花

doi:10.1371/journal.pone.0155461.t003

5.2 绩效评估

5.2.1 RDD 创建。SparkBWA 工作流程的第一阶段是创建
RDD,其中可能包含排序阶段（参见 4.1 节）。我们考虑了两种不同的方法来实现此阶段:Join 和
SortHDFS。第一种方法基于 Spark join
操作,并包括一个额外的可选步骤,按键对输入的双端读取进行排序
（sortByKey 操作）。后一种方法需要从 HDFS 进行读写操作。正如我们
前面指出,该解决方案可以被视为预处理阶段。两种解决方案
已针对不同数据集的开销进行了评估。结果如图5所示。

Join 方法的性能（有和没有 sortByKey 转换）
取决于 map 进程的数量,因此该操作使用 32 和 128 进行评估
映射器。随着映射器数量的增加,排序时间会有所改善,因为

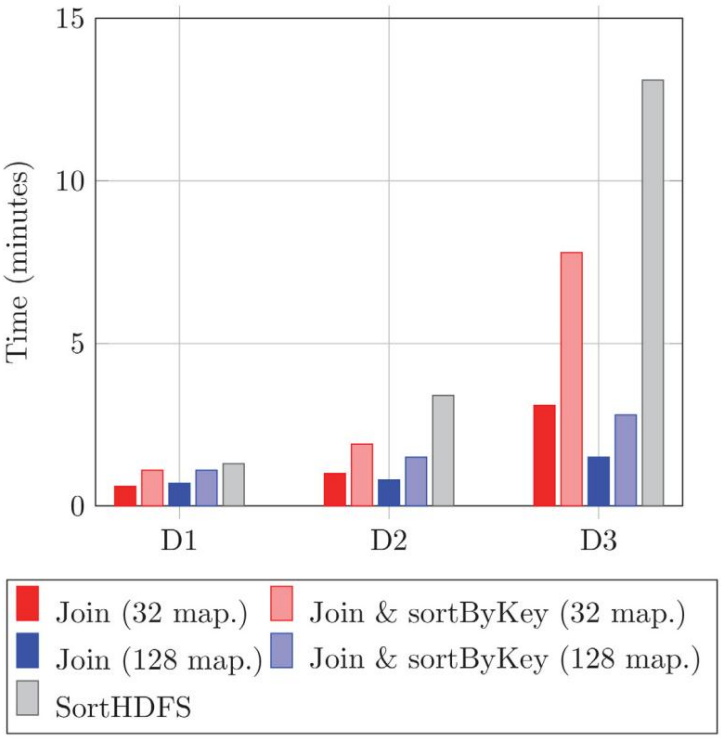

图 5.考虑不同数据集的 RDD 排序操作的开销。

doi:10.1371/journal.pone.0155461.g005

每个工作器计算的数据分割更小。所有数据集都观察到了这种现象,尤其是在考虑 D3 时。

正如预期的那样,所有方法的开销都会随着数据集的大小而增加。
然而,SortHDFS 的增量率更高。例如,对 D3 进行排序比对 D1 排序慢 10 倍,而使用 sortByKey 和不使用 sortByKey 的 Join 方法最多分别慢 5 倍和 7 倍。需要注意的是,D3 比 D1 大 14 倍以上(参见表 2)。

就开销而言,Join 方法总是更优,尤其是在 Map 进程数量增加的情况下。例如,使用 128 个 Mapper(仅执行 Join 操作)对 D3 进行排序仅需 1.5 分钟,这意味着相对于 SortHDFS,速度提高了 8.7 倍。此外,还可以观察到,按键对 RDD 进行排序会消耗额外的时间。具体而言,当仅执行 Join 转换时,开销意味着排序过程所需的时间平均会增加一倍。

另一方面,速度并不是唯一需要考虑的参数
对 RDD 进行排序。通过这种方式,我们也分析了内存消耗。为了说明两种排序方法的行为,我们以 D3 为数据集。
图 6显示了 map 进程在排序操作期间使用的内存。

根据结果,Join 方法总是比 SortHDFS 消耗更多的内存。
这是由 RDD 上的 Spark 操作 join 和 sortByKey 引起的,这两项操作都是内存转换。当 RDD 元素按键排序时,相对于仅应用 join 操作,观察到的差异尤其重要。这样,sortByKey 操作会为每个该数据集的映射器额外消耗大约 3 GiB,这意味着在此阶段 SparkBWA 所需的内存增加了 30% 以上。请注意,当考虑 32 个工作程序时,每个容器可用的内存已达到最大值。128 个工作程序使用的内存较低,因为与考虑 32 个工作程序相比,RDD 被拆分成更小的部分。另一方面,SortHDFS 最多需要 4 GiB 进行预处理

