极道科技（北京）有限公司

极道 Bioflow

WDL流程和作业使用说明

v1.7

WWW.XTAOTECH.COM

EXPLOIT DATA POTENTIAL

目的

Bioflow是以微服务的设计思想开发的生物信息作业调度器，基于容器调度构建面向生物信息的分布式计算平台。

Bioflow提供完全开放平台，可快速集成用户自定义的生物分析工具（Item），轻松管理和构建Pipeline，将数据应用于Pipeline生成Job，调度Job执行，操作简易快捷容易上手。面对大规模调度需求，能联合多调度器，多资源分配器，消除传统调度软件瓶颈障碍，解决系统规模和扩展能力难题。

使用Bioflow工具运行WDL的典型应用场景是：

- 编写WDL脚本。请参考官方网站的WDL语法说明，Bioflow完全兼容最新的WDL语法（https://software.broadinstitute.org/wdl/documentation/）。
- 定义Pipeline，指明WDL源代码及主Workflow程序。所谓主Workflow 程序指的是用户想要启动执行的那个Workflow。主Workflow可以以Task Call的方式调用其它的Workflow完成复杂的任务。
- 定义Job文件。Job文件指定需要执行的流程和主Workflow的输入。
- 通过bioflow命令行或者UI提交pipeline和job。Job将自动调度执行。
- bioflow命令行或UI可以管理（列举、暂停、取消、继续）job，并查看job的执行状态和细节。
- 定义的pipeline或者命令行可以被后续的job重用。

Bioflow语言工具帮助用户整合他们的二进制或容器镜像为一个完整的任务。它的设计目的是：

- 简化计算平台和生物信息开发人员的合作方式，两者不需要有编程接口间的依赖。生物信息开发人员将二进制工具提交到平台后即可以使用，不需要平台提供者介入。
- 简化生物信息开发人员与IT支持人员的合作方式。生物信息开发人员可以自由使用任意

或者任意版本的分析软件，自由升级和降级，无需IT人员介入，且不影响其它分析人员。

- 简化生物分析人员、计算系统和存储系统的交互方式。生物分析人员使用简单、相对的方式引用和访问数据，不依赖于IT人员管理存储的方式。Z

核心概念

强烈建议用户首先全面的阅读和理解Bioflow的核心概念。此节描述Bioflow是如何运行一个Pipeline任务。

流程语言

Bioflow支持两种语言编写流程：BSL和WDL。其中BSL的用法请参考极道的BSL语法文档。WDL的语法则完全兼容WDL的最新标准。具体请参考链接：https://software.broadinstitute.org/wdl/documentation/ 。

Pipeline

如果一个用户希望完成一个计算,可能需要按照次序执行许多步骤。其中一些步骤并行运行，每一步都会调用一些命令参数，使用或者生成一些文件。首先用户需要用WDL脚本表达比如:

- 为每个步骤定义一个WDL Task。用户需要指定容器镜像、命令行、资源需求和参数模板。Task中还可以指定一些极道特有的Runtime属性。
- 将多个Task组成Workflow，Workflow之间可以互相调用。
- 执行主Workflow需要给参数进行赋值，这在Workflow的输入文件中指定。Pipeline不包括输入文件，这些需要在Job文件中指定。

Pipeline JSON格式

Pipeline主要定义和描述流程，用JSON文件形式定义出来。

```
{
	"Name" : "gatk3-call-workflow",
	"Type" : "WDL",
	"Description" : "A pipeline for gatk3 calling",
	"WorkDir" : "vol1@alamo:bioflow/wdl-user",
	"Wdl" : {
		"WorkflowFile" : "call-workflow.wdl" ,
		"Blob" : "xxxxx"
	}

}
```

上述定义过程包括了基本信息：

- Name——Pipeline标识符
- Description——Pipeline描述
- Type——Pipeline的语言类型，此处为WDL
- WorkDir——Pipeline默认的工作目录，可以不指定。如果指定，运行的job的工作目录统一放在该目录下。
- Wdl——定义Pipeline引用的WDL程序信息。其中只需要指明：

  1）WorkflowFile：它指向WDL脚本代码目录的主程序文件，包含主Workflow定义。

  2）Blob：打包并压缩WDL源程序目录成为一个文件，然后读出整个文件作为字节流。打包的算法请咨询极道。极道提供Golang的源程序代码。

需要注意的是：Pipeline的JSON文件并不要求指定WDL程序的位置。主程序文件的路径相对源代码目录。通过biocli添加流程的时候需要指定源代码的路径。一个操作示例如下所示。

```
[jason@Cc1Biofl gatk3]$
[jason@Cc1Biofl gatk3]$ pwd
/home/jason/examples/wdl/gatk3
[jason@Cc1Biofl gatk3]$ ls
gatk3  gatk3-call.json  gatk3-call-workflow.json  job-gatk3-call.json  job-gatk3-call-workflow.json
[jason@Cc1Biofl gatk3]$ ls gatk3
bwa.wdl  call.wdl  call-workflow.wdl  gatk3.wdl  gatktool.wdl  merge_workflow.wdl  picard.wdl  samtools.wdl
[jason@Cc1Biofl gatk3]$ cat gatk3-call-workflow.json
{
    "Name" : "gatk3-call-workflow",
    "Type" : "WDL",
    "Description" : "A pipeline for gatk3 calling",
    "WorkDir" : "vol1@alamo:bioflow/wdl-user",
    "Wdl" : {
            "WorkflowFile" : "call-workflow.wdl"
    }
}
[jason@Cc1Biofl gatk3]$ biocli pipeline add gatk3-call-workflow.json -d gatk3
The pipeline added success
[jason@Cc1Biofl gatk3]$
```

