MarkLogic IO Testing

5/13/2014

Table of Contents

MarkLogic IO Testing

1 Introduction

The purpose of this document is to explain how to use the MarkLogic IO Test application, available at https://github.com/mustard57/MarkLogic-IO-Test.

This application allows you to benchmark the write capability of your MarkLogic system. It allows you to study the relative effects of different configuration parameters. As it can be run in a deterministic fashion, you can compare the efficiency of your system to other MarkLogic systems.

The purpose of the application is to allow the user insight into the write behaviour of MarkLogic in response to modification of configuration options. It allows the user to optimize those settings. It potentially allows the user to answer the question 'How good is my system?' by running the same tests on different systems.

The approach in this document is to give the reader examples of usage, rather than to list the functions of the application.

2 Setup

'MarkLogic IO Test' is roxy based - details available at https://github.com/marklogic/roxy. To start using the MarkLogic IO Test application, do

```
git clone https://github.com/mustard57/MarkLogic-IO-Test.
```

In build.properties set your administrator credentials - if you wish to use a user other than admin set

```
user=<required-user-name>
```

You can store your password in this file using

```
password=<admin-user-password>
```

If you do not do this, you will be prompted for the password at the command line.

From the project directory

```
ml local bootstrap

ml local deploy modules

ml local deploy content ( required )
```

will deploy to your local machine on port 8040.

To install on another machine, choose an alias e.g. myalias

In deploy/default.properties, alter the line starting with environments to

```
environments=local,dev,prod,myalias
```

Add your machine name to build.properties - put something like myalias-server=mymachinename.mydomain.com

at the foot of the file.

If you want to run on a different port, add a file named myalias.properties to the deploy directory, and add

app-port=<my-port-number>[1]

xcc-port=<myp-port-number-2>[2]

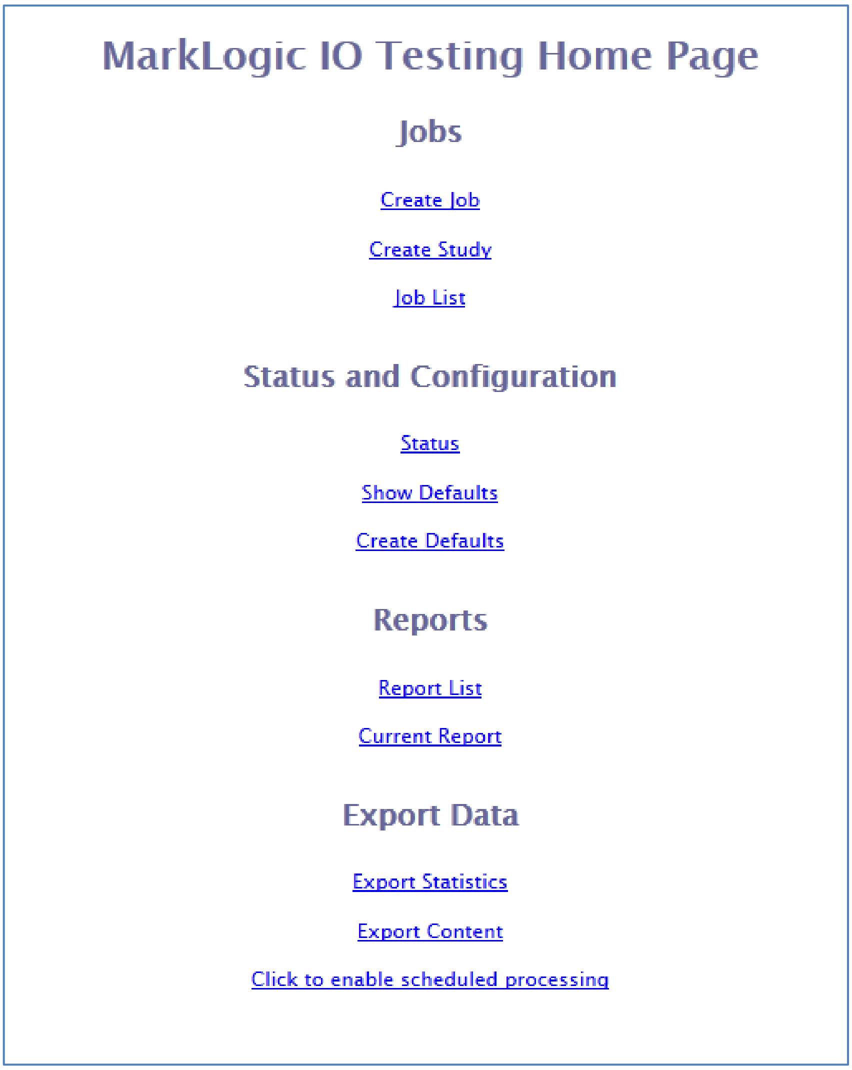
Also, if you have different access credentials, add

user=<required-user-name>

password=<admin-user-password>

as appropriate to this file.

If you go to http://localhost:8040[3] you should then see



MarkLogic IO Testing Home Page

Jobs

Create Job

Create Study

Job List

Status and Configuration

Status

Show Defaults

Create Defaults

Reports

Report List

Current Report

Export Data

Export Statistics

Export Content

Click to enable scheduled processing

[1] HTTP application port

[2] XDBC port - used by deploy modules

[3] Or http://mymachinename.mydomain.com:my-port-number as appropriate

3 Testing Approach

The basic approach is to ingest a user defined number of documents and measure the duration of that run, as well as io related statistics such as save-write. Ingestion can be controlled in terms of the batching of records, or the rate at which the requests are made.

Ingestion will be made to a dynamically created database, names Sample, whose properties are in part set in accordance with test run parameters.

Ingestion is accomplished by creating jobs and placing these on the task server. This approach allows the batching and request rate to be controlled.

3.1 Generate and Save Mode

A problem often encountered in testing is the sourcing of test data. The application can help you with this.

If tests are run using ‘generate and save’ mode (more later) the application will generate test data for you. It does this by generating text files, whose average size is defined by the user during the test run. The text files are generated by randomly selecting words from a content document[4] which has the virtue of generating representative word distributions. The selection stops when the size of the document exceeds the average requested.

3.2 Load From Disk Mode

Alternatively, content can be loaded from a named directory. This has the advantage of not requiring the computational overhead above[5]. Additionally, the content generated by ‘generate and save’ can be saved to a directory, allowing the test application to be fully self-sufficient, while avoiding computational overhead.

4 Default Values

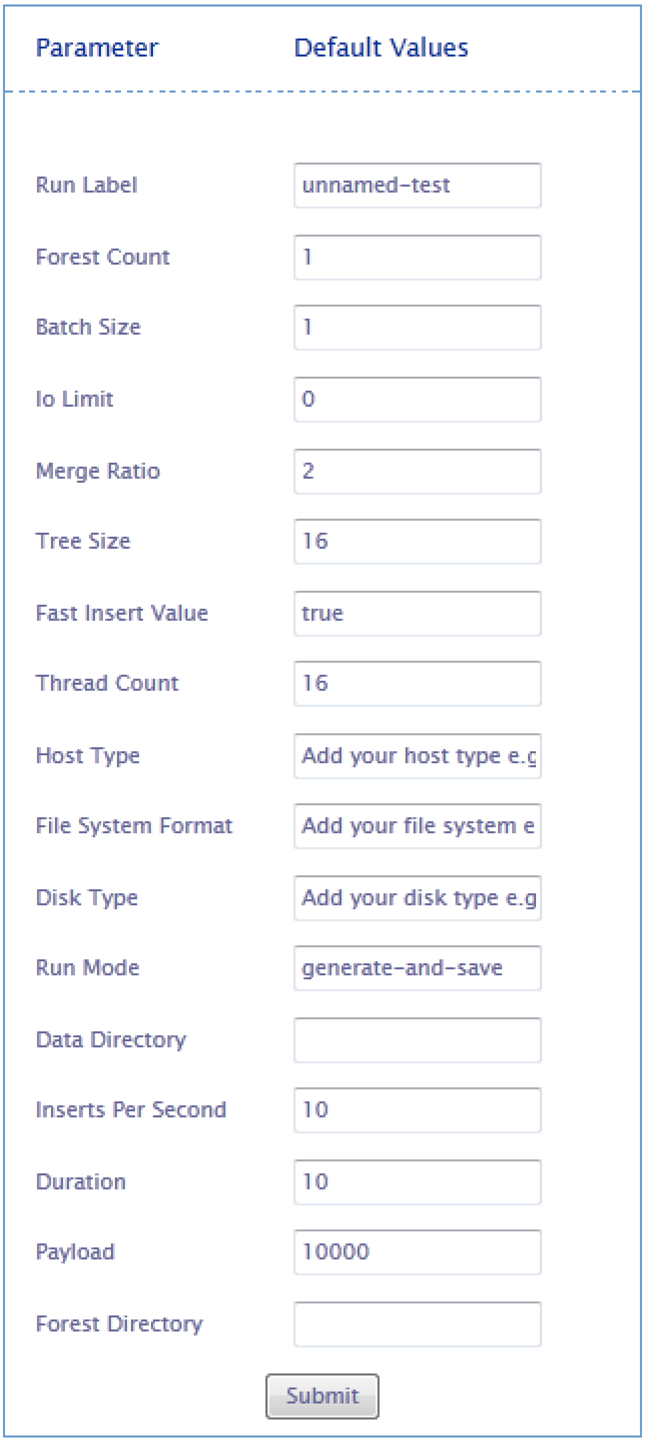
There are a significant number of values that need to be set when using the IO test application. To make things easier default values can be set. Default values can be seen by selecting ‘Show Defaults’ from the application home page, and set via ‘Create Defaults’ – screenshot below. All these values can be over-ridden by specific tests.

It makes sense to set these values before proceeding. They can always be modified.

The screenshot below shows the default default values.

[4] By default, /data/on-liberty.txt in the Git repository. This is John Stuart Mill’s famous ‘On Liberty’ essay, approximately 48,000 words. You can replace this with a document of your own chosing – make sure you change $constants:SOURCE-DOCUMENT if needed

[5] Note that in ‘generate and save’ mode, caching is used to minimize computational overhead



Parameter	Default Values
Run Label	unnamed-test
Forest Count	1
Batch Size	1
Io Limit	0
Merge Ratio	2
Tree Size	16
Fast Insert Value	true
Thread Count	16
Host Type	Add your host type e.g
File System Format	Add your file system e
Disk Type	Add your disk type e.g
Run Mode	generate-and-save
Data Directory	
Inserts Per Second	10
Duration	10
Payload	10000
Forest Directory	

In this section, the various values are explained.

4.1 Run Label

Tests can be grouped. The run-label parameter determines what group a test gets placed in. Grouping is made use of when reporting results – all tests with the same run label are shown together. Set the default run label value to something sensible, e.g. your host name or host type.

4.2 Forest Count

Default number of forests used for the test database.

4.3 Batch Size

Default number of documents per transaction.

4.4 IO Limit

Default value for the background io limit (see http://docs.marklogic.com/admin:group-set-background-io-limit)

4.5 Merge Ratio

Default value for merge min ratio (see http://docs.marklogic.com/admin:database-set-merge-min-ratio), which contributes to merge decisions.

4.6 Tree Size

Default value for in memory tree size (see http://docs.marklogic.com/admin:database-set-in-memory-tree-size) which determines how frequently in memory stands are written to disk.

4.7 Fast Insert Value

Whether to use 'in forest placement'. If set to true, existence of the URI being written to always assumed false, removing the need to check for existence which slightly reduces insertion rate.