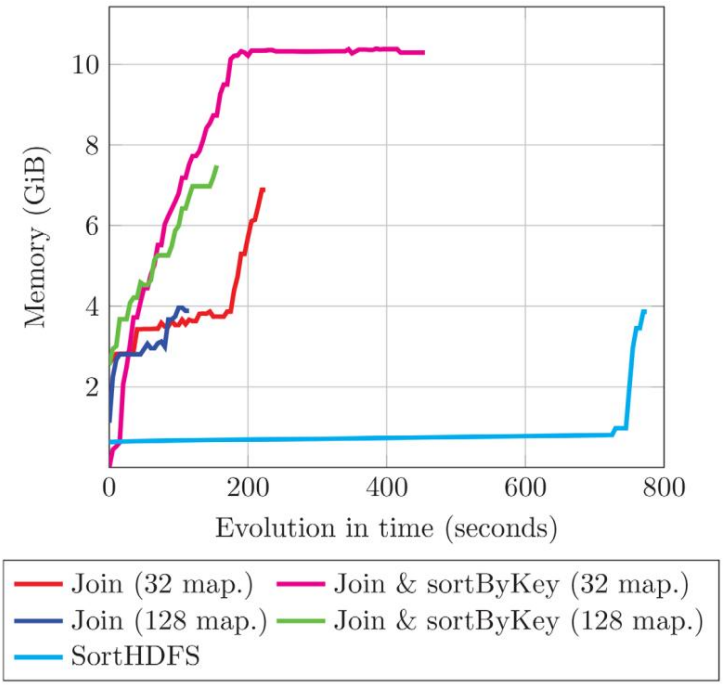

图 6. 考虑数据集 D3 时 SparkBWA 在 RDD 排序操作期间消耗的内存。

doi:10.1371/journal.pone.0155461.g006

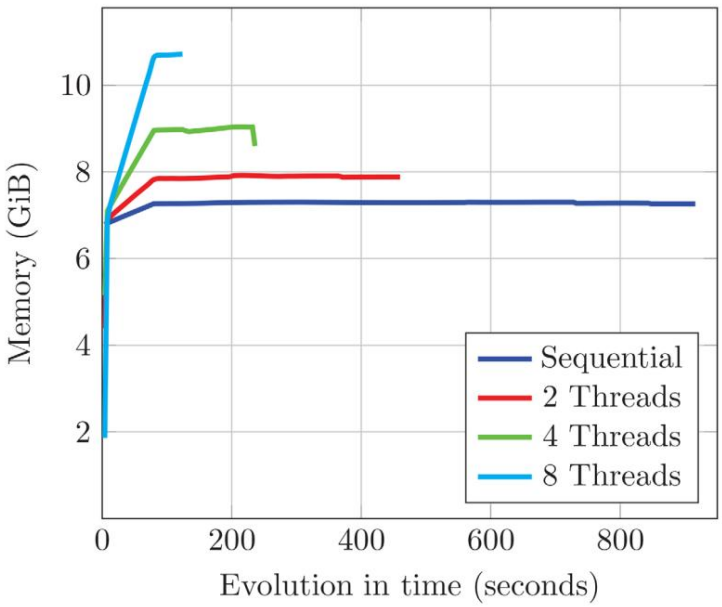

图 7. 使用不同线程执行 BWA-MEM 算法的工作进程所消耗的内存。

doi:10.1371/journal.pone.0155461.g007

示例中的数据集。因此,如果内存资源有限或不足以执行 Join 操作（无论是否使用 sortByKey）,SortHDFS 是最佳选择。请注意,图 6中所示的整体行为与其他数据集的观察结果一致。

5.2.2 混合模式。如第 4.1 节所述,SparkBWA 的设计分为两种软件层:
ers 允许每个工作器使用多个线程,以便利用两级并行性来执行对齐过程。因此,SparkBWA 有两种运行模式:常规模式和混合模式。混合模式是指每个映射进程使用多个线程,而常规模式则按顺序执行每个映射器。

启用混合模式时,每个映射器使用的内存会随着映射器数量而增加
参与计算的线程数。但是,由于 BWA 所需的索引参考基因组在线程之间共享,因此这种增加是适度的。图 7 说明了这种行为,其中使用不同数量的线程执行 BWA-MEM,并以 D1 的小分片作为输入。可以观察到,在常规模式下和 8 个线程的混合模式下,一个 SparkBWA map-per 使用的内存差异仅为 4 GiB。这意味着总内存消耗增加了约 30%,而每个 mapper 的线程数增加了 8 倍。

因此,考虑到我们的实验平台允许每个节点运行 22 个容器,最大内存为 11 GiB,在本例中,混合模式下的 SparkBWA 可以使用节点中的全部 64 个核心,例如运行 16 个 Mapper,每个 Mapper 运行 4 个线程。这与常规模式不同,常规模式最多只能使用节点的 22 个核心。因此,混合模式在计算节点包含大量核心但由于内存限制而只能使用其中少数核心的场景中非常有用。

接下来,我们将评估 SparkBWA 在两种操作模式下的性能。实验采用 BWA-MEM 算法,并在启用混合模式的情况下,分别考虑每个 map 进程使用 2 个和 4 个线程。图 8展示了所有数据集和使用不同数量 mapper 的性能结果。对于 128 个 mapper 和 4 个线程/mapper 的情况,没有结果,因为这意味着最佳执行需要 512 个核心,而我们的集群仅包含 384 个核心。

