Toil Documentation

Release 3.0.8a1

UCSC Computational Genomics Lab

December 17, 2015

Contents

See README for installation. Toil's Job class contains the Toil API, documented below. To begin, consider this short toil script:

```
from toil.job import Job
from optparse import OptionParser

class HelloWorld(Job):
    def __init__(self):
        Job.__init__(self,  memory=100000, cores=2, disk=20000)
    def run(self, fileStore):
        fileId = getEmptyFileStoreID()
        self.addChild(wrapJobFn(childFn, fileId,
                        cores=1, memory="1M", disk="10M"))
        self.addFollowOn(FollowOn(fileId),

def childFn(target, fileID):
    with target.fileStore.updateGlobalFileStream(fileID) as file:
        file.write("Hello, World!")

class FollowOn(Job):
    def __init__(self,fileId):
        Job.__init__(self)
        self.fileId=fileId
    def run(self, fileStore):
        tempDir = self.getLocalTempDir()
        tempFilePath = "/".join(tempDir,"LocalCopy")
        with readGlobalFileStream(fileId) as globalFile:
            with open(tempFilePath, w) as localFile:
               localFile.write(globalFile.read())

def main():
    parser = OptionParser()
    Job.Runner.addToilOptions(parser)
    options, args = parser.parse_args( args )
    Job.Runner.startToil(HelloWorld(),  options )

if __name__=="__main__":
    main()
```

The script consists of three Jobs - the object based HelloWorld and FollowOn Jobs, and the function based childFn Job. The object based Jobs inherit from the Job class, and must invoke the Job constructor and implement the run method, where the user's code to be executed should be placed.

Note that the constructor takes optional resource parameters, which specify the cores, memory, and disk space needed for the job to run successfully. These resources can be specified in bytes, or by passing in a string as in the constructor for the wrapJobFn().

Also notice the two types of descendant Jobs. HelloWorld specifies childFn as a child Job, and FollowOn as a follow on Job. The only difference between a parent, children, and a follow on is the order of execution. Parents are executed first, its children, finally followed by the follow ons. The children and follow on jobs are run in parallel.

CHAPTER 1

Toil Docs

Contents:

1.1 Toil API

1.1.1 Job Methods

class toil.job.**Job**(*memory=None*, *cores=None*, *disk=None*)

Represents a unit of work in toil. Jobs are composed into graphs which make up a workflow.

This public functions of this class and its nested classes are the API to toil.

addChild(*childJob*)

Adds the child job to be run as child of this job. Returns childJob. Child jobs are run after the Job.run method has completed.

See Job.checkJobGraphAcylic for formal definition of allowed forms of job graph.

addChildFn(*fn*, **args*, ***kwargs*)

Adds a child fn. See FunctionWrappingJob. Returns the new child Job.

addChildJobFn(*fn*, **args*, ***kwargs*)

Adds a child job fn. See JobFunctionWrappingJob. Returns the new child Job.

addFollowOn(*followOnJob*)

Adds a follow-on job, follow-on jobs will be run after the child jobs and their descendants have been run. Returns followOnJob.

See Job.checkJobGraphAcylic for formal definition of allowed forms of job graph.

addFollowOnFn(*fn*, **args*, ***kwargs*)

Adds a follow-on fn. See FunctionWrappingJob. Returns the new follow-on Job.

addFollowOnJobFn(*fn*, **args*, ***kwargs*)

Add a follow-on job fn. See JobFunctionWrappingJob. Returns the new follow-on Job.

addService(*service*)

Add a service of type Job.Service. The Job.Service.start() method will be called after the run method has completed but before any successors are run. It's Job.Service.stop() method will be called once the successors of the job have been run.

:rtype : An instance of PromisedJobReturnValue which will be replaced with the return value from the service.start() in any successor of the job.

checkJobGraphAcylic()
Raises a JobGraphDeadlockException exception if the connected component of jobs containing this job contains any cycles of child/followOn dependencies in the augmented job graph (see below). Such cycles are not allowed in valid job graphs. This function is run during execution.

A job B that is on a directed path of child/followOn edges from a job A in the job graph is a descendant of A, similarly A is an ancestor of B.

A follow-on edge (A, B) between two jobs A and B is equivalent to adding a child edge to B from (1) A, (2) from each child of A, and (3) from the descendants of each child of A. We call such an edge an "implied" edge. The augmented job graph is a job graph including all the implied edges.

For a job (V, E) the algorithm is O(**|V|^2). It is O(|V| + |E|**) for a graph with no follow-ons. The former follow on case could be improved!

checkJobGraphConnected()
Raises a JobGraphDeadlockException exception if getRootJobs() does not contain exactly one root job. As execution always starts from one root job, having multiple root jobs will cause a deadlock to occur.

checkJobGraphForDeadlocks()
Raises a JobGraphDeadlockException exception if the job graph is cyclic or contains multiple roots.

encapsulate()
See EncapsulatedJob.