4.8 Thread Count

No of task server threads. This value determines the extent to which your ingestion is parallelized. The maximum useful value will be a function of the number of cores available, but not necessarily equal to this number.

4.9 Host Type

This text field does not influence io tests, but it is useful to label your results with the host type, to aid comparison, and also as an aide-memoire. Set this value before proceeding.

4.10 File System Format

Similar to 4.9, this is a text value, but usefully labels your data if you are testing different file system types e.g. ext3,ext4. Set this value before proceeding.

4.11 Disk Type

Again, similar to 4.9, use this to record the disk type being used. For instance, if working on AWS you may wish to record whether you are using standard EBS, or PIOPS.

4.12 Run Mode

As discussed above, permitted values are generate-and-save and load-from-disk.

4.13 Data Directory

If running in load-from-disk-mode, the directory content will be loaded from this directory. Note that it may make sense to place this on a file system separate from the forest directory – see 4.17

4.14 Inserts per second

Controls the rate at which jobs are placed on the task server. If set to a high value, this effectively will cause the process to run as fast as possible

4.15 Duration

Controls the length and size of the test. Total number of documents inserted will be duration * inserts per second. If inserts per second exceeds the maximum available processing rate, the actual duration will exceed the requested duration.

4.16 Payload

Average payload size – used if running in generate-and-save mode – see 3.1.

4.17 Forest Directory

Directory under which the test database forests will be placed.

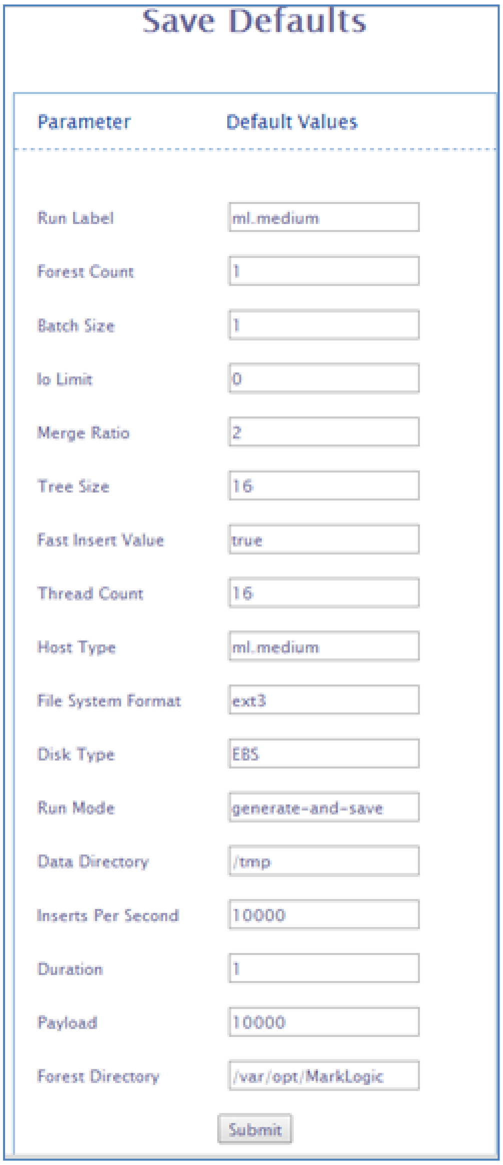
5 Running your First Test

5.1 Execution

I am going to execute a ‘generate and save’ test. I would like to run using 10k * 10kb documents, as fast as possible.

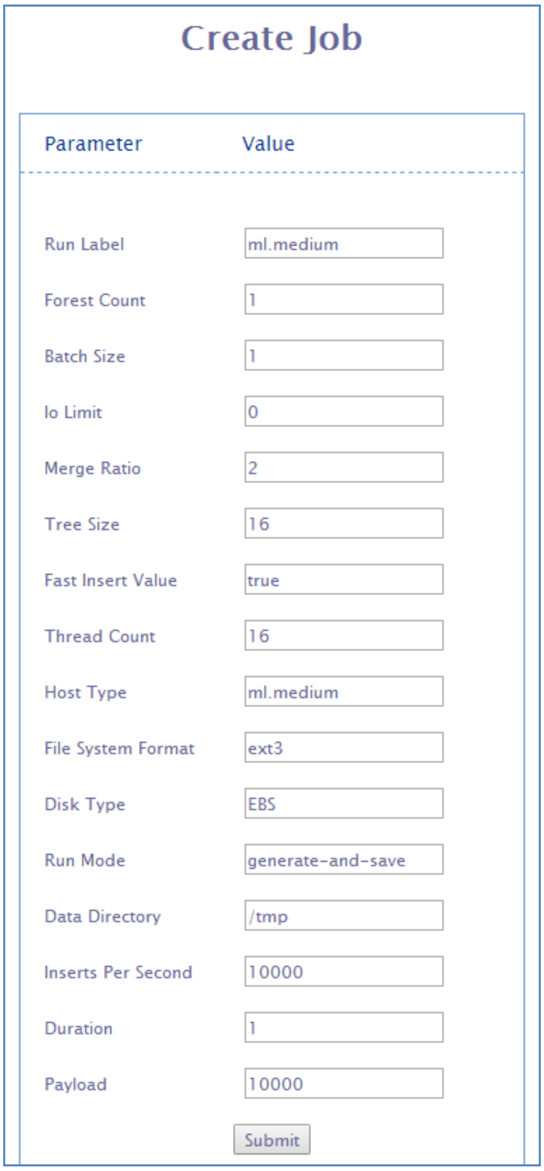
First of all, I set my default values. I will be running on an AWS ml.medium, so I set this as my run label. I will be running using the standard EBS storage type, so Disk Type is set to EBS. The disk is formatted as ext3 so this is the file system format value. The run mode is generate-and-save. I want 10000 10k updates as quickly as possible so set inserts per second = 10000, duration = 1 and payload = 10000.

Next, from the Home page, select ‘Create Job’. You will see the values populated as per the defaults. We’re not going to change anything here. Click Submit.



Save Defaults

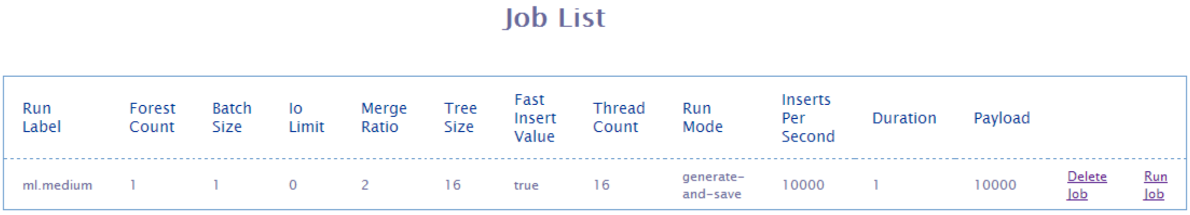
Parameter	Default Values
Run Label	ml.medium
Forest Count	1
Batch Size	1
Io Limit	0
Merge Ratio	2
Tree Size	16
Fast Insert Value	true
Thread Count	16
Host Type	ml.medium
File System Format	ext3
Disk Type	EBS
Run Mode	generate-and-save
Data Directory	/tmp
Inserts Per Second	10000
Duration	1
Payload	10000
Forest Directory	/var/opt/MarkLogic



Create Job

Parameter	Value
Run Label	ml.medium
Forest Count	1
Batch Size	1
Io Limit	0
Merge Ratio	2
Tree Size	16
Fast Insert Value	true
Thread Count	16
Host Type	ml.medium
File System Format	ext3
Disk Type	EBS
Run Mode	generate-and-save
Data Directory	/tmp
Inserts Per Second	10000
Duration	1
Payload	10000

You will be taken to the Job List page. Click ‘Run job’.[6]



Job List

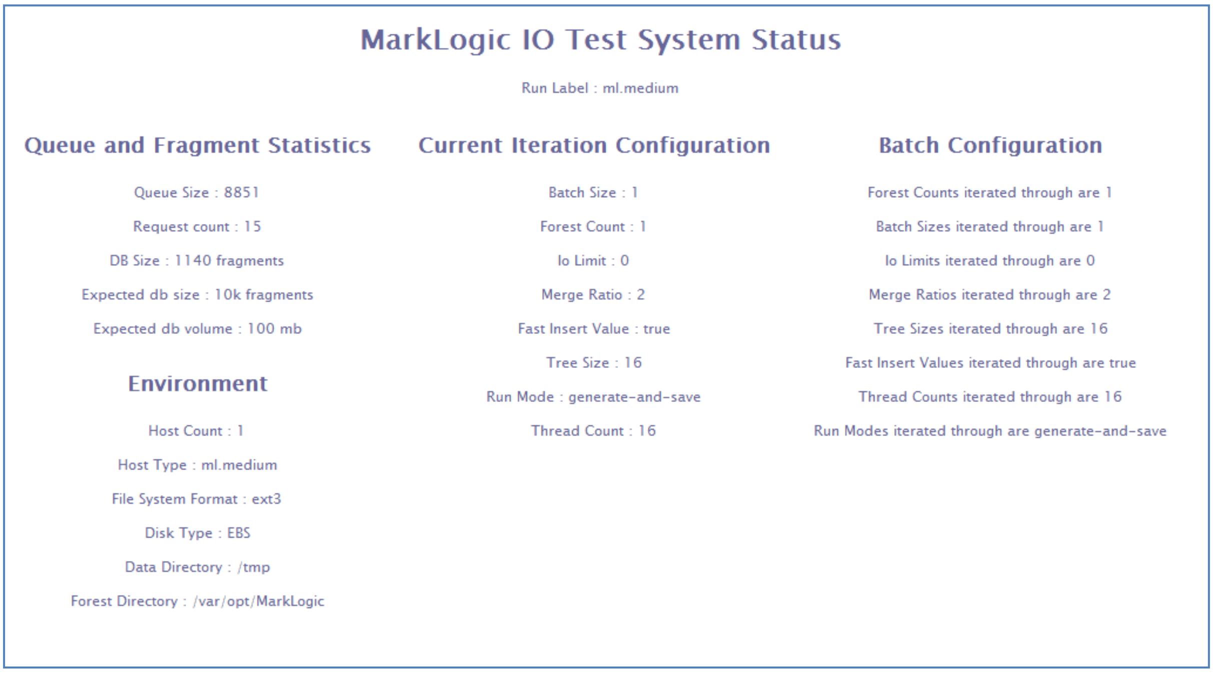
Run Label	Forest Count	Batch Size	Io Limit	Merge Ratio	Tree Size	Fast Insert Value	Thread Count	Run Mode	Inserts Per Second	Duration	Payload		
ml.medium	1	1	0	2	16	true	16	generate-and-save	10000	1	10000	Delete Job	Run Job

You will briefly see a page saying ‘Your job has been spawned’, before being taken back to the ‘Job List’ page. Best place to go now is the ‘Status’ page – link bottom left.

The screenshot below shows a sample ‘Status’ output. On the left hand side are details of your current job. We see there are 8851 insert jobs on the queue, with 1140 having been completed

[6] This may seem redundant, but you can create jobs while other jobs are running. You can also automate running of jobs – more on this in section 10.

already[7]. The Environment is as per our default settings. Iteration configuration is not relevant at this point as we are not running a job with multiple iterations. This aspect of operation will be discussed in section 9.



