Arvados - Feature #11015

Improve throughput of crunch-run output-uploading stage using multi-threaded transfers

01/31/2017 07:42 PM - Tom Clegg

Status:	Resolved	**Start date:**	01/31/2017
Priority:	Normal	**Due date:**	
Assigned To:	Radhika Chippada	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	2017-03-01 sprint		

Description

To improve throughput of crunch-run job output uploading, add support for multi-threaded asynchronous transfers to hide the latency inherent in cloud environments.

Refactoring to support public APIs in the Go SDK is a separate task.

Subtasks:

Task # 11108: Review 11015-crunch-run-output-upload	**Resolved**

Associated revisions

Revision 0b125c52 - 02/28/2017 06:29 PM - Radhika Chippada

closes #11015
Merge branch '11015-crunch-run-output-upload'

History

#1 - 01/31/2017 09:21 PM - Tom Morris

- Description updated

#2 - 02/15/2017 08:42 PM - Tom Morris

- Target version set to Arvados Future Sprints

#3 - 02/15/2017 09:15 PM - Tom Morris

- Target version changed from Arvados Future Sprints to 2017-03-01 sprint

#4 - 02/15/2017 09:19 PM - Tom Clegg

- Description updated

#5 - 02/15/2017 09:20 PM - Tom Morris

- Subject changed from Improve throughput of crunch-run output-uploading stage using multithreaded transfers to Improve throughput of crunch-run output-uploading stage using multi-threaded transfers

- Description updated

- Assigned To set to Radhika Chippada

- Story points set to 2.0

#6 - 02/21/2017 04:29 PM - Tom Clegg

Possible implementation

In (*CollectionFileWriter)goUpload(), currently we have

```
goUpload() {
  for block := range uploader {
    compute md5
    write to keep
    append signed locator to m.ManifestStream.Blocks
    (or append err to errors)
  }
}
```

We can add a buffered channel to CollectionWriter and pass it to (CFW)goUpload() and use it to limit the total number of blocks being written by all

CFWs of a given CollectionWriter.

```
func (m *CollectionWriter) Open(path string) io.WriteCloser {
  ...
  m.mtx.Lock()
  defer m.mtx.Unlock()
  if m.workers == nil {
    if m.MaxWriters < 1 {
      m.MaxWriters = 1
    }
    m.workers = make(chan struct{}, m.MaxWriters)
  }
  go fw.goUpload(m.workers)
  m.Streams = append(m.Streams, fw)
  ...
}

goUpload(workers chan struct{}) {
  var blocks []string
  var mtx sync.Mutex
  for block := range uploader {
    mtx.Lock()
    blockIndex := len(blocks)
    blocks = append(blocks, "")
    mtx.Unlock()

    workers <- struct{}{}    // wait for an available worker slot
    wg.Add(1)

    go func(blockIndex int) {
      compute md5
      write to keep
      <-workers              // release worker slot

      mtx.Lock()
      m.ManifestStream.Blocks[blockIndex] = signedLocator
      (or append err to errors)
      mtx.Unlock()

      wg.Done()
    }(blockIndex)
  }
  wg.Wait()
  finish <- errors
}
```

#7 - 02/22/2017 03:38 PM - Radhika Chippada

- Status changed from New to In Progress

#8 - 02/28/2017 04:44 PM - Lucas Di Pentima

Sorry for taking so long to send comments, was trying to understand the code the best I could.
One question: Are the upload workers number configurable? I'm not able to see where, it seems to be hardcoded to be 2.

I've run services/crunch-run tests locally without issues.

#9 - 02/28/2017 06:18 PM - Radhika Chippada

> Are the upload workers number configurable?

Number of writers >1 will help us as this let's us use multiple threads to upload.

Thanks.

#10 - 02/28/2017 06:35 PM - Radhika Chippada

- Status changed from In Progress to Resolved

- % Done changed from 0 to 100

Applied in changeset arvados|commit:0b125c52cd816e6a4120c414d3817f354cad1055.