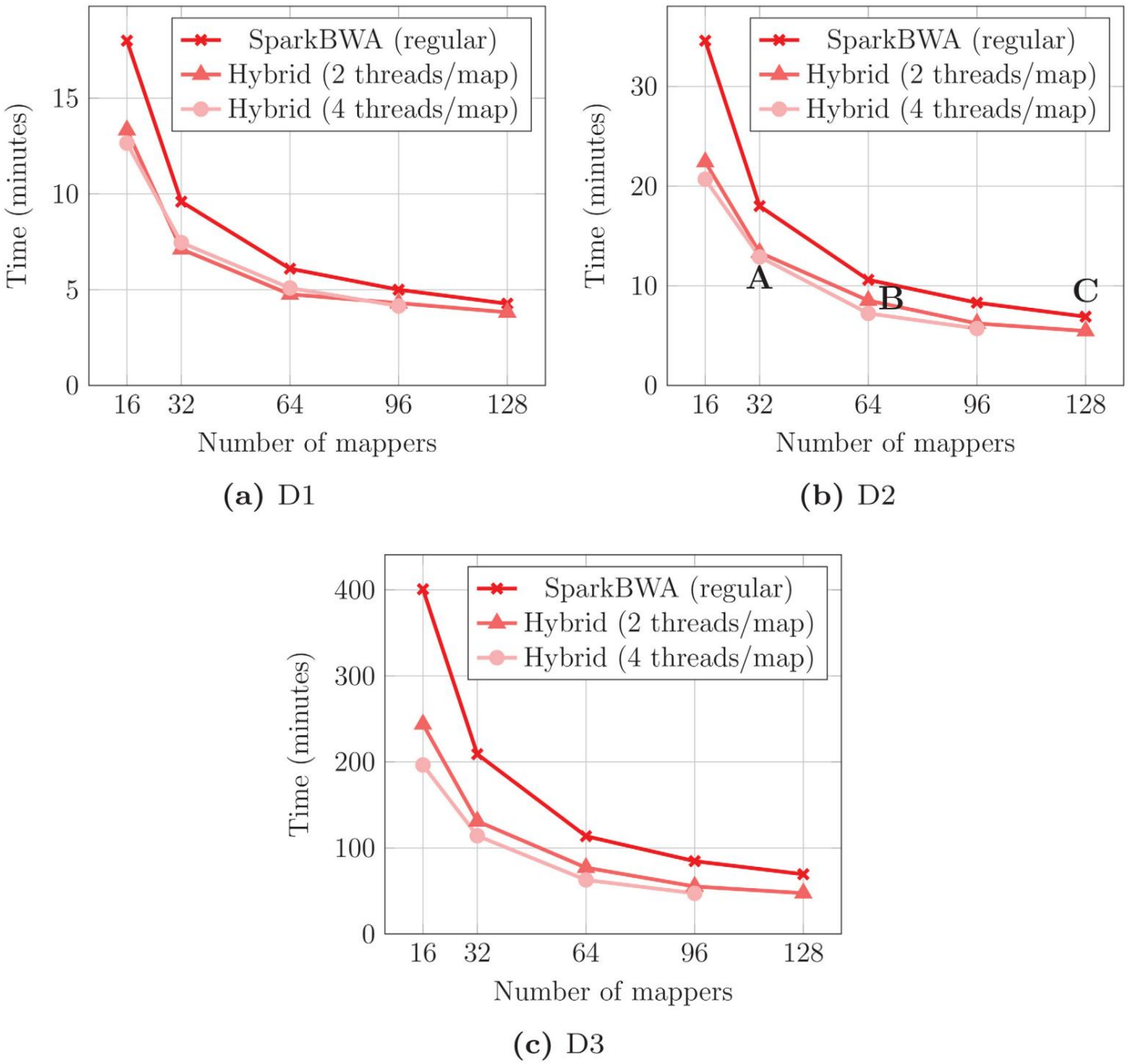

图 8. SparkBWA 使用 BWA-MEM 算法的常规和混合操作模式获得的执行时间。

doi:10.1371/journal.pone.0155461.g008

从性能结果中可以得出几个结论。SparkBWA 在 Mapper 数量方面表现出良好的可扩展性，尤其是在常规模式下（即每个 Mapper 顺序计算）。假设 Mapper 数量相同，混合模式下每个 Mapper 增加线程数仅对最大数据集 (D3) 有利。这表明，在计算中使用更多线程的好处并不能弥补线程同步带来的开销。

表 4.SparkBWA 性能结果摘要。

数据集	操作模式	测绘员数量	时间（分钟）	对对齐/秒	加速
D1	常规混合	128	4.3	46,512	60×
	动力车 (2 th/map) 混	128	3.8	52,632	67.9×
	合动力车 (4 th/map)	96	4.1	48,780	62.9×
D2	常规混合	128	6.9	58,213	71.9×
	动力车 (2 th/map) 混	128	5.5	73,030	90.2×
	合动力车 (4 th/map)	96	5.7	70,468	87.0×
D3	常规混合	128	69.4	23,727	85.6×
	动力车 (2 th/map) 混	128	47.5	34,667	125.0×
	合动力车 (4 th/map)	96	47.3	34,813	126.2×

doi:10.1371/journal.pone.0155461.t004

另一方面,考虑到计算中使用的核心（#threads × #mappers
核心）,我们可以观察到常规模式的性能优于混合模式。例如,
图 8(b)中的 A、B 和 C 点是使用相同数量的核心获得的。常规模式下的 SparkBWA（点 C)明显优于混合
版本。这种行为在大多数
的情况。这样,正如我们之前指出的,SparkBWA 混合模式应该
仅在内存限制不允许使用所有的情况下才是首选选项
每个节点中的核心。

表 4总结了 SparkBWA 在所有数据集上的性能结果。
显示了 SparkBWA 在我们的硬件上执行对齐所需的最短时间
平台、使用的映射器数量、以每对比对的数量来衡量的速度
秒,并且相对于 BWA 的顺序执行也具有相应的加速。
D1、D2 和 D3 的连续时间分别为 258、496 和 5,940 分钟。
对于 D3 的特殊情况,这意味着需要超过 4 天的计算时间。值得注意的是,使用
SparkBWA 这次的时间减少到不到一小时,加速比超过 125 倍。

最后,我们通过将 SparkBWA 的输出与 BWA（顺序版本)生成的输出进行比较,验证了 SparkBWA
在常规和混合模式下的正确性。我们仅发现一些唯一映射的 reads（即具有

质量大于零）。因此,所有情况下的映射坐标都是相同的
已考虑。差异影响唯一映射读取总数的 0.06% 到 1%。
由于质量计算取决于
插入大小统计信息,根据输入流的样本窗口计算得出
序列。这些样本窗口对于 BWA（顺序)中的每个读取都是不同的,并且任何
其他并行实现将输入分成几部分（SEAL、pBWA、Halvade、
BWA 线程版本、SparkBWA 等）。这样,任何基于 BWA 的并行对齐器都将
获得与 BWA 顺序版本略有不同的映射质量分数。
例如,SEAL 报告称,唯一映射读数中平均存在 0.5% 的差异[21]。