MarkLogic IO Test System Status

Run Label : ml.medium

Queue and Fragment Statistics

Queue Size : 8851

Request count : 15

DB Size : 1140 fragments

Expected db size : 10k fragments

Expected db volume : 100 mb

Environment

Host Count : 1

Host Type : ml.medium

File System Format : ext3

Disk Type : EBS

Data Directory : /tmp

Forest Directory : /var/opt/MarkLogic

Current Iteration Configuration

Batch Size : 1

Forest Count : 1

Io Limit : 0

Merge Ratio : 2

Fast Insert Value : true

Tree Size : 16

Run Mode : generate-and-save

Thread Count : 16

Batch Configuration

Forest Counts iterated through are 1

Batch Sizes iterated through are 1

Io Limits iterated through are 0

Merge Ratios iterated through are 2

Tree Sizes iterated through are 16

Fast Insert Values iterated through are true

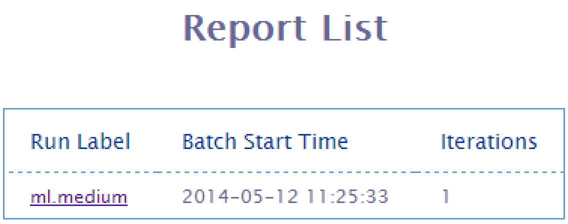
Thread Counts iterated through are 16

Run Modes iterated through are generate-and-save

At completion, queue size will be zero, and DB size will be equal to expected DB size. At this point we can look at our first report.

5.2 Reporting

Go to 'Report List' from the bottom of the 'Status' page (or from the Home page). Your screen will look something like



Report List

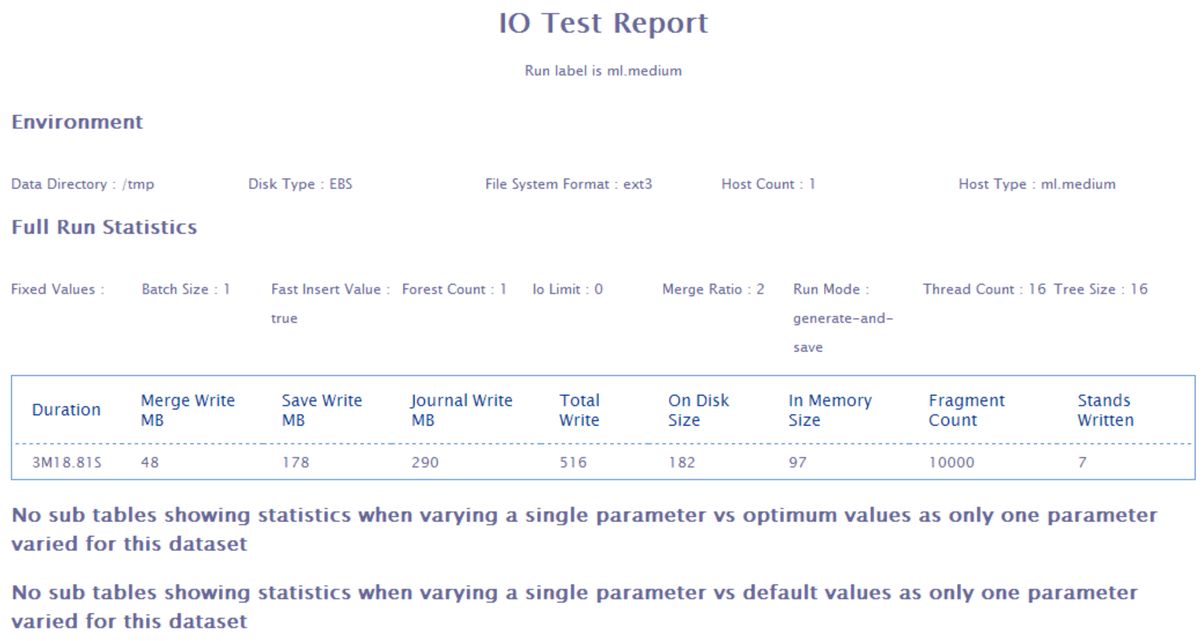
Run Label	Batch Start Time	Iterations
ml.medium	2014-05-12 11:25:33	1

Click on 'ml.medium'.

Our first report is below. All the key input parameters are shown. The run statistics are duration and a number of IO statistics. The text reading 'No sub tables...' applies as our test run has not varied any input parameters.[8]

[7] This job will require 10,000 jobs, as batch size is 1. The status screen shows that 1140 of these have already completed, and 8851 are remaining. The effect of different batch sizes can be seen in section 9

[8] See section 7 for more details.



IO Test Report

Run label is ml.medium

Environment

Data Directory : /tmp Disk Type : EBS File System Format : ext3 Host Count : 1 Host Type : ml.medium

Full Run Statistics

Fixed Values : Batch Size : 1 Fast Insert Value : true Forest Count : 1 Io Limit : 0 Merge Ratio : 2 Run Mode : generate-and-save Thread Count : 16 Tree Size : 16

Duration	Merge Write MB	Save Write MB	Journal Write MB	Total Write	On Disk Size	In Memory Size	Fragment Count	Stands Written
3M18.81S	48	178	290	516	182	97	10000	7

No sub tables showing statistics when varying a single parameter vs optimum values as only one parameter varied for this dataset

No sub tables showing statistics when varying a single parameter vs default values as only one parameter varied for this dataset

6 Making a Comparison

The results above, though possibly interesting in themselves, are more useful when they are relative comparisons.

To that end, in this section we start with setting up a test that compares ingestion using two different file system types – ext3 and ext4.

First of all, format the device in question[9]

```
mkfs.ext3 -m 0 -f <YOUR_DEVICE_NAME>[10]
```

Next create a mount point, set the permissions so that it can be used by MarkLogic, and mount.

```
mkdir <YOUR_POINT_POINT>[11]
mount –o noatime <YOUR_DEVICE_NAME> <YOUR_MOUNT_POINT>[12]
chmod 777 <YOUR_MOUNT_POINT>
```

Next, set the Forest Directory to be <YOUR_MOUNT_POINT> in the default values.

Then run job as before, but setting the Run Label to something meaningful – I will use ‘file-system-compare’.

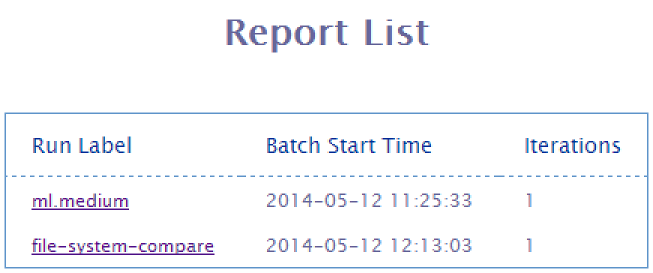
Once the job is complete, the report list page should look something like the image below.

[9] In my test, I added an EBS device on /dev/xvdm

[10] The -m flag sets the amount of super-user only space

[11] I will use /space

[12] The -o noatime means read access to files is not recorded, this would otherwise degrade performance



Report List

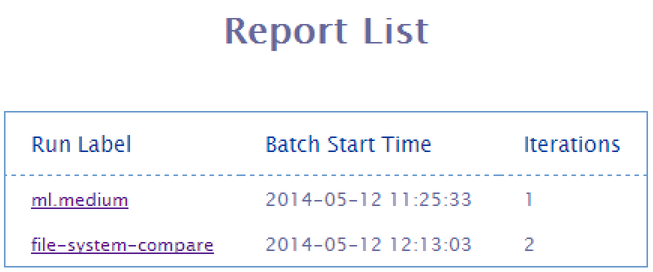
Run Label	Batch Start Time	Iterations
ml.medium	2014-05-12 11:25:33	1
file-system-compare	2014-05-12 12:13:03	1

The next stage is to run the same test, but with an ext4 file system format. To do that

```
/etc/init.d/MarkLogic stop ( you cannot unmount unless you do this )
umount /space
mkfs.ext4 -m 0 -f <YOUR_DEVICE_NAME>
mount -o noatime <YOUR_DEVICE_NAME> <YOUR_MOUNT_POINT>
chmod 777 <YOUR_MOUNT_POINT>
/etc/init.d/MarkLogic start
```

Next create your job, but set file system format to ext4, and run label to ‘file-system-compare’. This second instruction makes sure that this test is grouped with the previous one for reporting purposes.

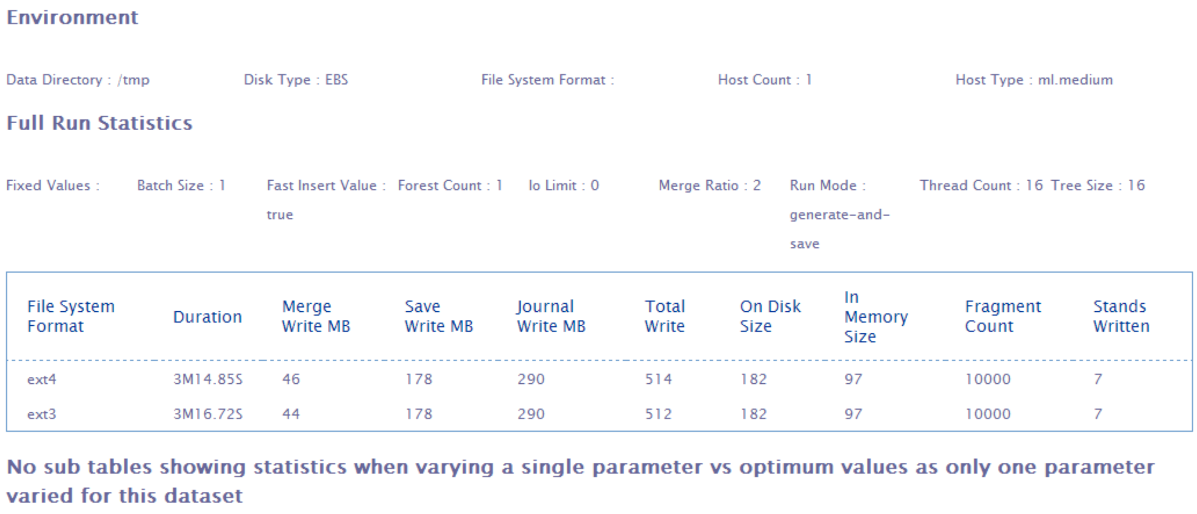
When the job has finished, the report list page looks like



Report List

Run Label	Batch Start Time	Iterations
ml.medium	2014-05-12 11:25:33	1
file-system-compare	2014-05-12 12:13:03	2

Note there are now two iterations for the file-system-compare report. If we click on this, we see



Environment

Data Directory : /tmp   Disk Type : EBS   File System Format :   Host Count : 1   Host Type : ml.medium

Full Run Statistics

Fixed Values :   Batch Size : 1   Fast Insert Value : true   Forest Count : 1   Io Limit : 0   Merge Ratio : 2   Run Mode : generate-and-save   Thread Count : 16   Tree Size : 16

File System Format	Duration	Merge Write MB	Save Write MB	Journal Write MB	Total Write	On Disk Size	In Memory Size	Fragment Count	Stands Written
ext4	3M14.85S	46	178	290	514	182	97	10000	7
ext3	3M16.72S	44	178	290	512	182	97	10000	7

No sub tables showing statistics when varying a single parameter vs optimum values as only one parameter varied for this dataset

This is our first comparison. It shows a marginal improvement for the ext4 file system, though the scale of the test means the results are not meaningful. In general, you should probably look to have your test running for at least 30min to eliminate variance due to extraneous factors.

Also note that as we are only varying a single parameter (file system type) to at this stage, no sub-tables are showing.