Job作业

Pipeline仅仅包含WDL的Workflow程序，参数与数据均是不确定的，所以用户需要定义Job作业去指明数据和明确每一个参数值。因此，一个Job作业就是根据明确的数据和参数运行Pipeline模板的实例。

Bioflow以数据驱动的方式执行，只有当Job作业与Pipeline同时明确时，任务和执行次序才被完全确定。如果用户已经定义Pipeline，可以使用不同的数据和参数定义和提交作业，实现Pipeline一次定义，多次使用。

一个WDL的流程的Job定义包括几个组件：

- Pipeline——作业执行哪个Pipeline
- Work Directory——作业Pipeline在哪个目录下执行
- Priority——作业执行的优先级，取值0～10，10为最高，0为最低。默认值是0。
- ExecMode——作业执行模式。“Docker”是默认值，表明任务在Docker容器中运行。如果设置为“Host”，任务将在计算主机的操作系统上运行，请先确保用户信息和软件被正确安装。用户还可以设置为“Singularity”以Singularity容器的方式运行任务。
- InputDataSet——指明输入数据集。对WDL的job来说，只需要指定WorkflowInput。其中格式与WDL定义的标准Workflow输入格式相同。

一个典型的WDL作业的JSON定义如下所示：

```
{
    "Name" : "gatk3-wdl-job",
    "Pipeline" : "gatk3-call-workflow",
    "InputDataSet" : {
        "WorkflowInput" : {
                "VarationCall.readPairs" : [
                    ["vol1@alamo:bioflow/samples/control/161001_NS500823_HN5TGBGXY_L001_HUMCAEEEHPEI-276-v11_1.fq.gz", "vol1@alamo:bioflow/samples/control/161001_NS500823_HN5TGBGXY_L001_HUMCAEEEHPEI-276-v11_2.fq.gz"],
                    ["vol1@alamo:bioflow/samples/control/161001_NS500823_HN5TGBGXY_L002_HUMCAEEEHPEI-276-v11_1.fq.gz", "vol1@alamo:bioflow/samples/control/161001_NS500823_HN5TGBGXY_L002_HUMCAEEEHPEI-276-v11_2.fq.gz"],
                    ["vol1@alamo:bioflow/samples/control/161001_NS500823_HN5TGBGXY_L003_HUMCAEEEHPEI-276-v11_1.fq.gz", "vol1@alamo:bioflow/samples/control/161001_NS500823_HN5TGBGXY_L003_HUMCAEEEHPEI-276-v11_2.fq.gz"],
                    ["vol1@alamo:bioflow/samples/control/161001_NS500823_HN5TGBGXY_L004_HUMCAEEEHPEI-276-v11_1.fq.gz", "vol1@alamo:bioflow/samples/control/161001_NS500823_HN5TGBGXY_L004_HUMCAEEEHPEI-276-v11_2.fq.gz"]
                ],
            "VarationCall.reference" : "vol1@alamo:biodata/data/ref/bwa/hs37d5/hs37d5.fa",
            "VarationCall.siteFile1" : "vol1@alamo:biodata/data/database/gatk_b37/dbsnp_138.b37.vcf",
            "VarationCall.siteFile2" : "vol1@alamo:biodata/data/database/gatk_b37/1000G_phase1.indels.b37.vcf",
            "VarationCall.siteFile3" : "vol1@alamo:biodata/data/database/gatk_b37/Mills_and_1000G_gold_standard.indels.b37.vcf",
            "VarationCall.sample" : "frank"
        }
    },
    "Priority" : 10

}
```

作业JSON格式定义的有效字段包括：

字段	值	描述
Name	字符串	作业描述名称，可以不唯一
Pipeline	字符串	1. Pipeline标识符 2. 必需项
Description	字符串	1. 描述 2. 可选项
WorkDir	有效目录URI	1. 作业的工作目录 2. 可选项，如果没设置将使用Pipeline的
InputDataSet	JSON描述的数据集，仅有一项与WDL有关： WorkflowInput	1. 标准的WDL的workflow输入文件的JSON格式
SMId	字符串	1. 作业输入数据的样本名称
Priority	0~10	1. 优先级，0为最低，10为最高。默认为0。

	1) Docker 2) Host	1. 任务执行模式，Docker为容器方式执行，Host为直接在主机上运行。

Constraints	{
	“key1” : “value1”,
	…
	}
		用户可以指定job的task在带有哪些属性的服务器上运行。具体key/value的设定请咨询极道工程师和系统管理员。

StageQuota	整数	指定该Job已经调度出去但是未完成的Stage数的最大值。达到这个值，将不再提交该Job的新的stage，直到有stage完成释放了一些Quota。
ScheduleDomains	1或者多个字符串用“，”分开，例如: 1) “dom1,dom2” 2) “dom3”	调度域类似传统调度器的队列，系统管理员将一个或者多个计算节点指定为一个调度域。用户调度Job指定调度域，将任务投递到对应的调度域的计算节点上运行。

Volumes	JSON格式如下:
	{
	“/data/vol1” : “/mnt/xxx”,
	“/data/vol2” : “/mnt/yyy”
	}
		指定Job级别的卷映射，即Job所有Task启动的Docker容器都会额外映射这些卷（除开Bioflow自动映射的卷之外）。
		注意：格式是“容器路径”：“主机路径”。

EnvVars	JSON格式如下:
	{
	“EnvVar1” : “xxxxx”,
	“EnvVar2” : “yyyy”
	}
		指定Job作业级别的环境变量。该Job执行的所有Task都会附加这些环境变量。