5.2.3 与其他对齐器比较。接下来,对不同对齐器进行性能比较。
基于 BWA 的对齐器和 SparkBWA 如下。评估的工具列于表 3。
以及相应的并行化技术。其中一些利用了
经典并行范式,如 Pthreads 或 MPI,而其他则基于大数据技术,如 Hadoop。所有实验均使用 SparkBWA
以常规模式进行。对于
出于比较目的,本小节中的所有图表都包括考虑相对于 BWA 顺序执行的理想加速的相应结果。

已经考虑了两种不同的双端读取算法:BWA-backtrack 和
BWA-MEM。BWA-backtrack 算法的评估是使用

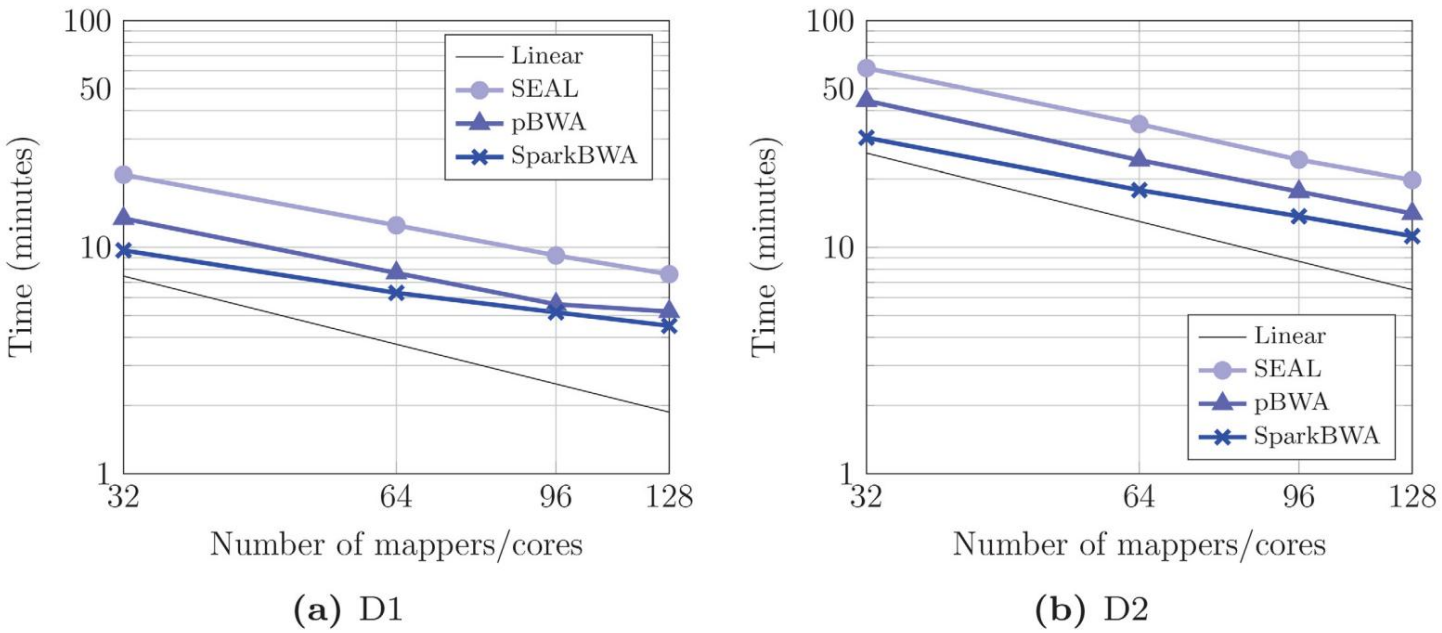

图 9. 考虑运行 BWA-backtrack 算法的几个基于 BWA 的对齐器的执行时间（轴为对数尺度）。

doi:10.1371/journal.pone.0155461.g009

以下比对软件:pBWA、SEAL 和 SparkBWA。当使用成对读段作为输入数据时,BWA-backtrack 包含三个阶段。首先,必须对其中一个输入 FASTQ 文件进行序列比对。然后,对另一个输入文件执行相同的操作。

最后,使用前几个阶段的结果转换为 SAM 输出格式。SparkBWA 和 SEAL 负责整个工作流程,使其对用户完全透明。需要注意的是,SEAL 需要一个预处理阶段来准备输入文件,因此这部分额外时间已计入测量值。另一方面,尽管 BWA-回溯算法是并行执行的,但 pBWA 需要独立执行每个阶段。因此,pBWA 时间计算为每个阶段时间的总和。pBWA 不执行任何预处理。

由于 BWA-backtrack 是专门为较短的读取（<100 bp)设计的,我们考虑 D1 作为输入数据集,但为了完整性起见,D2 也包含在比较中。图 9显示了使用不同数量映射器的对齐时间。在这种情况下,每个映射过程使用一个核心,因此映射器和核心这两个术语是等效的。结果表明,在所有情况下,SparkBWA 的性能都明显优于 SEAL 和 pBWA。正如我们之前提到的,SEAL 时间包括预处理阶段造成的开销,D1 和 D2 的平均预处理时间分别约为 1.9 分钟和 2.9 分钟。这种开销对性能有很大影响,尤其是对于最小的数据集。

图 10显示了 BWA-backtrack 对齐器获得的相应加速比。我们使用了 BWA 顺序时间作为参考。结果证实了 SparkBWA 相对于 SEAL 和 pBWA 的良好表现。例如,SparkBWA 对 D1 和 D2 分别实现了高达 57 倍和 77 倍的加速比。SEAL 实现的最大加速比仅为 31 倍和 42 倍左右,而 pBWA 的相应值分别为 46 倍和 59 倍。因此,SparkBWA 平均分别比 SEAL 和 pBWA 快 1.9 倍和 1.4 倍。