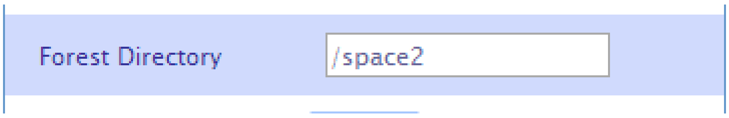
7 More Comparisons

To take the above idea a little further, we repeat the tests above using a different disk type. You could look to see what the effects are of using an SSD vs local disk, or of using disk sets with different RAID configurations. Here, use of an AWS PIOPS (Prioritized IOPS) disk is compared with normal EBS.

My new device is on /dev/xvdo. I am going to mount it on /space2.

```
mkfs.ext3 -m 0 -f /dev/xvdo
mkdir /space2
mount -o noatime /dev/xvdo /space2
chmod 777 /space2
```

Next in my defaults page I change ‘Forest Directory’ to /space2.



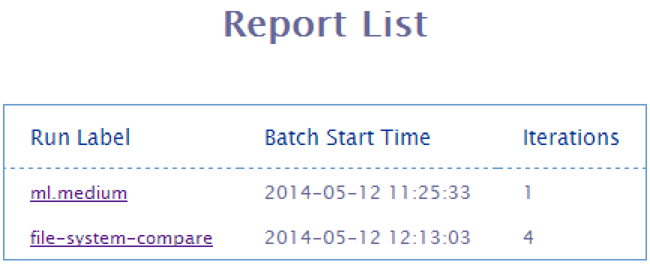
Forest Directory: /space2

Now I create my job, but with Run Label set to ‘file-system-compare’, disk-type = PIOPS-4000[13], and file system format set to ext3, and run.

Next, I reformat my drive as ext4, and run the test again.

```
/etc/init.d/MarkLogic stop
umount /space2
mkfs.ext4 -m 0 -f /dev/xvdo
mount -o noatime /dev/xvdo /space2
chmod 777 /space2
/etc/init.d/MarkLogic start
```

The screenshot below shows that I now have four iterations of tests with the ‘file-system-compare’ run label.



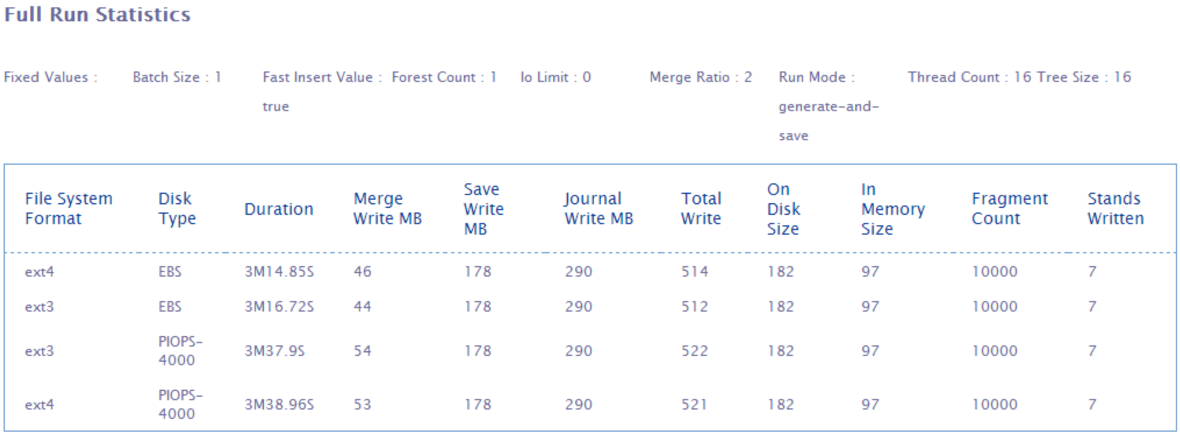
Report List

Run Label	Batch Start Time	Iterations
ml.medium	2014-05-12 11:25:33	1
file-system-compare	2014-05-12 12:13:03	4

Screenshots below show the file-system-compare report.

The first screenshot shows the four individual test iterations, sorted by duration. You should not read too much into the actual numbers due to the short duration, but if the duration was extended, you would be able to genuinely see the impact of choice of different file systems and disk types.

[13] The prioritized IOPS count for my device.

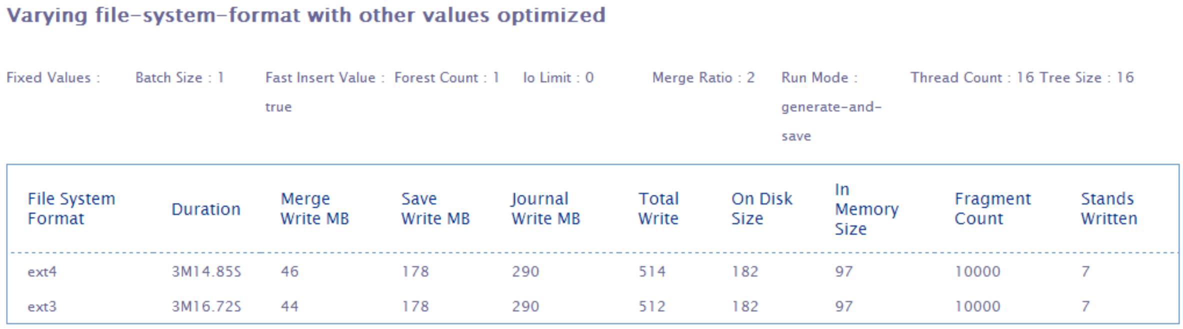


Full Run Statistics

Fixed Values :	Batch Size : 1	Fast Insert Value : true	Forest Count : 1	Io Limit : 0	Merge Ratio : 2	Run Mode : generate-and-save	Thread Count : 16 Tree Size : 16

File System Format	Disk Type	Duration	Merge Write MB	Save Write MB	Journal Write MB	Total Write	On Disk Size	In Memory Size	Fragment Count	Stands Written
ext4	EBS	3M14.85S	46	178	290	514	182	97	10000	7
ext3	EBS	3M16.72S	44	178	290	512	182	97	10000	7
ext3	PIOPS-4000	3M37.9S	54	178	290	522	182	97	10000	7
ext4	PIOPS-4000	3M38.96S	53	178	290	521	182	97	10000	7

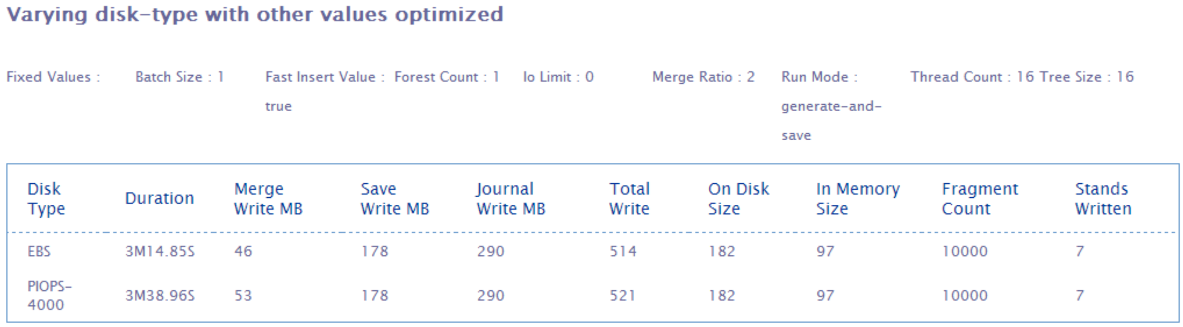
Next, the application will take the optimal configuration (which in this case is ext4 and EBS) and show what happens if just one of these variables is changed, resulting in the tables below.



Varying file-system-format with other values optimized

Fixed Values :	Batch Size : 1	Fast Insert Value : true	Forest Count : 1	Io Limit : 0	Merge Ratio : 2	Run Mode : generate-and-save	Thread Count : 16 Tree Size : 16

File System Format	Duration	Merge Write MB	Save Write MB	Journal Write MB	Total Write	On Disk Size	In Memory Size	Fragment Count	Stands Written
ext4	3M14.85S	46	178	290	514	182	97	10000	7
ext3	3M16.72S	44	178	290	512	182	97	10000	7



Varying disk-type with other values optimized

Fixed Values :	Batch Size : 1	Fast Insert Value : true	Forest Count : 1	Io Limit : 0	Merge Ratio : 2	Run Mode : generate-and-save	Thread Count : 16 Tree Size : 16

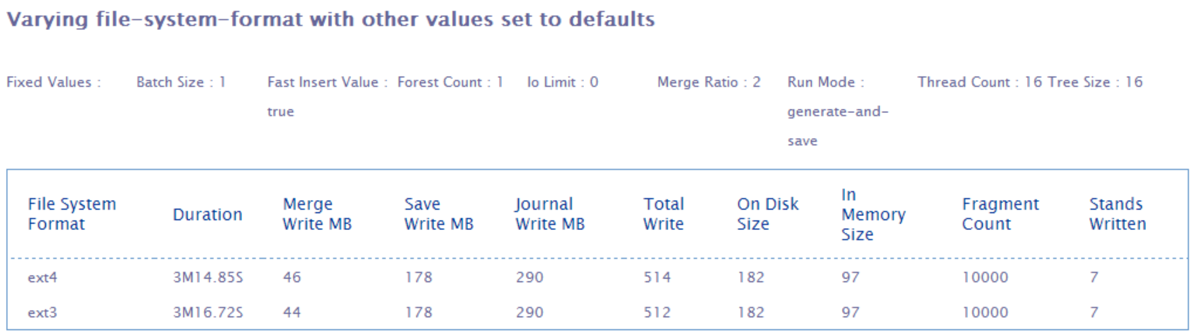
Disk Type	Duration	Merge Write MB	Save Write MB	Journal Write MB	Total Write	On Disk Size	In Memory Size	Fragment Count	Stands Written
EBS	3M14.85S	46	178	290	514	182	97	10000	7
PIOPS-4000	3M38.96S	53	178	290	521	182	97	10000	7

The first table therefore shows EBS vs ext3 and ext4 , while the second shows ext4 vs EBS and PIOPS.

Finally the report shows what happens if we take our defaults (EBS and ext3) and change just one variable.

The first table therefore shows EBS vs ext3 and ext4, while the second shows ext3 vs EBS and PIOPS.

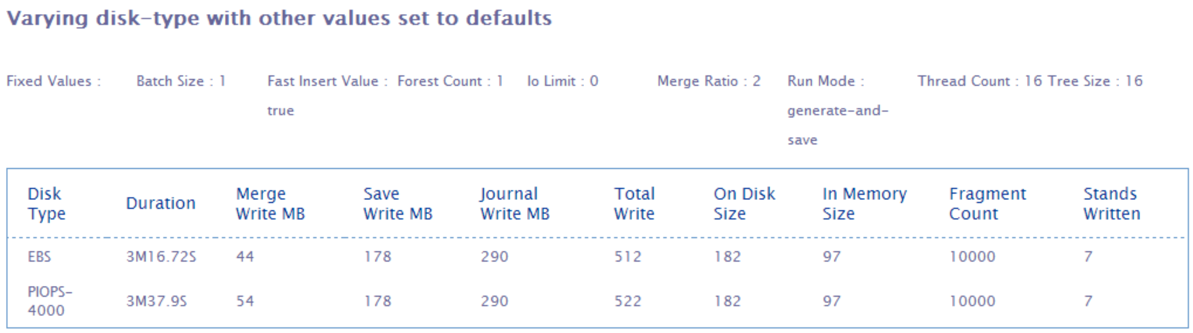
Note that this analysis is not limited to two variables - if we were varying more than two variables, we would get, for each variable, a table showing how performance varies when just one of the optimal variables is changed, and another showing how performance varies when just one of the default variables is changed. A later example will make this clearer.