Labels	JSON格式如下：
	{
	“Key1” : “value1”,
	“key2” : “value2”
	}
		附加到Job的所有Task的标签，以Key、Value对的形式存在

ExecMode	取值范围为： 1) docker: 表示认为以Docker 容器化模式运行（默认值） 2) singularity：表示以Singularity模式运行 3) host：表示任务以进程方式在操作系统主机上运行	用以指定任务运行方式，支持三种执行形式，默认以Docker形式执行。
MemLimi-tRatio	Float	-1： Job运行的所有task的内存不受限制 x > 1: Job运行的所有task的内存限制是申请值的x倍 其它值：内存限制与申请值相同

InputDataSet的JSON定义格式如下：

字段	值	描述

Files	字符串数组 []string	指明Job需要处理的文件/对象列表。 注意：如果用户的Job的输入数据以Object的形式存在，需要将它放在Files中。Bioflow将自动下载到本地，然后计算。
FilesInOrder	二维字符串数组 [][]string	用来存储Job需要的有序输入Object列表。与Files的区别在于，第二层数组的数据会被Bioflow按照在数组中的顺序下载到本地再计算。
InputDir	输入数据所在的目录或者bucket	1. BSL流程：存放输入数据所在的目录 2. WDL 流程：没有使用。当使用Object作为输入的时候指定对象所在的bucket。
WorkflowInput	JSON Map	标准的WDL的workflow输入文件的JSON格式
InputMap	JSON Map	BSL job的输入参数
RestorePath	String	如果Job的输入数据存放在Object的bucket上，此处指定上传Job计算结果的bucket。默认是输入的bucket。
RestoreFilter	String	如果Job的输入数据存放在Object的bucket上，此处指定上传计算结果到bucket上object id的前缀。这用于方便用户快速搜索访问某个job的结果。

LogPath	String	如果Job的输入数据存放在Object的bucket上，此处指定上传Job的日志的bucket。默认是输入的bucket。

DataSet数据集

Bioflow是数据驱动调度器，作业需要明确数据集，以便本Job作业的Pipeline执行。WDL作业的InputDataSet只需要指明WorkflowInput。

WorkflowInput

WorkflowInput主要以JSON格式定义主Workflow的参数和Task的参数。其中WDL标准严格定义了各种WDL数据类型如Array、Map、Pair或者Object的标准输入格式。Bioflow严格遵守这些标准。

下述是一个简单的示例：

```
{
  "wf.t1.s": "some_string",
  "wf.t2.s": "some_string",
  "wf.int_val": 3,
  "wf.my_ints": [5,6,7,8],
  "wf.ref_file": "/path/to/file.txt"
}
```

其中每一项的格式是：

```
“<workflow name>.<task name>.<variable name>” : <variable type literal>,
                                …
```

需要注意的是：

1）其中WorkflowInput的各个变量的输入格式请严格按照WDL标准（https://github.com/openwdl/wdl/blob/master/versions/draft-3/SPEC.md）中 “Array Literals”，“Map Literals”，“Object Literals”或者“Pair Literals”章节规定的格式。

2）如果一个task或者workflow被call了多次，需要将其中的<task name>换成相应的call name。call name的具体解释上述链接中也有明确的规定。

3）JSON文件中WorkflowInput实质上是标准中“Specifying Workflow Inputs in JSON”章节规定的。

File路径统一表示

WDL标准支持File类型，用户可以通过指明路径访问一个文件。Bioflow为了提高用户的流程的可移植性，建议用户以“卷@集群：路径”的方式指定文件路径。这样方便用户自由使用任意存储或者存储集群的数据，无需再引入任何相对于客户端的绝对路径（例如/mnt/store1/vol1/…/hs357.fa）。

Bioflow在执行用户的Workflow的时候，对File类型的变量自动处理，为用户任务的Docker容器自动挂载卷，并将用户Task的command部分的命令中File变量对应的路径名转换为相对容器的路径名。用户的程序就像工作在本机一样在服务器端执行。这将极大的方便用户的使用。

文件或目录URI可以是这样：

```
“Volume name”@”cluster name”: full path file name
```

请看以下示例：

```
Vol1@xtaofs1:/var/data/jobs/job1/testfile1
Vol2@xtaofs2:/root/hellos/kkfile
Vol3:/Jason/workdirs/files/test3
```

最后一条URI记录省略了集群名称，在只有一个集群的时候，允许这样的表示。

文件路径表示中的概念和使用方式为：

1. volname是卷的名字，通常对应一个文件系统。它只是一个文件系统或者名字空间的描述，不要求该存储有对应的“卷”的概念。只需要给它起个名字，然后系统管理员知道如何挂载对应的数据即可。
2. cluster的名字也是逻辑概念，只需要给不同存储集群不同的名字即可。

因此URI只是用于将不同文件系统的数据区分开，且不引入任何特定挂载的绝对路径。具体的卷和集群的数据在计算节点上如何挂载，可以在bioflow中动态配置。数据分析人员不用关心。

兼容绝对路径

为了兼容用户使用习惯，最新版本的Bioflow允许用户在变量赋值的时候采用绝对路径（例如:/mnt/vol1/….），Bioflow自动识别绝对路径中的存储和卷信息，运行任务时候的完成存储卷的自动映射，达到上节中同样的效果。