最后,我们评估了 BWA-MEM 算法,并考虑了以下工具:BWA、BigBWA、Halvade 和 SparkBWA。图 11展示了不同映射器（核心)数量下所有数据集的相应执行时间。BWA 使用 Pthreads 来并行化

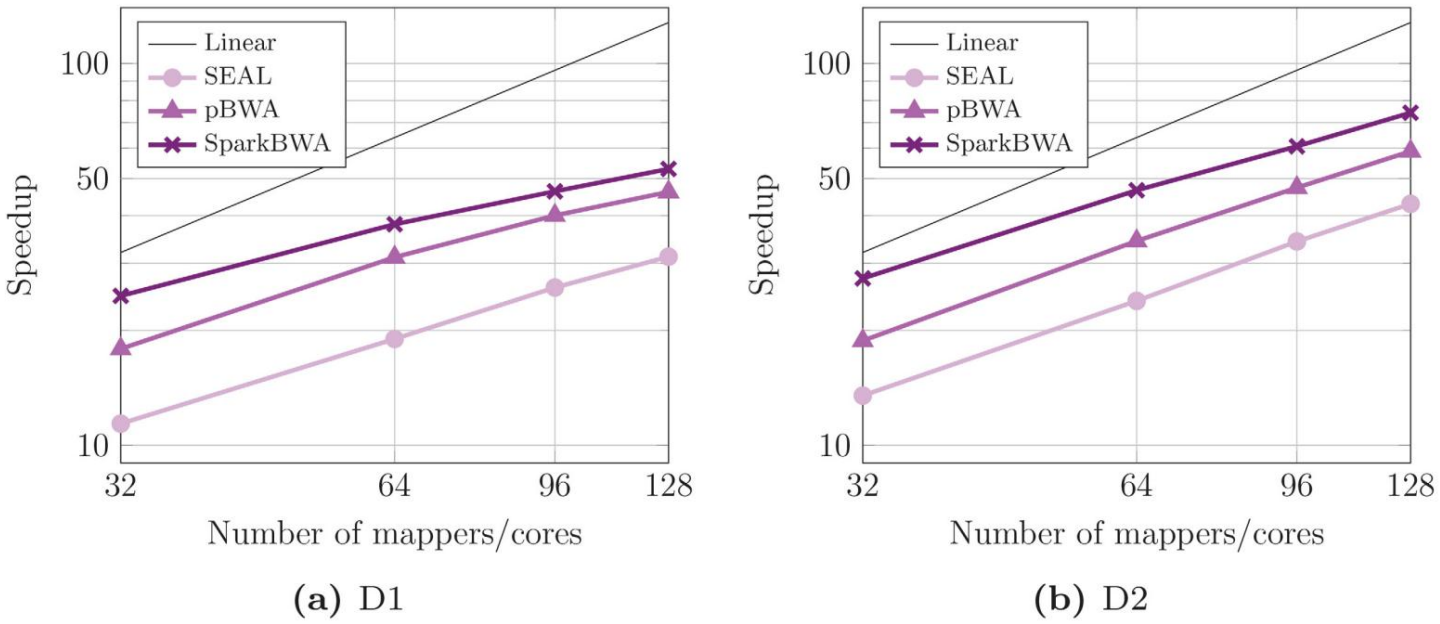

图 10. 考虑运行 BWA-backtrack 算法的几个基于 BWA 的对齐器的加速（轴为对数尺度）。

doi:10.1371/journal.pone.0155461.g010

由于比对过程复杂,因此只能在单个集群节点（64 核)上执行。BigBWA 和 Halvade 均基于 Hadoop,并且都需要预处理阶段来准备比对过程的输入数据。BigBWA 平均需要 2.4、5.8 和 23.6 分钟来预处理每个数据集,而 Halvade 则分别需要 1.8、6.6 和 22.7 分钟。BigBWA 的预处理是顺序执行的,而 Halvade 可以并行执行。

此开销与计算中使用的映射器数量无关。为了比较公平起见,此阶段的开销已包含在两个工具的相应执行时间中,因为 BWA 和 SparkBWA 的时间涵盖了整个比对过程。

性能结果表明,当使用 32 个映射器时,BWA 与基于 Hadoop 的工具（BigBWA 和 Halvade)相比具有竞争力,但其可扩展性非常差。除非数据集足够大,否则在计算中使用更多线程并不能补偿由同步引起的开销。BigBWA 和 Halvade 相对于 BWA 表现出更好的整体性能。这两种工具的行为方式相似,性能差异很小。最后,SparkBWA 优于所有考虑的工具。为了说明我们提案的优势,值得注意的是,例如,当使用 128 个映射器时,SparkBWA 比 BigBWA 和 Halvade 平均快 1.5 倍,而当使用 64 个映射器时,SparkBWA 比 BWA 平均快 2.5 倍。

相对于 BWA 顺序执行的加速性能结果
如图 12所示。图中清楚地揭示了 BWA 的可扩展性问题。
基于 Hadoop 的工具表现出更好的可扩展性,但还不足以接近 SparkBWA。
使用 128 个工作器时,BigBWA 和 Halvade 的平均加速比分别为 50 倍和 49.2 倍。SparkBWA 的这一数值最高可达 72.5 倍。值得注意的是,SparkBWA 在处理最大数据集时（图 12(c)）,其可扩展性尤为出色,最高加速比可达 85.6 倍。换句话说,并行效率为 0.67。

由此可见,SparkBWA 在所有场景下都表现出极高的一致性,并超越了其他基于 BWA 的先进对齐器所获得的结果。此外,我们必须强调的是,随着数据集规模的扩大,SparkBWA 的表现会更加出色。