Varying file-system-format with other values set to defaults

Fixed Values :	Batch Size : 1	Fast Insert Value : true	Forest Count : 1	Io Limit : 0	Merge Ratio : 2	Run Mode : generate-and-save	Thread Count : 16	Tree Size : 16

File System Format	Duration	Merge Write MB	Save Write MB	Journal Write MB	Total Write	On Disk Size	In Memory Size	Fragment Count	Stands Written
ext4	3M14.85S	46	178	290	514	182	97	10000	7
ext3	3M16.72S	44	178	290	512	182	97	10000	7



Varying disk-type with other values set to defaults

Fixed Values :	Batch Size : 1	Fast Insert Value : true	Forest Count : 1	Io Limit : 0	Merge Ratio : 2	Run Mode : generate-and-save	Thread Count : 16	Tree Size : 16

Disk Type	Duration	Merge Write MB	Save Write MB	Journal Write MB	Total Write	On Disk Size	In Memory Size	Fragment Count	Stands Written
EBS	3M16.72S	44	178	290	512	182	97	10000	7
PIOPS-4000	3M37.9S	54	178	290	522	182	97	10000	7

8 Load From Disk

One possible criticism of the generate-and-save mode is that it is not a true IO test as a substantial amount of CPU is used.

To avoid this, the load-from-disk method is available. In essence, this is quite simple – the application simply loads all content from a given directory. It does this by creating document-count / batch-size jobs, and placing on the task server. The job creation rate can be throttled by the 'inserts per second' parameter.

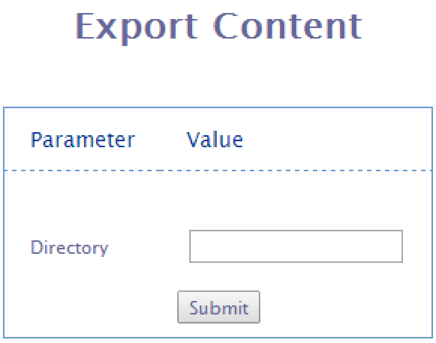
A further enhancement is to allow the content from the generate-and-save method to be saved directly to disk. This makes the application completely self sufficient and gets round the problem of sourcing test data, while avoiding CPU overhead.

In this section, we run two tests to demonstrate. First of all we run the test in section 5 to generate the data. If you are reproducing the steps in this document, not that in section 7 we set Forest Directory to /space2. I will be moving back to my original disk, /space.

Set up a job as before, but with a new run label. I will use compare-modes. Leave run mode as generate-and-save. Run the test. In 'Report List' you should have a report with the above name, with one iteration.

The above test results in creation of a test database with 10k * 10kb documents. The application will let you export these by, from the home page, selecting 'Export Content'. You will be prompted for a directory to export to.[14]

[14] It is probably a good idea to put this on a file system different to the forest directory file system, to avoid contention between the two when loading data. I put my /data directory on a separate device.



Export Content

Parameter	Value
Directory	

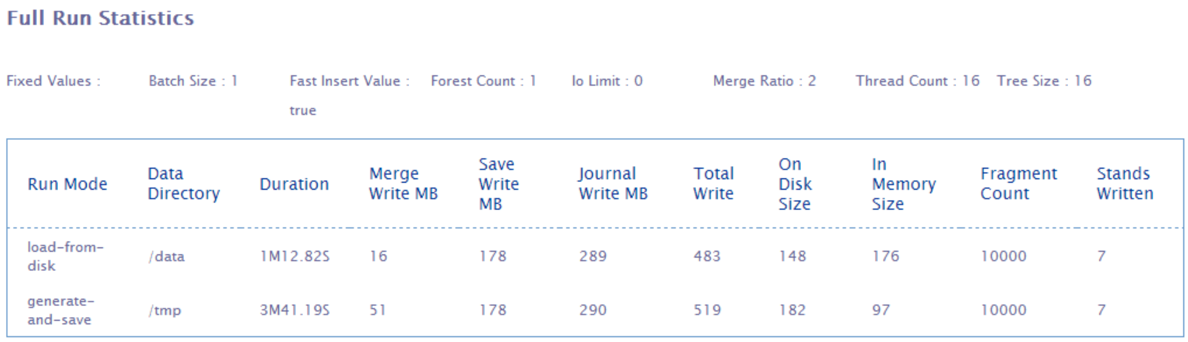
If you do ls -l <DIRECTORY>/*.txt | wc -l[15], you will be able to tell when the process has finished.

We will now use this data to perform a load-from-disk test.

First of all, in the defaults, set ‘Data Directory’ equal to the directory you have just written to.

Next, create a job with ‘Run Mode’ = load-from-disk. Set your ‘Run Label’ so it matches the previous test e.g. ‘compare-modes’. Run.

I get the following for my compare-modes report.



Full Run Statistics

Fixed Values : Batch Size : 1 Fast Insert Value : true Forest Count : 1 Io Limit : 0 Merge Ratio : 2 Thread Count : 16 Tree Size : 16

Run Mode	Data Directory	Duration	Merge Write MB	Save Write MB	Journal Write MB	Total Write	On Disk Size	In Memory Size	Fragment Count	Stands Written
load-from-disk	/data	1M12.82S	16	178	289	483	148	176	10000	7
generate-and-save	/tmp	3M41.19S	51	178	290	519	182	97	10000	7

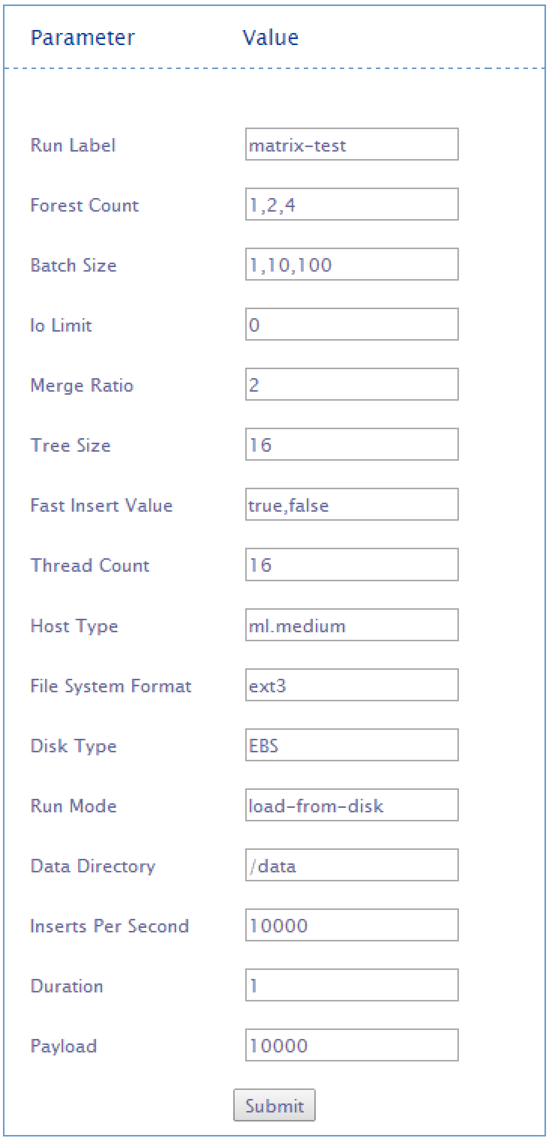
You can see that the load-from-disk mode, even for a small volume test, is substantially quicker on this platform. This is not surprising as an m1.medium is being used, and the CPU is the bottleneck on such a low power machine. The generate-and-save test can be useful however, especially when CPU load is required. Remember that one of the benefits of this application is that the tests can be easily repeated, so cross hardware comparisons can be made.

9 Matrix Tests

In sections 6 and 7 we run multiple versions of the same tests, but with different parameters in order to study the effects. Where those parameters are internal to MarkLogic, the application allows this process to be automated.

[15] Counting the number of documents exported

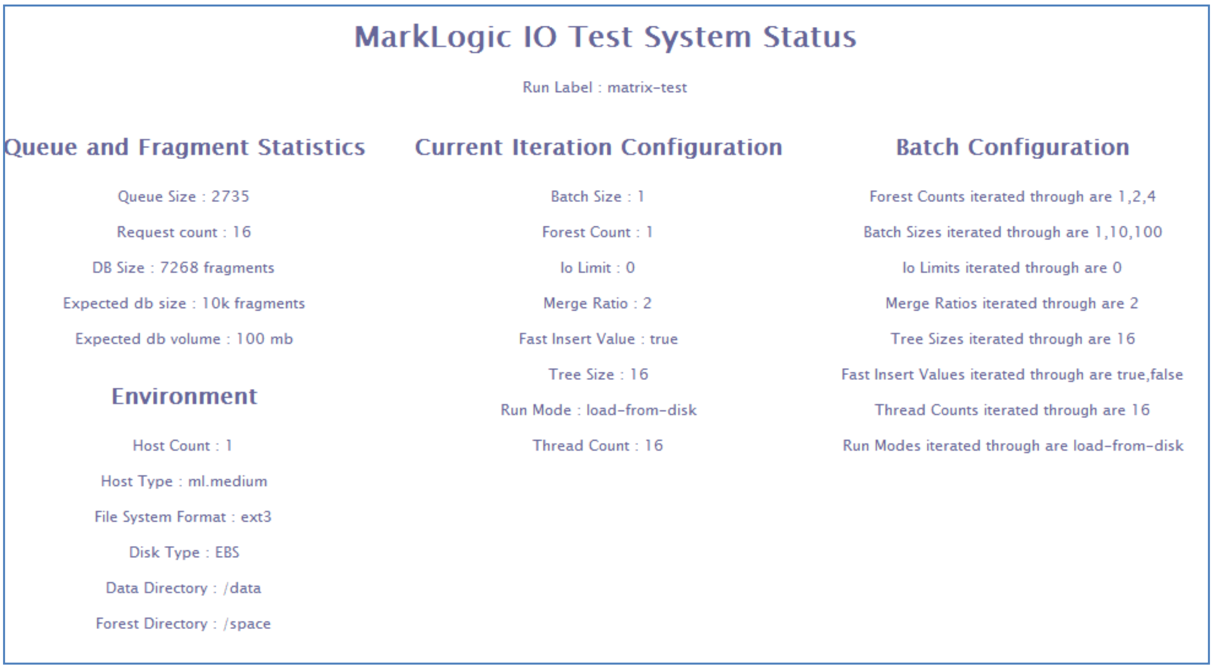
In this section we are going to run a matrix test. This involves setting up a job with multiple values for multiple parameters. In particular we are going to see what the effects are of varying forest count, batch size and fast insert value. To do this, set up the job as follows.



Parameter	Value
Run Label	matrix-test
Forest Count	1,2,4
Batch Size	1,10,100
Io Limit	0
Merge Ratio	2
Tree Size	16
Fast Insert Value	true,false
Thread Count	16
Host Type	ml.medium
File System Format	ext3
Disk Type	EBS
Run Mode	load-from-disk
Data Directory	/data
Inserts Per Second	10000
Duration	1
Payload	10000

To run with multiple values, create your job using comma separated values as above. Here I have three different forest count values, three different batch size values and am running using fast insert enabled and disabled. This is a total of 3 * 3 * 2 = 18 tests. My ‘matrix-test’ report will have 18 iterations therefore.

The status page will start as follows. Note that the iteration and batch configuration sections are now more useful. The iteration section shows the iteration running at a given point in time, while the batch configuration section shows the full job.