WDL语言及Bioflow支持

标准功能

Bioflow 支持WDL语言标准定义的所有语言特性，其中不但完全支持了初始版本，还支持了最新标准（标准本身尚在制定）中的能够提升用户体验的部分功能。请访问链接https://

github.com/openwdl/wdl获取WDL语言标准。需要特别指出的是：

- Bioflow支持任意层数的scatter嵌套

例如，用户可以运行下述类型的WDL程序。

```
workflow mainworkflow {
    Array[String] idArray
    Array[Int] numArray

    …

    scatter( id in idArray) {
        . . .
        scatter( num in numArray) {
          …
        }
    }
}
```

其中scatter嵌套的层数没有上限。

- Bioflow支持scatter的collection**类型为map**

```
workflow mainworkflow {
    Map[String, Int] map
    …

    scatter( pair in map) {
        String key = pair.left
        Int value = pair.right
    }

    output {
        Array[String] keys = key
        Array[Int] values = value
    }
}
```

- Bioflow支持WDL脚本中定义一个变量之前使用它。

```
workflow mainworkflow {
    String a
    String b = c
    String c = call1.output

    …
    call task1 as call1 {
      …
    }
    …
}
```

- Bioflow支持after关键字。

当两个task的call之间没有输入、输出的依赖关系的时候，bioflow将以任意的次序执行它们。用户可以通过“after”关键字在两个或者多个call之间建立执行顺序关系。示例程序如下图所示。其中call1将在call2和call3之后执行，call2和call3之间没有顺序关系。

- Bioflow支持特殊的Runtime属性，帮助用户实现更好的使用体验和性能。详细请参考下一节。
- Bioflow支持Struct结构以及绝大多数在Development版本的标准，请参考https://github.com/openwdl/wdl/blob/main/versions/development/SPEC.md

Bioflow特有的Runtime特性

```
task task1 {
    …
}

task task2 {
    …
}

workflow mainworkflow {
    …
    call task1as call1 after call2 after call3 {
      …
    }
    …
    call task2 as call2 {
      …
    }

    call task2 as call3 {
      …
    }
    …
}
```

WDL标准允许不同的提供商支持不同的运行时属性来为用户提供更高级的功能。Bioflow支持所有标准的Runtime属性：

- docker——指明task的docker镜像名字
- memory——task的内存需求
- cpu——task的cpu资源需求
- disk——task的disk资源需求

Bioflow支持下述非标准的Runtime属性，方便用户运行任务。

名字(大小写无关)	取值	描述
gpu	整数	Task需要的GPU的数量
gpui	整数	Task需要的GPU的MIG的数量

gpumemory	浮点数（M为单位）	Task需要的GPU显存大小
servertype	normal|fat	需要的服务器类型: 1）normal： 普通服务器 2）fat：胖节点
retry	整数	任务失败后重试的次数
tag	字符串表达式	通过biocli job status查看状态时，显示为task的stage的name
sparkexecutoruri	hdfs路径	用于Spark任务，用户指定spark程序所在的hdfs的路径
usextaosparkscheduler	true|false	用于Spark任务，为Spark自动设置和初始化资源分配器环境变量
usextaoscheduler	true|false	用于MPI/Tensorflow/Caffe等任务，表示是否设置极道的集群调度环境

slaveconstraints	{
	“key1” : “value1”,
	“key2” : “value2” ,
	…
	}
		指定执行任务的服务器节点的限制属性，常用于选定特定的计算节点来执行任务。

volumes	{
	“/a/dir1” : “/mnt/dir1”,
	…
	}
		用于映射主机卷到docker容器，格式为“容器路径:主机路径”

cputuneratio	浮点数，>= 0 默认值：0	如果Task失败，重投任务的时候，将CPU的资源需求设置为：上次CPU资源 * (1 + cputuneratio)。
memtuneratio	浮点数，>= 0 默认值：0	如果Task失败，重投任务的时候，将Mem的资源需求设置为：上次Mem资源 * (1 + memtuneratio)。
label	任意有效的WDL表达式	用于用户为Task添加label，可在查看job status的时候作为Task的信息显示出来。
iopattern	sequential | random | hybrid	用于为底层调度提供Task的IO访问特点，Bioflow基于该特性进行自动优化。 1）sequential：顺序访问 2）random：随机读写 3）hybrid：混合读写

rwpattern	read|write|reread|rewrite|hybrid	用于为底层调度提供Task的IO读写特点，Bioflow基于该特性进行自动优化。 1）read：以读为主 2）reread：大量重复读 3）write：以写为主 4）rewrite：大量重复写 3）hybrid：读写都有
workingset	x K/G/M，x为浮点数	Task读或者写的总大小上限
isolationlevel	stage | io	Task的IO的隔离级别。 1）stage：该Task写的数据只有在Task结束后才被其它的Task或者应用程序读取。 2）io：该Task写的数据，一旦写操作返回就有可能被其它Task访问。
epheremallevel	stage | job | user	Task写的数据的生命周期特性。 1）stage：数据在Task结束后不再需要。 2）job：数据在整个Job结束后不再需要。 3）user：数据始终需要被保留。

epheremalfilepattern	带“*”的模式字符串	与epheremallevel联合使用，表示满足该模式的文件或者目录具备指定的生命周期特性。

epheremalmap	一个JSON格式的数据:
	{
	“pattern” : “level”,
	…
	}
		用于扩展Task的属性epheremalfilepattern和epheremallevel，用户可以指定多于一对的pattern和level。

workdir	相对路径	与Job的工作目录拼接组成Task的工作目录。
starvetimeout	整数	任务的排队超时时间，单位秒。如果任务的输入数据就绪，投递到计算集群等待资源，等待时间超过这个限制就会失败返回给调度器。该选项用于防止任务因为资源不足无限等待。