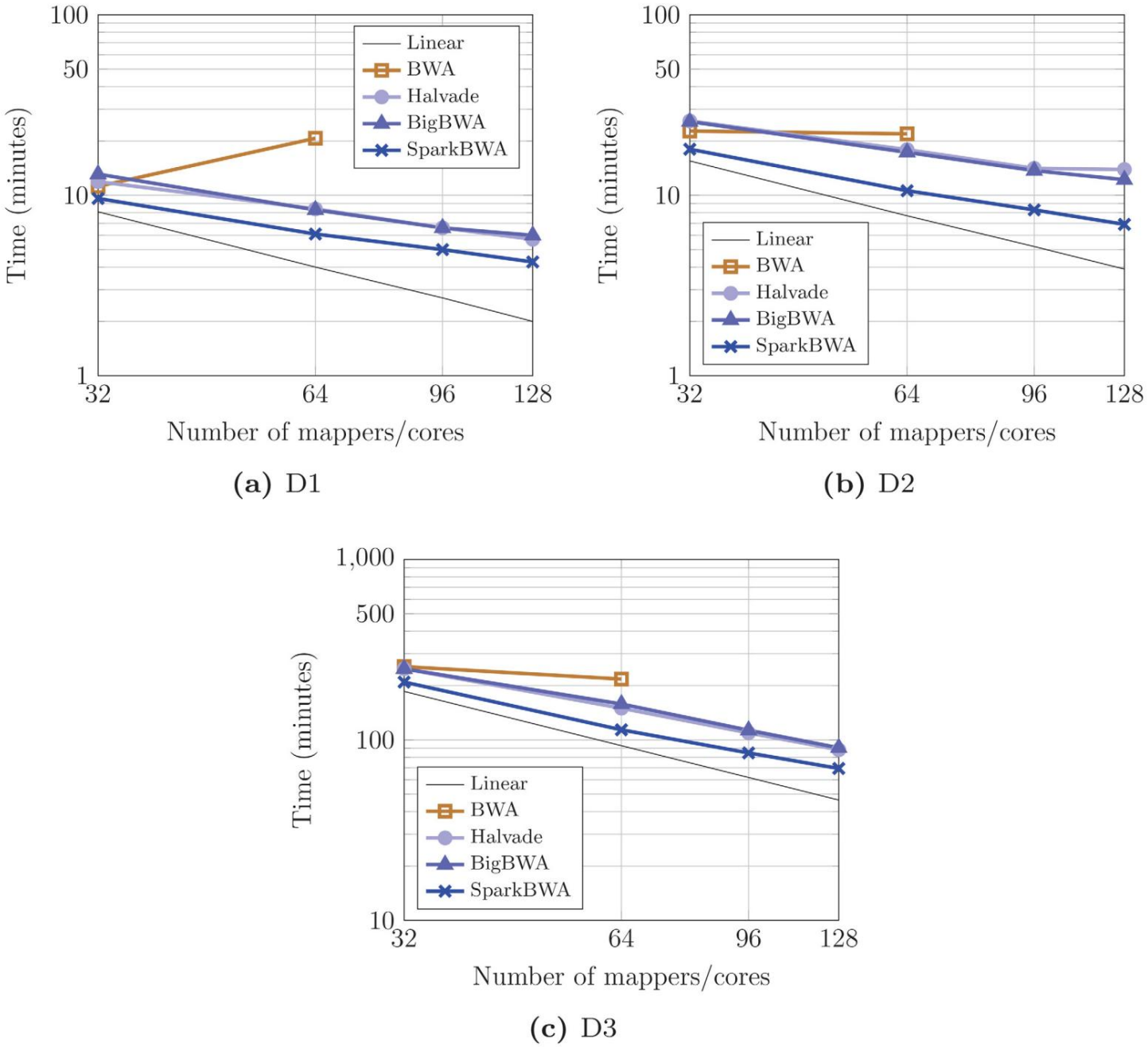

图 11. 考虑运行 BWA-MEM 算法的几个基于 BWA 的对齐器的执行时间（轴为对数尺度）。

doi:10.1371/journal.pone.0155461.g011

6 结论

在本文中,我们介绍了 SparkBWA,这是一款利用 Apache Spark 等大数据技术来提升 Burrows-Wheeler 比对软件 (BWA) 性能的新工具。BWA 是一款非常流行的软件,用于将 DNA 序列读取映射到大型参考基因组。BWA 包含几种专门针对短读取比对的算法。SparkBWA 的设计理念是无需对原始 BWA 进行任何修改