MarkLogic IO Test System Status

Run Label : matrix-test

Queue and Fragment Statistics

Queue Size : 2735

Request count : 16

DB Size : 7268 fragments

Expected db size : 10k fragments

Expected db volume : 100 mb

Environment

Host Count : 1

Host Type : ml.medium

File System Format : ext3

Disk Type : EBS

Data Directory : /data

Forest Directory : /space

Current Iteration Configuration

Batch Size : 1

Forest Count : 1

Io Limit : 0

Merge Ratio : 2

Fast Insert Value : true

Tree Size : 16

Run Mode : load-from-disk

Thread Count : 16

Batch Configuration

Forest Counts iterated through are 1,2,4

Batch Sizes iterated through are 1,10,100

Io Limits iterated through are 0

Merge Ratios iterated through are 2

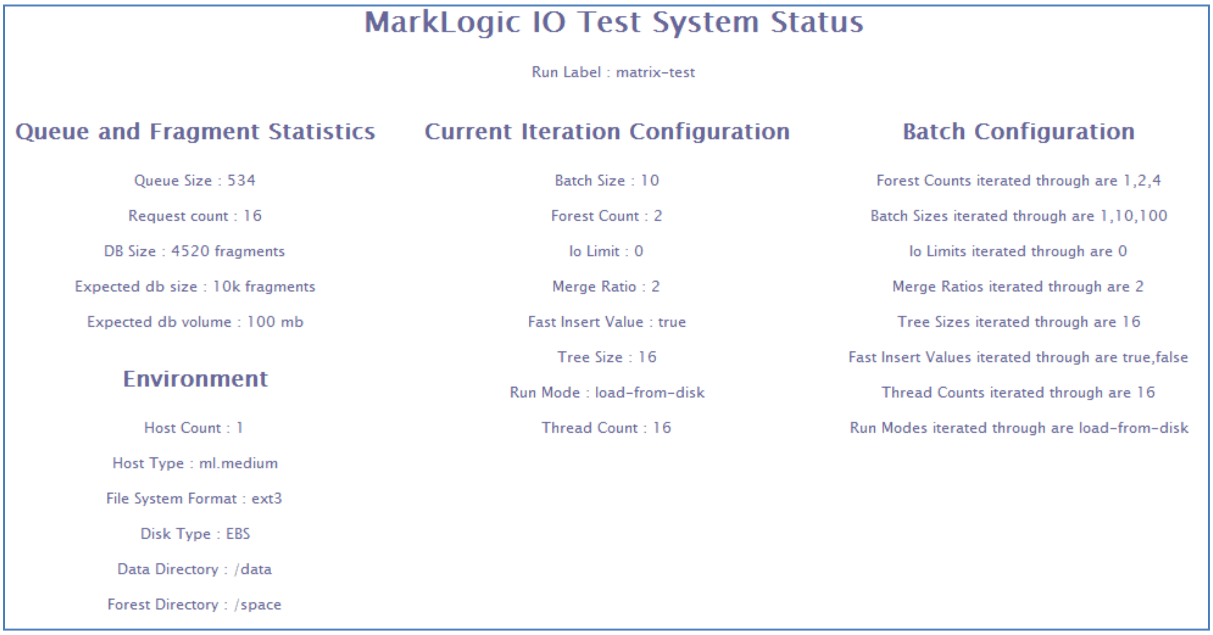
Tree Sizes iterated through are 16

Fast Insert Values iterated through are true,false

Thread Counts iterated through are 16

Run Modes iterated through are load-from-disk

The snapshot below shows an intermediate status – for batch size = 10 and forest count = 2. Note that as the batch size is 10, the total number of jobs will have been 10,000 / 10 = 1000. We can see that 452 such batches have completed, with 534 remaining.[16]



MarkLogic IO Test System Status

Run Label : matrix-test

Queue and Fragment Statistics

Queue Size : 534

Request count : 16

DB Size : 4520 fragments

Expected db size : 10k fragments

Expected db volume : 100 mb

Environment

Host Count : 1

Host Type : ml.medium

File System Format : ext3

Disk Type : EBS

Data Directory : /data

Forest Directory : /space

Current Iteration Configuration

Batch Size : 10

Forest Count : 2

Io Limit : 0

Merge Ratio : 2

Fast Insert Value : true

Tree Size : 16

Run Mode : load-from-disk

Thread Count : 16

Batch Configuration

Forest Counts iterated through are 1,2,4

Batch Sizes iterated through are 1,10,100

Io Limits iterated through are 0

Merge Ratios iterated through are 2

Tree Sizes iterated through are 16

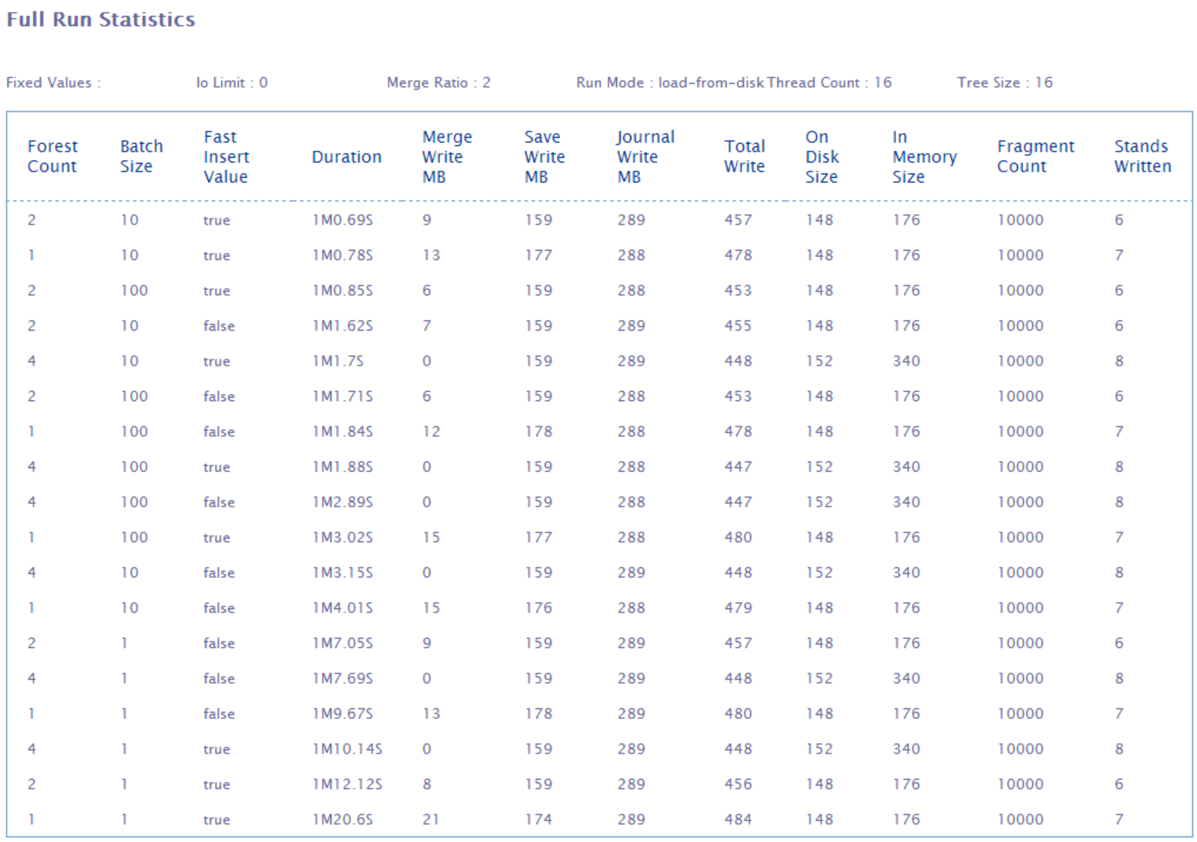
Fast Insert Values iterated through are true,false

Thread Counts iterated through are 16

Run Modes iterated through are load-from-disk

Once complete, my report looks like

[16] The remainder will be running requests.

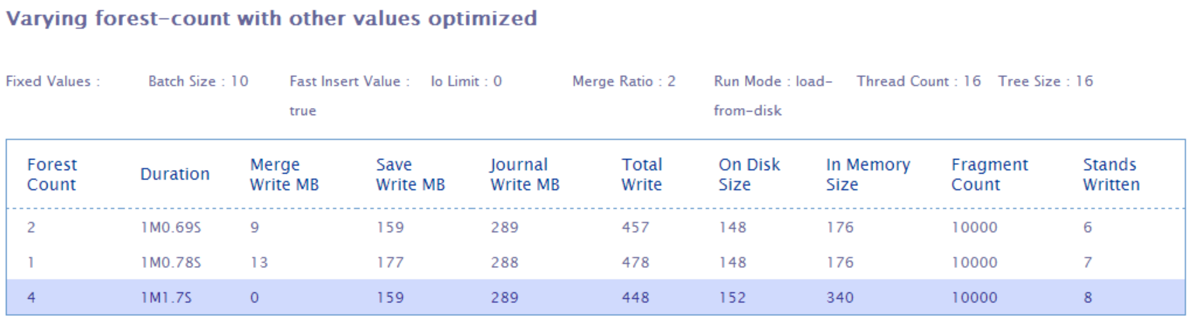


Full Run Statistics

Fixed Values : Io Limit : 0 Merge Ratio : 2 Run Mode : load-from-disk Thread Count : 16 Tree Size : 16

Forest Count	Batch Size	Fast Insert Value	Duration	Merge Write MB	Save Write MB	Journal Write MB	Total Write	On Disk Size	In Memory Size	Fragment Count	Stands Written
2	10	true	1M0.69S	9	159	289	457	148	176	10000	6
1	10	true	1M0.78S	13	177	288	478	148	176	10000	7
2	100	true	1M0.85S	6	159	288	453	148	176	10000	6
2	10	false	1M1.62S	7	159	289	455	148	176	10000	6
4	10	true	1M1.7S	0	159	289	448	152	340	10000	8
2	100	false	1M1.71S	6	159	288	453	148	176	10000	6
1	100	false	1M1.84S	12	178	288	478	148	176	10000	7
4	100	true	1M1.88S	0	159	288	447	152	340	10000	8
4	100	false	1M2.89S	0	159	288	447	152	340	10000	8
1	100	true	1M3.02S	15	177	288	480	148	176	10000	7
4	10	false	1M3.15S	0	159	289	448	152	340	10000	8
1	10	false	1M4.01S	15	176	288	479	148	176	10000	7
2	1	false	1M7.05S	9	159	289	457	148	176	10000	6
4	1	false	1M7.69S	0	159	289	448	152	340	10000	8
1	1	false	1M9.67S	13	178	289	480	148	176	10000	7
4	1	true	1M10.14S	0	159	289	448	152	340	10000	8
2	1	true	1M12.12S	8	159	289	456	148	176	10000	6
1	1	true	1M20.6S	21	174	289	484	148	176	10000	7

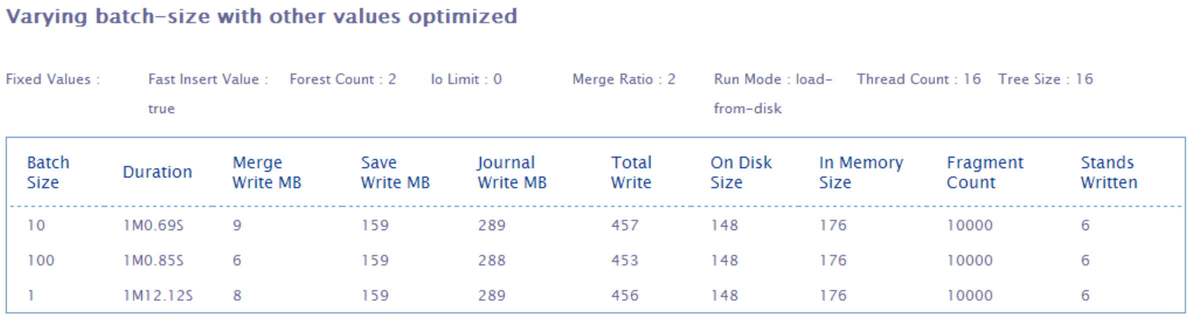
Again, as the test is short duration, there is no particular pattern to the results. I see a further six tables showing me what happens if each of the available parameters is altered for the optimum configuration (which is Forest Count =2, Batch Size = 10, Fast Insert = true), and the default configuration (which is Forest Count =1, Batch Size = 1, Fast Insert = true). The first three of these are shown.