globalresources	{
	“fastx-license1” : “1”,
	“gatk-license” : “1”,
	…
	}
		Task运行所需的全局资源，可用于支持浮动许可证（license）。用户指明所需要的license的key和数量。

tracesource	{
	“output.vcf” : [file1, file2],
	outputBamFile: [file3]
	}
		该选项配合极道数据管理。Bioflow将会自动根据用户指定的输入输出关系，为输出文件打上tag，指明产生它的源文件，用于后续的数据溯

ldapauth	true/false “true”/“false”	指明该容器是否安装了ldap客户端，需要通过极道计算环境的ldap服务进行用户认证。如果设为true，bioflow将为容器进行必要的设置，使得ldap认证可以工作。
runner	“partisaner”	指定task的执行引擎为partisaner。用于运行分布式的tensorflow、pytorch和mpi程序。Bioflow将读取runneroptions，通过极道的partisaner集群，自动构建tensorflow、mpich或者pytorch集群进行计算。用户无需知道partisaner
runneroptions	JSON格式的数据	与runner选项一起工作，提供tensorflow、pytorch或者mpich集群的配置信息。

scheduledomains	1或者多个字符串用“，”分开，例如: 1) “dom1,dom2” 2) “dom3”	调度域类似传统调度器的队列，系统管理员将一个或者多个计算节点指定为一个调度域。用户调度Job指定调度域，将任务投递到对应的调度域的计算节点上运行。
execmode	取值范围为: 1) docker: 表示认为以Docker容器化模式运行（默认值） 2) singularity: 表示以Singularity模式运行 3) host: 表示任务以进程方式在操作系统主机上运行	用以指定任务运行方式，支持三种执行形式，默认以Docker形式执行。
networkmode	网络模式，其取值范围为: 1）host: 使用主机网络 2）bridge：使用网桥模式 3）none: 不指定，采用docker或Singularity默认网络选项（默认值） 4）PTP:容器之间通过PTP网络	容器运行的时候，调度器根据该配置选择网络模式， 默认采用none模式。应与port配合使用。 注意：目前docker支持none、host和 bridge三种模式 Singularity支持none 、bridge和ptp三种模式

port	1）在host网络模式下:
	
	port: {
	"Num": 2,
	"ports":[]
	}
	
	2）在bridge网络模式下:
	
	port: {
	"Num": 2,
	“ports": [
	{
	"ContainerPort": "20009",
	"HostPort": ""
	},
	{
	"ContainerPort": "200010",
	"HostPort": ""
	}
	]
	}
		申请端口资源：
		1）在host模式下，分配要求数量的端口，通过环境变量CONTAINER_PORTS_AVAILABLE_HOST可获取分配的端口列表。
		2）在bridge模式下，分配要求数量的端口，如果用户指定了将通过端口映射将获取的主机端口映射到指定的ContainerPort。分配到的端口可以通过环境变量CONTAINER_PORTS_AVAILABLE_BRIDGE获取

resourcescale	支持三种资源调节模式： 1) “auto” 2) up 3) down	Bioflow根据任务的运行情况动态调节任务所需的cpu和memory资源。初始值为task中定义的“cpu”和“memory”值。在模式“auto”下，task的资源可以自动增加或者减少。“up”模式只允许增加资源，“down”模式只允许减少资源。 需要注意的是，如果资源调节不及时，task仍然可能OOM退出。如果需要避免OOM退出，请参考”disableoom”属性。
disableoom	true false	如果设定为true，task任务的memory资源使用超出限制后不会因为OOM被杀掉。通常和“resourcescale”选项配合使用。

shell	所使用的Shell所在路径（字符串型），例如："/usr/bin/sh"	用户通过shell来指定自己执行的Linux shell版本。
returncodes	支持三种设置: 1） 固定为"*"（字符串型） 2：0~255 间任意整型 3：0~255间任意整数组成的整型数组，例如：[1,2,5,127]	用户可以通过returncodes自定义任务成功的退出码，设置为*则认为所有的退出码都是成功，设置为1则只有1是成功的返回码，也可以设置为整型数组则数组中的返回码都认为成功，注意默认返回码0是成功

singularity	/mnt/nfs/xx.sif /home/Jason/xx.sif	说明Task以Singularity容器格式运行，且指向一个sif文件的绝对路径
memlimitratio	Float: -1 0.5 3	-1:　task的内存不受限制 x > 1：task的内存限制是申请值的x倍 其它值：task的内存限制与申请值相同

kvtags	{
	“key1” : “value1”,
	“key2” : “~{xxx_var}” ,
	…
	}
		附加到Task上的Key、Value对，将会显示在biocli job status的task信息中
		可以引用task的变量

affinity	[
	{
	"cpu" : 10,
	"gpu" : 0,
	"memory" : "10G",
	"waittime" : 60
	},
	{
	…
	}
	]
		备选资源选项，当runtime里指定的首选资源不满足则按照等待时间顺序依次选择匹配备选资源。其它cpu、gpu、memory、gpui如果指定则使用指定的值，不指定则使用runtime中的旧值。
		waittime单位为秒，表示等待多少时间后考虑该备选项。

priority	Int: 0 ~ 10	指定Task级别的优先级，优先于Job的优先级，但是受队列优先级控制
tier	fast latency throughput slow	指定任务的IO需求类型，Bioflow将自动为任务在特定的存储集群上创建镜像workdir，使得任务达到最好的性能： 1）fast：需求快速存储 2）latency：需要延迟较低的存储 3）throught：需要高带宽存储 4）slow：可使用普通低性能存储 一般环境中，如果对任务IO性能有要求，推荐使用fast选项

fadvise	{
	"read" : [fq1, fq2],
	"write" : ["${sampleName}.html"]
	}
		一个JSON格式的说明，Key包括：
		1）read：任务读取的文件列表
		2）write：任务写的文件列表
		