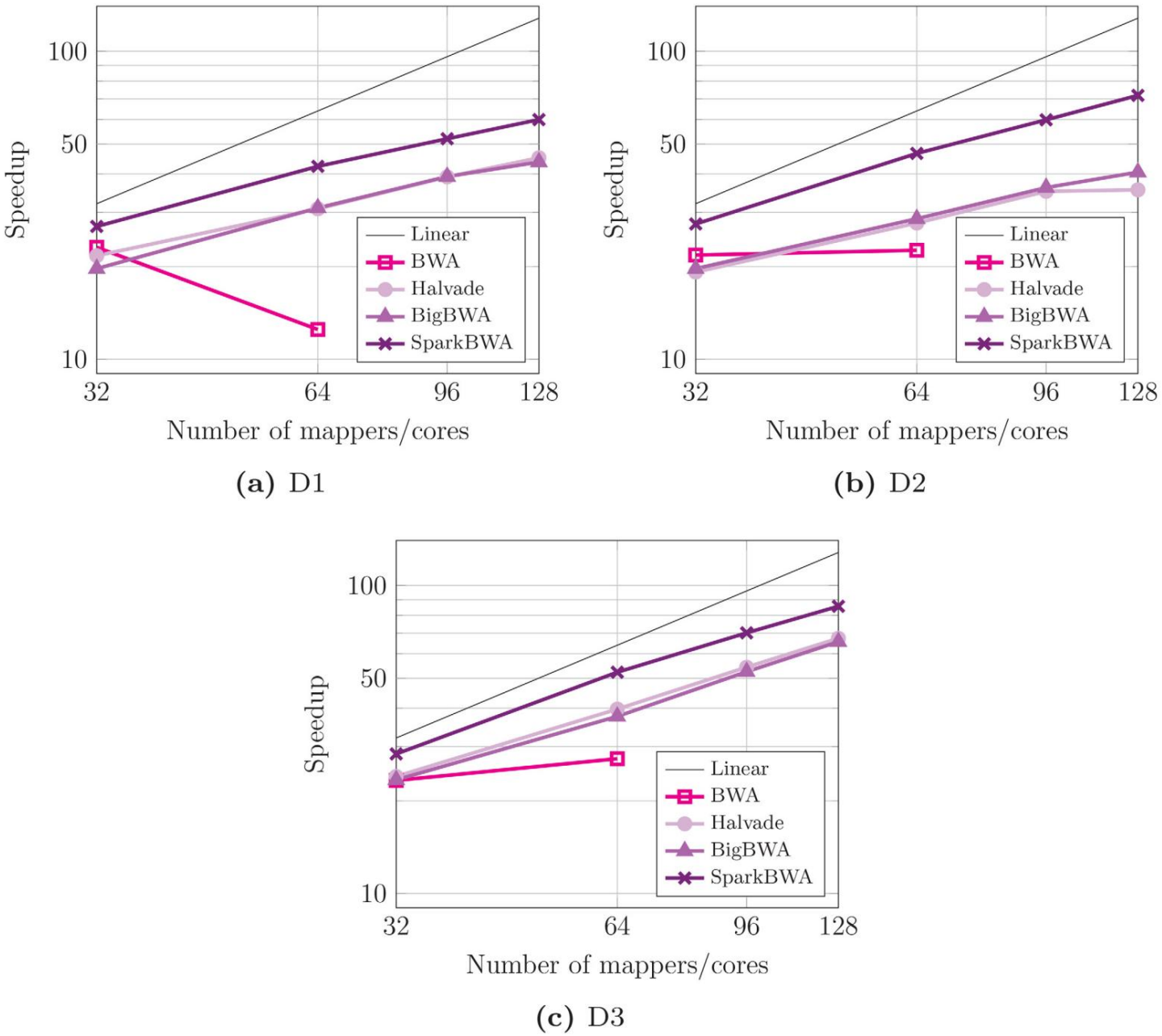

图 12. 考虑运行 BWA-MEM 算法的几个基于 BWA 的对齐器的加速（轴为对数尺度）。

doi:10.1371/journal.pone.0155461.g012

无需源代码。这样,SparkBWA 就能与任何 BWA 软件版本（无论是未来版本还是旧版本)保持兼容。

从性能、可扩展性和内存方面对 SparkBWA 的行为进行了评估
消耗。此外,SparkBWA 与几种状态的彻底比较
进行了基于 BWA 的对齐。这些工具利用不同的并行
诸如 Pthreads、MPI 和 Hadoop 等方法来提高 BWA 的性能。评估表明,当考虑对齐短读取的算法
（BWA-backtrack)时，
SparkBWA 平均比 SEAL 和 pBWA 快 1.9 倍和 1.4 倍。对于较长的读取和

BWA-MEM算法,SparkBWA相对于BigBWA和Halvade工具实现的平均加速比为1.4倍。

最后,值得注意的是,大多数新一代测序 (NGS) 专业人员并非大数据或高性能计算方面的专家。因此,为了使 SparkBWA 更适合这些专业人员,我们提供了一个简单灵活的 API,以方便社区采用这款新工具。该 API 允许从 Apache Spark shell 管理序列比对过程,并向用户隐藏所有计算细节。

SparkBWA 的源代码在 GitHub 存储库（https://github.com/citiususc/SparkBWA)上公开提供。

致谢这项工作得到了Ministryio de

Economía y Competitividad （西班牙http://www.mineco.gob.es)的支持拨款编号:TIN2013-41129-P 和 TIN2014-54565-JIN。本研究未收到其他外部资金。

作者贡献

实验构思及设计:J. Abuín JCP TFP J. Amigo。实验执行:J. Abuín。数据分析:J. Abuín JCP。试剂/材料/分析工具贡献:J. Abuín JCP TFP J. Amigo。论文撰写:J. Abuín JCP TFP J. Amigo。

参考

1. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark:基于工作集的集群计算。载于:第二届 USENIX 云计算热点会议（HotCloud)论文集;2010 年,第 10-10 页。

2. Li H, Durbin R. 利用Burrows-Wheeler变换实现快速准确的短读比对。生物信息学。2009;25(14):1754–1760。doi: 10.1093/bioinformatics/btp324 PMID： 19451168

3. Dean J, Ghemawat S. MapReduce:简化大型集群数据处理。《第六届操作系统设计与实现研讨会论文集》第6卷。

OSDI 04。美国加利福尼亚州伯克利:USENIX 协会;2004 年,第 10-10 页。

4. Apache Hadoop 主页； http://hadoop.apache.org/。

5. Shvachko K、Kuang H、Radia S 和 Chansler R. Hadoop 分布式文件系统。《IEEE 第 26 届海量存储系统与技术研讨会 (MSST)》论文集;2010 年,第 1-10 页。

6. Vavilapalli VK、Murthy AC、Douglas C、Agarwal S、Konar M、Evans R 等。Apache Hadoop YARN:又一个资源协商器。《第四届云计算研讨会（SOCC)论文集》,2013 年,第 5:1–5:16 页。

7. Srirama SN、Jakovits P、Vainikko E. 利用 MapRe-duce 将科学计算问题应用于云端。《未来一代计算机系统》。2012;28(1):184–192。doi: 10.1016/j.future.2011.05.025