Varying forest-count with other values optimized

Fixed Values : Batch Size : 10 Fast Insert Value : true Io Limit : 0 Merge Ratio : 2 Run Mode : load-from-disk Thread Count : 16 Tree Size : 16

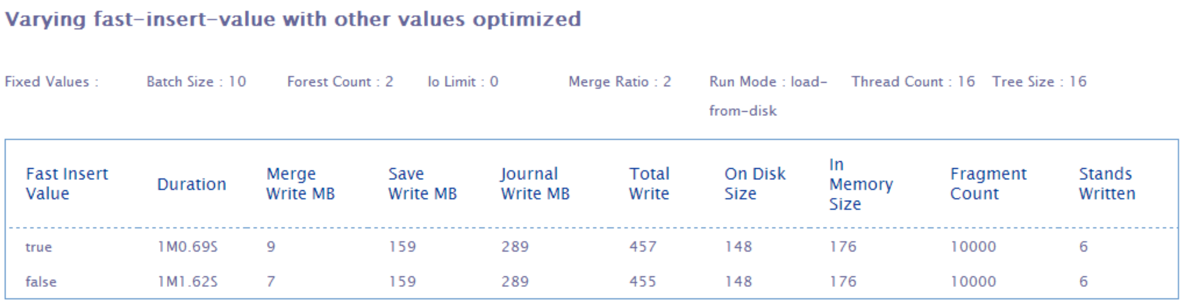
Forest Count	Duration	Merge Write MB	Save Write MB	Journal Write MB	Total Write	On Disk Size	In Memory Size	Fragment Count	Stands Written
2	1M0.69S	9	159	289	457	148	176	10000	6
1	1M0.78S	13	177	288	478	148	176	10000	7
4	1M1.7S	0	159	289	448	152	340	10000	8



Varying batch-size with other values optimized

Fixed Values : Fast Insert Value : true Forest Count : 2 Io Limit : 0 Merge Ratio : 2 Run Mode : load-from-disk Thread Count : 16 Tree Size : 16

Batch Size	Duration	Merge Write MB	Save Write MB	Journal Write MB	Total Write	On Disk Size	In Memory Size	Fragment Count	Stands Written
10	1M0.69S	9	159	289	457	148	176	10000	6
100	1M0.85S	6	159	288	453	148	176	10000	6
1	1M12.12S	8	159	289	456	148	176	10000	6



Varying fast-insert-value with other values optimized

Fixed Values : Batch Size : 10 Forest Count : 2 Io Limit : 0 Merge Ratio : 2 Run Mode : load-from-disk Thread Count : 16 Tree Size : 16

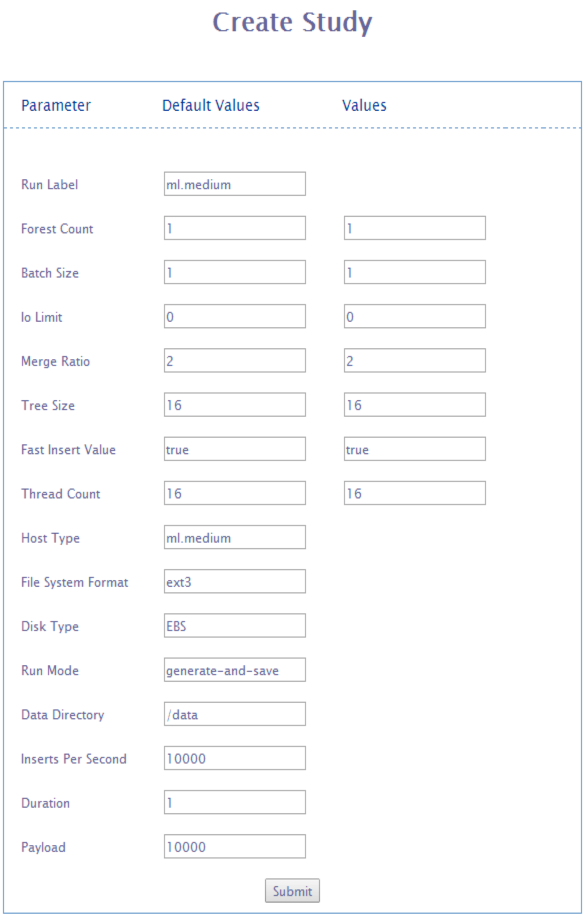
Fast Insert Value	Duration	Merge Write MB	Save Write MB	Journal Write MB	Total Write	On Disk Size	In Memory Size	Fragment Count	Stands Written
true	1M0.69S	9	159	289	457	148	176	10000	6
false	1M1.62S	7	159	289	455	148	176	10000	6

10 Running a Study

A disadvantage of the Matrix test is that the number of tests quickly gets large, as the total number of tests is the product of the number of possible values for each modified variable. If we were running half hour tests, the run above would take 9 hours in total.

An alternative is, to start with a set of default values and to vary parameters one at a time. This is the idea of a 'Study'. It results in a smaller number of tests, as the total is the sum of the possible values, for each modified variable.

To create a study, from the home page, click 'Create Study'.



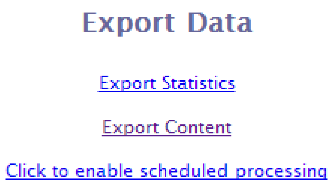
Create Study

Parameter	Default Values	Values
Run Label	ml.medium	
Forest Count	1	1
Batch Size	1	1
Io Limit	0	0
Merge Ratio	2	2
Tree Size	16	16
Fast Insert Value	true	true
Thread Count	16	16
Host Type	ml.medium	
File System Format	ext3	
Disk Type	EBS	
Run Mode	generate-and-save	
Data Directory	/data	
Inserts Per Second	10000	
Duration	1	
Payload	10000	

You can enter your default values, and then the values that you would like to have changing. I'm going to try different forest count values (1,2,4,8), different batch counts (1,10,100) and different merge ratios (2,4,6).

A study creates multiple jobs – corresponding to each parameter being varied. The job name is the run label concatenated with the parameter being varied.

We will also take advantage of the scheduled processing capability – without this, each job will require running manually from the jobs list page. To do this, from the home page, select 'Click to enable scheduled processing'.



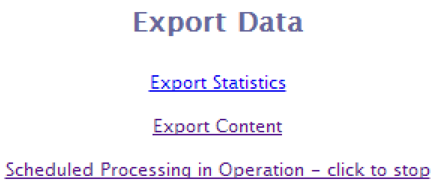
Export Data

Export Statistics

Export Content

Click to enable scheduled processing

You will see a screen labelled 'Scheduling activated'. The option in the home page will change.



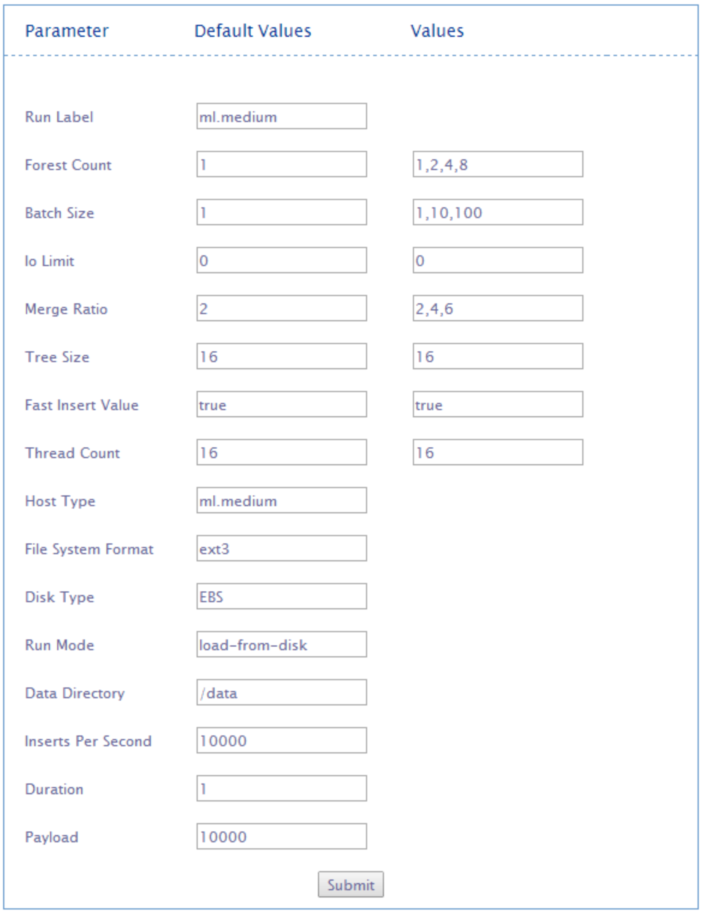
Export Data

Export Statistics

Export Content

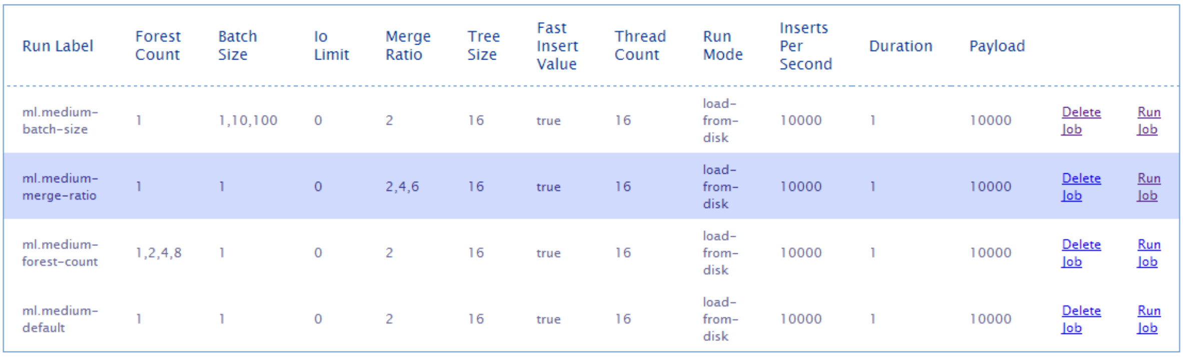
Scheduled Processing in Operation – click to stop

Here is my 'Create Study' screen.