		这些信息用于Bioflow自动计算任务的IO负载，实现按照io负载调度任务

resources	{
	“mpu” : 1,
	“fpga” : 2
	}
		JSON格式的说明用于指明任务需要的扩展资源（即系统内存在的非cpu、mem、disk、gpu的资源），其中键值为资源的名字，对应的值为资源量，可以为浮点数或者整数

Bioflow特有的库函数

Bioflow是支持企业级数据流计算的调度引擎，提供了自己的库函数方便用户进行大规模数据处理。

- copy_file：拷贝一个（或者多个）文件到其它存储

这个函数接受4个参数如下：

参数序号	类型	说明
1	File或者String	需要拷贝的文件路径
2	String	目标存储类型。 目前只支持”hdfs”
3	String或者File	希望拷贝文件去的目标目录路径，可以为空“”。如果为空，将自动分配存储路径。
4	String	需要同时拷贝的同一个目录下同一个名字但是不同的后缀名的文件的后缀名。 例如调用copy_file(file1，“hdfs”，“”，“idx”)，如果file1指向文件”/mnt/vol1/frank.bam”，那么将会把”/mnt/vol1/frank.bam”和”/mnt/vol1/frank.idx”都拷贝到目标路径下。

如果不使用copy_file，用户需要在WDL脚本里添加一个task拷贝数据到目标路径。使用copy_file可以避免添加额外的task。Bioflow使用内部的高性能并行数据传输引擎拷贝数据，非常高效，用户无需再关心数据拷贝的细节。
具体使用可以参考下述示例：

这个例子展示了一个省略了很多细节的结合gatk3和gatk4的spark版本的一个例子。用picard产生了一个sort过的bam文件，接下来需要用gatk4的spark版本的read pipeline来分析。但是spark版本要求文件是一个hdfs文件。如果不用copy_file，用户需要再添加一个task将bam文件拷贝到hdfs上。这个例子，用户只需要调用一个copy_file，当下次使用hdfsFile这个File变量的时候，bioflow确保它已经在hdfs上了。

- write_paths：将多个文件绝对路径写入一个文件

```
task picard_sort_task {
   …
}

task gatk4_read_pipeline {
   …
}

workflow gatk4calling {
    File hdfsFile
    …
    call picard_sort_task as SortBam {
       …
    }
    hdfsFile = copy_file(SortBam.bam, “hdfs”, “”, “”)

    …

    call gatk4_read_pipeline as ReadSparkPipeline {
        input: inputBamFile=hdfsFile, …
        …
    }

    …
}
```

这个函数接受1个参数:

参数序号	类型	说明
1	Array[File]或者 Array[String]	被写入文件的多个文件路径

数组中的File或者String形式为vol@cluster:path，bioflow会将这种形式的路径自动解析为可访问的绝对路径。

具体使用可以参考下述示例：

```
import "bwa.wdl" as bwatool
…

workflow gatk {
    Array[File] fileList
    File reads = write_paths(fileList)
    …
    call bwatool.bwa_mem_tool as MapRead {
        input: reads=reads, …
    }
    …
}
```

典型用例

本节描述了用户经常遇到的Bioflow运行WDL的问题及说明。

如何在Docker容器中访问存储？

用户Task运行时需要访问存储数据，因此需要将存储卷映射到Docker容器，Task中的command区域内的脚本访问文件的路径也需要使用相对于容器的路径。这些操作严重依赖运行环境，非常繁琐，易出错。Bioflow可以自动为用户完成这些工作，它提供了三种工作模式。

1. Bioflow自动挂载（推荐）

用户在编写WDL脚本的时候，使用File类型的变量来访问文件。文件的路径用极道的统一路径表示：volume@cluster:/path。例如：

```
File a = “vol1@cluster1:/a/b/c”
```

或者

```
String x = “vol1@cluster1:/a/b/c”
File a = x
```

如果在Task的command section中使用File变量a，例如：

```
Task {
    …
    command {
        cat ${a} > output.txt
    }
}
```

当Task的call作为docker容器执行的时候，Bioflow将自动为容器挂载文件变量a所在的卷，并将${a}转换为容器内a文件所对应的路径。多个File变量、Array[File]等都可以自动处理。所以在这种使用方式下，用户不需要做任何操作，程序像运行在本地一样运行在服务器上。

2. 使用Volume的runtime属性映射

Bioflow支持用户在Task的Runtime属性映射主机的路径到容器中，类似于docker run –v 参数的作用。

3. 使用绝对路径（不推荐）

为了兼容用户使用习惯，新版本的Bioflow允许用户在变量赋值的时候采用绝对路径（例如:/mnt/vol1/….），Bioflow自动识别到文件的存储和卷，完成运行任务时候的存储卷的自动映射。

如何运行不在Docker镜像中的程序？

有些情况下，用户的程序不方便一次性打包到Docker的镜像中。用户可以将程序的路径用Volume的方式映射到Task的容器中，然后调用程序或者脚本。极道建议用户最终的docker

镜像应该包括所有的执行程序或者脚本，但是提供手段让用户可以在开发或者测试过程中运行镜像之外的脚本或者程序。

例如用户有一个脚本，放在存储上卷下的目录/a/b/test.sh，该存储卷已经挂载到每个服务器上，目录是/mnt/vol1，则用户的task可以描述成下述方式：

```
Task {
   …
   command {
       /opt/scripts/a/b/test.sh
   }

   runtime {
     volumes : {
                    “/opt/scripts” : “/mnt/vol1”
                }
      …
   }

   …
}
```