8. Islam M、Huang AK、Battisha M、Chiang M、Srinivasan S、Peters C 等。Oozie:面向 Hadoop 的可扩展工作流管理系统。载于:第一届 ACM SIGMOD 可扩展工作流执行引擎与技术研讨会论文集。ACM;2012 年,第 4 页。

9. Cascading 主页； http://www.cascading.org/。

10. Zaharia M、Chowdhury M、Das T、Dave A、Ma J、McCauley M 等。弹性分布式数据集:面向内存集群计算的容错抽象。载于:第九届 USENIX 网络系统设计与实现会议论文集;2012 年,第 2-2 页。

11. Apache Cassandra 主页； http://cassandra.apache.org/。

12. Apache HBase 主页； http://hbase.apache.org/。

13. Apache Parquet 主页； http://parquet.apache.org/。

14. Hindman B、Konwinski A、Zaharia M、Ghodsi A、Joseph AD、Katz R 等。Mesos:数据中心细粒度资源共享平台。《第八届 USENIX 网络系统设计与实施会议论文集》,2011 年,第 295–308 页。

15. Li H, Durbin R. 利用Burrows-Wheeler变换实现快速准确的长读比对。生物信息学。2010;26(5):589–595。doi: 10.1093/bioinformatics/btp698 PMID: 20080505

16. Li H. 使用 BWA-MEM 比对序列读取、克隆序列和组装重叠群。
arXiv:13033997v2。2013；。

17. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N 等。序列比对/图谱格式和SAM工具。生物信息学。2009;25(16):2078–2079。doi: 10.1093/bioinformatics/btp352 PMID： 19505943

18. Cock PJA, Fields CJ, Goto N, Heuer ML, Rice PM. 含质量评分序列的Sanger FASTQ文件格式,以及Solexa/Illumina FASTQ变体。《核酸研究》。2010;38(6):1767–1771。doi: 10.1093/nar/gkp1137 PMID： 20015970

19. Abuín JM、Pichel JC、Pena TF、Amigo J. BigBWA:接近洞穴 Wheeler Aligner 与大数据技术。生物信息学。 2015年； 31（24）:4003-4005。

20. Decap D、Reumers J、Herzeel C、Costanza P、Fostier J. Halvade:基于 MapReduce 的可扩展序列分析。《生物信息学》。2015;31(15):2482–2488。doi: 10.1093/bioinformatics/btv179 PMID： 25819078

21. Pireddu L、Leo S、Zanetti G. SEAL:一种分布式短读映射和重复删除工具。
生物信息学。2011;27(15):2159–2160。doi: 10.1093/bioinformatics/btr325 PMID： 21697132

22. Leo S, Zanetti G. Pydoop:用于 Hadoop 的 Python MapReduce 和 HDFS API。见:第 19 届研讨会论文集
syum on HPDC;2010 年,第 819–825 页。

23. Peters D, Luo X, Qiu K, Liang P. 利用 pBWA 加速大规模下一代测序数据分析。《应用生物信息学与计算生物学杂志》。2012;1(1):1–6。

24. Klus P、Lam S、Lyberg D、Cheung MS、Pullan G、McFarlane I 等。BarraCUDA 一种使用图形处理单元的快速短读序列比对器。BMC 研究笔记。2012;5:27。
doi: 10.1186/ 1756-0500-5-27 PMID： 22244497

25. Liu Y, Schmidt B, Maskell DL. CUSHAW:基于Burrows-Wheeler变换的CUDA兼容短读比对器,用于大型基因组比对。《生物信息学》。2012;28(14):1830–1837。doi: 10.1093/ bioinformatics/bts276 PMID： 22576173

26. 刘春梅,黄天赐,吴鹗,罗蓉,姚思明,李燕等。SOAP3:基于GPU的超快速短读并行比对工具。生物信息学。2012;28(6):878–879。doi: 10.1093/bioinformatics/bts061 PMID： 22285832

27. Luo R, Wong T, Zhu J, Liu CM, Zhu X, Wu E 等. SOAP3-dp:快速、准确、灵敏的 GPU-
基于短读比对。PLoS ONE。2013;8(5)。doi: 10.1371/journal.pone.0065632

28. Cui Y, Liao X, Zhu X, Wang B, Peng S. mBWA:一种大规模并行序列读取比对器。载于 :第八届计算生物学和生物信息学实际应用国际会议。《智能系统与计算进展》第294卷;2014年,第113-120页。

29. You L、Congdon C. 为英特尔至强融核协处理器构建和优化 BWA ALN 0.5.10； https://
github.com/intel-mic/bwa-aln-xeon-phi-0.5.10。

30. Luo R, Cheung J, Wu E, Wang H, Chan SH, Law WC 等。MICA:一种充分利用众集成核心架构 (MIC) 的快速短读比对器。BMC 生物信息学。2015;17:7。

31. Arram J, Tsoi KH, Luk W, Jiang P. 基因序列比对的硬件加速. 可重构计算:架构、工具和应用计算机科学讲义. 2013;
7806:13–24。

32. Sogabe Y, Maruyama T. 一种利用 FPGA 加速短读映射的方法。在 :现场可编程技术 (FPT) 国际会议;2013 年,第 350-353 页。

33. Waidyasooriya H, Hariyama M. 基于Burrows-Wheeler变换的短读比对硬件加速。IEEE并行与分布式系统学报。2015;页(99):1–1。

34. Zhao G, Ling C, Sun D. SparkSW:用于大规模生物序列比对的可扩展分布式计算系统。见:第 15 届 IEEE/ACM 集群、云和网格计算（CCGrid)国际研讨会;2015年,第 845–852 页。

35. Altshuler D 等人。基于群体规模测序的人类基因组变异图谱。《自然》。2010 年;
467:1061–1073。 doi： 10.1038/nature09534 PMID： 20981092