:rtype : A new EncapsulatedJob for this job.

getRootJobs()
A root is a job with no predecessors. :rtype : set, the roots of the connected component of jobs that contains this job.

getUserScript()

run(*fileStore*)
Do user stuff here, including creating any follow on jobs.

The fileStore argument is an instance of Job.FileStore, and can be used to create temporary files which can be shared between jobs.

The return values of the function can be passed to other jobs by means of the rv() function.

Note: We disallow return values to be PromisedJobReturnValue instances (generated by the Job.rv() function - see below). A check is made that will result in a runtime error if you attempt to do this. Allowing PromisedJobReturnValue instances to be returned does not work because the mechanism to pass the promise uses a jobStoreFileID that will be deleted once the current job and its descendants have been completed. This is similar to scope rules in a language like C, where returning a reference to memory allocated on the stack within a function will produce an undefined reference. Disallowing this also avoids nested promises (PromisedJobReturnValue instances that contain other PromisedJobReturnValue).

rv(*argIndex=None*)
Gets a PromisedJobReturnValue, representing the argIndex return value of the run function (see run method for description). This PromisedJobReturnValue, if a class attribute of a Job instance, call it T, will be replaced by the actual return value when the T is loaded. The function rv therefore allows the output from one Job to be wired as input to another Job before either is actually run.

Parameters **argIndex** – If None the complete return value will be returned, if argIndex

is an integer it is used to refer to the return value as indexable (tuple/list/dictionary, or in general object that implements __getitem__), hence rv(i) would refer to the ith (indexed from 0) member of return value.

static wrapFn(*fn*, **args*, ***kwargs*)
Makes a Job out of a function.

Convenience function for constructor of FunctionWrappingJob

static **wrapJobFn**(*fn*, **args*, ***kwargs*)
Makes a Job out of a job function.

Convenience function for constructor of JobFunctionWrappingJob

1.1.2 Job.FileStore

The FileStore is an abstraction of a Toil run's shared storage

class Job.**FileStore**(*jobStore*, *jobWrapper*, *localTempDir*)
Class used to manage temporary files and log messages, passed as argument to the Job.run method.

deleteGlobalFile(*fileStoreID*)
Deletes a global file with the given fileStoreID. Returns true if file exists, else false.

getEmptyFileStoreID()
Returns the ID of a new, empty file.

getLocalTempDir()
Get the local temporary directory. This directory will exist for the duration of the job only, and is guaranteed to be deleted once the job terminates.

globalFileExists(*fileStoreID*)
:rtype : True if and only if the jobStore contains the given fileStoreID, else false.

logToMaster(*string*)
Send a logging message to the leader. Will only ne reported if logging is set to INFO level (or lower) in the leader.

readGlobalFile(*fileStoreID*, *localFilePath=None*)
Returns a path to a local copy of the file keyed by fileStoreID. The version will be consistent with the last copy of the file written/updated to the global file store. If localFilePath is not None, the returned file path will be localFilePath.

readGlobalFileStream(*fileStoreID*)
Similar to readGlobalFile, but returns a context manager yielding a file handle which can be read from. The yielded file handle does not need to and should not be closed explicitly.

updateGlobalFile(*fileStoreID*, *localFileName*)
Replaces the existing version of a file in the global file store, keyed by the fileStoreID. Throws an exception if the file does not exist.

updateGlobalFileStream(*fileStoreID*)
Similar to updateGlobalFile, but returns a context manager yielding a file handle which can be written to. The yielded file handle does not need to and should not be closed explicitly.

writeGlobalFile(*localFileName*)
Takes a file (as a path) and uploads it to to the global file store, returns an ID that can be used to retrieve the file.

writeGlobalFileStream()
Similar to writeGlobalFile, but returns a context manager yielding a tuple of 1) a file handle which can be written to and 2) the ID of the resulting file in the job store. The yielded file handle does not need to and should not be closed explicitly.

1.1.3 Job.Runner

The Runner contains the methods needed to configure and start a Toil run.

class Job.**Runner**

Used to setup and run a graph of jobs.

static addToilOptions(*parser*)

Adds the default toil options to an optparse or argparse parser object.

static getDefaultOptions(*jobStore*)

Returns an optparse.Values object of the options used by a toil.

static startToil(*job*, *options*)

Runs the toil workflow using the given options (see Job.Runner.getDefaultOptions and Job.Runner.addToilOptions) starting with this job.

raises toil.leader.FailedJobsException if at the end of function their remain

failed jobs

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

A

C

D

E

G

J

L

R

S

U

W