需要注意的是：为了用户的方便，此处主机卷的描述方式采用了直接的挂载路径，而不是volume@cluster的统一表示。

如何在不同存储之间快速移动数据？

Bioflow的统一文件表示方法支持计算任务混合使用多套存储。但是某些情况下，不同的计算任务可能使用不同的协议访问存储，例如GATK3使用Posix文件系统，而GATK4使用HDFS，因此需要在不同存储集群、存储与HDFS之间拷贝大批量数据。极道提供

DataMover镜像帮助用户在不同存储之间并行拷贝数据。

Bioflow也支持更方便的copy_file调用，帮助用户快速方便的拷贝数据。具体使用参考“Bioflow特有的库函数”章节。

如何使用软链接？（不推荐）

很多应用程序会通过“ln –s xxx yyy”建立软链接。如果用户在软链接使用了绝对路径，那该链接文件将依赖于当时存储在服务器上的挂载路径。如果系统管理员更换了挂载路径，文件将失效。Bioflow支持用户使用软链接，但是不建议用户使用。

如何在计算中使用Object（Alibaba/Tencent/Xtao）数据？

在公有云的场景下，或者用户通过对象（Object）服务获取用户数据的情况下，用户Job的输入数据存在Object服务的一个Bucket中。这带来一个严重问题：用户输入是Object，但是计算任务使用Posix接口访问数据。Bioflow支持对象自动按需下载，使得用户的计算任务像所有数据在共享存储上一样工作。具体的工作方式如下：

1) 系统管理员配置对象服务所在的存储。例如：

```
Cluster aliyun
   FSType: oss-alibaba
   Servers:  xxxx
```

2) 用户在Job的InputDataSet中描述输入对象列表，或者在参数中指定对象。例如：

```
{
   …
    “InputDataSet” : {
        “WorkflowInput” : {
            …
            “mainworkflow.knownsite” : “os@aliyun:/bucket1/site-object”,
            …
        }
    }
    …
}
```

3) 用户提交Job。当用户的Task需要访问对象数据时，Bioflow将自动将数据下载到本地共享存储中，然后将文件路径转换为共享存储中的路径计算。例如：

```
workflow mainworkflow {
    …
    File knownsite;
    …

    call gatk {
        input: site=knownsite, …
    }
}
```

可以看出，用户的WDL程序不必对对象做任何特殊处理。当gatk这个call在执行的时候，knownsite指向的位置已经是object被下载到共享存储中的位置了。

4) 作业完成后，Bioflow自动将计算结果上传到Bucket中。

如何运行Gatk4的Spark版本？

Bioflow既能支持批量计算，也能支持Spark计算。以基因计算的GATK4为例，它的某些阶段调用了Spark框架进行处理。Bioflow支持用户的WDL的任意Task都可以使用Spark，且能够在同一个Workflow中混合使用批量计算和Spark。

1) 由于Gatk4的Spark需要使用HDFS，系统管理员需要部署HDFS文件系统或者使用已经

部署好的HDFS文件系统。添加一个新的计算集群：

```
Cluster spark
    FSType: hdfs
    Servers:  xxxx:xxx
```

2) 用户可以使用极道的DataMover在存储和HDFS之间传输数据，可以使用vol@cluster的方式在同一个Task中混合使用共享存储和HDFS。Bioflow将做合适的路径转换和自动挂载。

3) 对于Spark任务，需要在Task中设置合适的runtime属性，例如：

```
task analysis {
    File SparkBinary = “hdfs@spark:/tom/spark-2.2.0-bin.tgz””
     command {
     …
     }

     runtime {
         …
         SparkExecutorURI: “${SparkBinary}”
         UseXtaoSparkScheduler: true
         ldapauth: true
         …
```

其中：

1）SparkExecutorURI指定用户自己的Spark的binary的存放位置，需要在HDFS中。

2）UseXtaoSparkScheduler选项告诉Bioflow自动设置SparkMaster到极道的环境，无需用户关心。Spark集群需要SparkMaster的选项。例如Gatk4 spark版本，需要指定参数--sparkRunner SPARK --sparkMaster ${SPARK_MASTER_URI}。 其中SPARK_MASTER_URI是环境变量，当用户设定了UseXtaoSparkScheduler，可以通过该环境变量获取Spark的Master。

3）ldapauth选项告诉Bioflow为容器进行ldap认证必须的设置。很多spark或者hadoop程序需要访问用户的home目录存储一些配置。因此需要在容器内安装用户认证的基础服

务。极道使用ldap的统一认证方式，并提供基础docker镜像帮助用户构建自己的镜像。这个选项用于支持ldap认证。

如何指定单个Task或者Job在指定服务器上运行？

Bioflow为用户提供云模式的计算环境，用户无需关心任务在哪台服务器上进行计算。Bioflow保证将用户调度到合适的计算节点，可以访问需要的存储和硬件资源。但在某些特殊场景下，用户可能要求计算被调度到指定的计算节点。Bioflow提供多种方法满足这个需求。

1. 使用“胖节点”

某些生物计算任务需要使用超大内存节点，因此计算任务需要调度到指定的胖节点。用户需要在Task的runtime属性中指定ServerType，例如：

```
task analysis {
    …
    command {
    …
    }

    runtime {
        …
        servertype: “fat”
        …
    }
    …
```

2. 指定计算节点的属性

处于某些管理需求，系统管理员可能会将计算服务器分成某些区域或者组，为每个节点打上类似Key/Value对的标签。计算任务可以通过在Task的runtime属性slaveconstraint或者Job JSON的constraints中指定计算节点需要满足的Key/Value对。例如：

```
task analysis {
    …
    command {
    …
    }

    runtime {
        …
        slaveconstraint:  {
            “region” : “region1”,
            “group” : “group1”
        }
        …
    }
    …
```

如果用户提交Job的时候，在Job的JSON中指定了Constraints属性，则这个Job的所有Task将自动具备了这个SlaveConstraints属性。例如：

```
{
    …

   “Constraints”:  {
          “region” : “region1”,
          “group” : “group1”
     }
        …
}
```

如果需要指定任务在计算节点Cc2BCluster上运行，则可以在Job文件中指定：

```
{
    …

   “Constraints”:  {
          “hostname” : “Cc2BCluster”
     }
        …
}
```

3. 指定不同的用户组使用不同区域的机器

处于某些管理需求，系统管理员可能需要将计算服务器分成某些区域，让不同的用户组使用不同的区域。管理员需要进行下述配置：

1) 启用LDAP用户认证模式。极道新系统默认支持该用户认证模式。

2) 为每个计算节点打上标签, 例如： servergroups:group1,group2,group3

其中group1、group2、group3都是用户所属的group的名字。打标签的方法可与极道工程师咨询。每一个计算节点可以指定一个或者多个用户组。

3）用户jason提交job，无需在job文件中指定，job可以自动调度到jason所属的用户组的机器上运行。如果jason属于多个组，group1或者group3，job可以在标记有group1或者group3的机器上运行。

如何调度浮动许可证（license）？

有些情况下，用户可能需要使用商用软件进行计算。很多商用软件采用浮动许可证的方式。极道的平台可以很好的支持浮动许可证的调度。

1）系统管理员将License服务器部署到集群中或者计算节点可以通过网络访问到的服务器上。

2）用户将商业软件打包成容器，或者以Host模式运行。

3）系统管理员通过grmc命令添加浮动license资源（例如expensive-license 10）到极道的全局资源管理服务（GRMD）。具体操作请咨询极道工程师。

4）用户的WDL流程中使用商业软件的Task需要添加下述runtime属性：

```
Task {
   …
   command {
       /opt/start-expensivetool.sh
   }

   runtime {
      …
      globalresources : {
                        “expensive-license” : “1”
                    }
       …
   }

   …
}
```

当任务投递后，调度服务将根据浮动license资源情况决定是否启动task。如果没有足够的浮动license资源，任务将排队，直到使用license的某个任务结束。当任务被启动后，通过连接license服务器进行授权。

如何通过WDL进行分布式机器学习（Tensorflow）？

如果用户需要通过WDL运行分布式Tensorflow进行机器学习训练，极道的WDL执行引擎提供了简单的方式。

1）用户将自己的Tensorflow训练程序放在Tensorflow的docker镜像中，或者用户使用标准的Tensorflow镜像，将训练程序放在共享存储上，通过volumes选项映射进入容器。

2）用户通过写如下的Task和runtime属性runner和runneroptions，调度一个分布式Tensorflow程序：

```
Task  {
   File hostPath
   …
   command {
      export TRAIN_STEPS=50000
      sh /tfwork/bin/run.sh
   }

   runtime {
         …
         docker: "tensorflow-abc"
         cpu: "1"
         memory: "1G"
         volumes: {
            "/tfwork" : hostPath
         }

         runner: "partisaner"
         runneroptions: {
            "cluster": "tensorflow",
            "logdir": "/tfwork/logs",
            "workdir": "/tfwork",
            "loginterval": 60,
            "abortonfail": true,
            "roles": [{
                  "name": "ps",
                  "procs": 1,
                  "cpu": 2,
                  "memory": 1024,
                  "server": true
               },
               {
                  "name": "worker",
                  "procs": 2,
                  "cpu": 1,
                  "memory": 1024
               }
            ]
         }
         …
   }

   …
}
```

如上例，其中command{ … }调用的脚本为放置在共享存储上的hostPath路径的tensorflow训练程序，通过极道传入的环境变量区分role（即ps或者worker）。runneroptions中指明了tensorflow集群的配置信息。

3）Task可以像其它普通task一样引用变量、输入输出或者File，与普通task没有任何区别。

如何通过WDL运行MPICH？

如果用户需要通过WDL运行分布式MPICH程序进行高性能计算，极道的WDL执行引擎提供了简单的方式。

1）用户将自己的MPICH程序编写、编译，打包到自己的docker镜像中，或者用户使用标准的mpich镜像，将程序放在共享存储上，通过volumes选项映射进入容器。

2）用户通过写如下的Task和runtime属性runner和runneroptions，调度一个分布式集群进行MPICH计算：

```
  Task  {
     …
     File hostPath
     Int procs

     command {
        /home/mpi/mpich-install/run/calculator-PI steps 50000
     }

     runtime {
        docker: "mpich-abc"
        cpu: "1"
        memory: "1G"
        volumes: {
           "/mpiwork" : hostPath
        }

        runner: "partisaner"
        runneroptions: {
           "cluster": "mpich",
           "workdir": "/mpiwork",
           "procs": procs
        }

        usextaoscheduler: "true"
        ldapauth: "true"
     }
  }

  …
}
```

如上例，其中command{ … }调用的脚本为打包进mpich-abc镜像的MPICH程序。runneroptions中指明了MPICH集群的配置信息，包括了运行的进程数、工作目录等等。

3）Task可以像其它普通task一样引用变量、输入输出或者File，与普通task没有任何区别。