Parameter	Default Values	Values
Run Label	ml.medium	
Forest Count	1	1,2,4,8
Batch Size	1	1,10,100
Io Limit	0	0
Merge Ratio	2	2,4,6
Tree Size	16	16
Fast Insert Value	true	true
Thread Count	16	16
Host Type	ml.medium	
File System Format	ext3	
Disk Type	EBS	
Run Mode	load-from-disk	
Data Directory	/data	
Inserts Per Second	10000	
Duration	1	
Payload	10000	

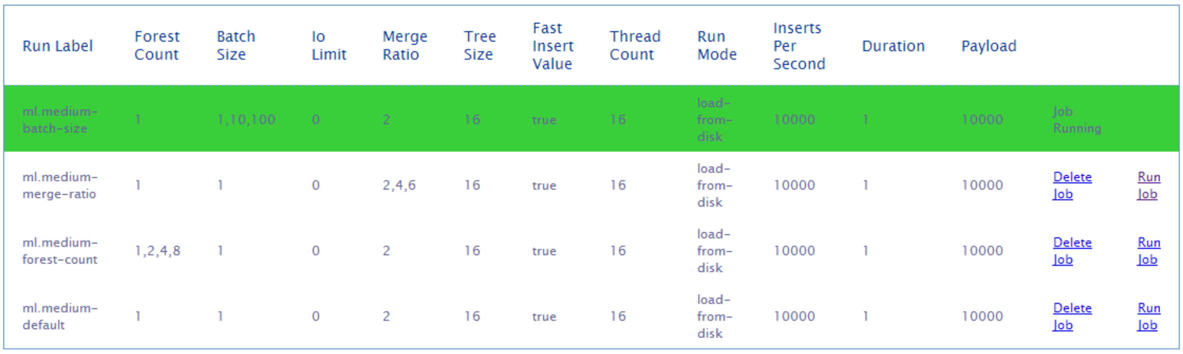
My job list screen, after submission, looks like



Run Label	Forest Count	Batch Size	Io Limit	Merge Ratio	Tree Size	Fast Insert Value	Thread Count	Run Mode	Inserts Per Second	Duration	Payload		
ml.medium-batch-size	1	1,10,100	0	2	16	true	16	load-from-disk	10000	1	10000	Delete Job	Run Job
ml.medium-merge-ratio	1	1	0	2,4,6	16	true	16	load-from-disk	10000	1	10000	Delete Job	Run Job
ml.medium-forest-count	1,2,4,8	1	0	2	16	true	16	load-from-disk	10000	1	10000	Delete Job	Run Job
ml.medium-default	1	1	0	2	16	true	16	load-from-disk	10000	1	10000	Delete Job	Run Job

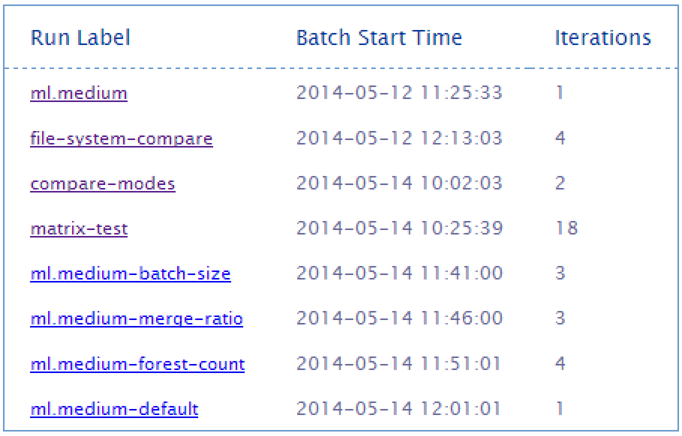
You can see I have one job for each parameter, and also a job for the default values- as it is possible to not run for the default value set (e.g. if I'd missed out 2 in the merge-ratio values).

Note that the scheduler runs every five minutes, so you may need to wait a little time before the first job starts. Your screen will then look like



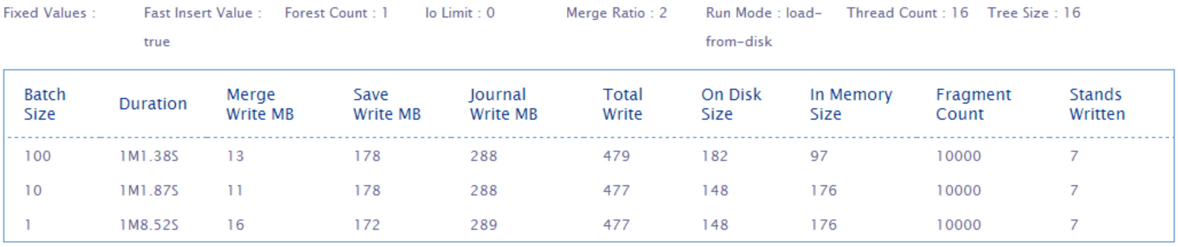
Run Label	Forest Count	Batch Size	Io Limit	Merge Ratio	Tree Size	Fast Insert Value	Thread Count	Run Mode	Inserts Per Second	Duration	Payload		
ml.medium-batch-size	1	1,10,100	0	2	16	true	16	load-from-disk	10000	1	10000	Job Running	
ml.medium-merge-ratio	1	1	0	2,4,6	16	true	16	load-from-disk	10000	1	10000	Delete Job	Run Job
ml.medium-forest-count	1,2,4,8	1	0	2	16	true	16	load-from-disk	10000	1	10000	Delete Job	Run Job
ml.medium-default	1	1	0	2	16	true	16	load-from-disk	10000	1	10000	Delete Job	Run Job

The application will run each job in succession. Following completion you will be able to see results via the report list page.



Run Label	Batch Start Time	Iterations
ml.medium	2014-05-12 11:25:33	1
file-system-compare	2014-05-12 12:13:03	4
compare-modes	2014-05-14 10:02:03	2
matrix-test	2014-05-14 10:25:39	18
ml.medium-batch-size	2014-05-14 11:41:00	3
ml.medium-merge-ratio	2014-05-14 11:46:00	3
ml.medium-forest-count	2014-05-14 11:51:01	4
ml.medium-default	2014-05-14 12:01:01	1

For each parameter being modified I have a clearly named report. If I look at ml.medium-batch-size for example I see



Fixed Values : Fast Insert Value : true Forest Count : 1 Io Limit : 0 Merge Ratio : 2 Run Mode : load-from-disk Thread Count : 16 Tree Size : 16

Batch Size	Duration	Merge Write MB	Save Write MB	Journal Write MB	Total Write	On Disk Size	In Memory Size	Fragment Count	Stands Written
100	1M1.38S	13	178	288	479	182	97	10000	7
10	1M1.87S	11	178	288	477	148	176	10000	7
1	1M8.52S	16	172	289	477	148	176	10000	7

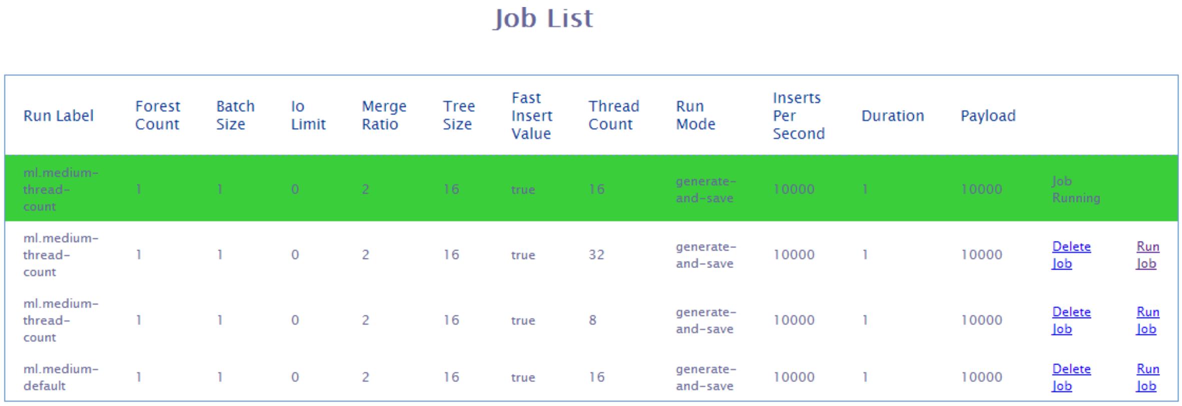
11 Tests involving Thread Count

If you wish to see how performance is affected by selection of thread count, you will not be able to do this via ‘Job Creation’ – instead you will have to use a Study.

The reason for this is that modifying thread count requires a server re-start – an individual job will not survive this.

To get round this problem, if thread count is being modified, the study creates multiple jobs with the same run label, so that for reporting purposes, they will all be grouped together.

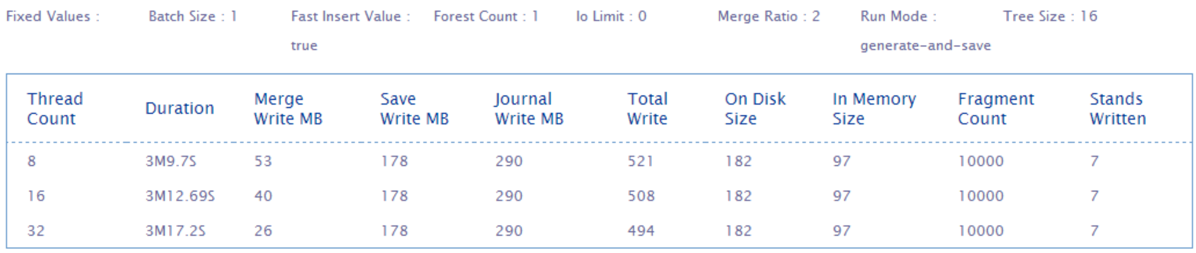
The screenshot below shows the results of my having created a study where the thread count values field was set to 8,16,32.



Job List

Run Label	Forest Count	Batch Size	Io Limit	Merge Ratio	Tree Size	Fast Insert Value	Thread Count	Run Mode	Inserts Per Second	Duration	Payload		
ml.medium-thread-count	1	1	0	2	16	true	16	generate-and-save	10000	1	10000	Job Running	
ml.medium-thread-count	1	1	0	2	16	true	32	generate-and-save	10000	1	10000	Delete Job	Run Job
ml.medium-thread-count	1	1	0	2	16	true	8	generate-and-save	10000	1	10000	Delete Job	Run Job
ml.medium-default	1	1	0	2	16	true	16	generate-and-save	10000	1	10000	Delete Job	Run Job

My resulting report looked like



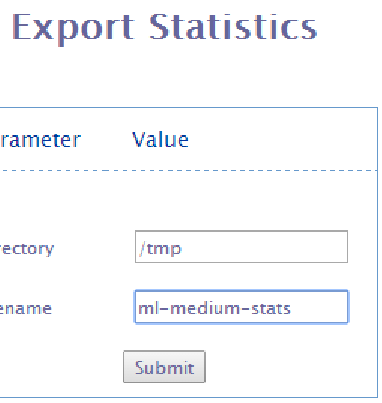
Fixed Values : Batch Size : 1 Fast Insert Value : true Forest Count : 1 Io Limit : 0 Merge Ratio : 2 Run Mode : generate-and-save Tree Size : 16

Thread Count	Duration	Merge Write MB	Save Write MB	Journal Write MB	Total Write	On Disk Size	In Memory Size	Fragment Count	Stands Written
8	3M9.7S	53	178	290	521	182	97	10000	7
16	3M12.69S	40	178	290	508	182	97	10000	7
32	3M17.2S	26	178	290	494	182	97	10000	7

12 Exporting Statistics

Having performed your tests, you can export your statistical data and import into another instance of the MarkLogic IO Test application.

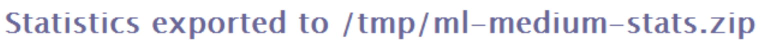
Select ‘Export Statistics’ from the home page and supply a directory and file name.



Export Statistics

Parameter	Value
Directory	/tmp
Filename	ml-medium-stats

You will see a message similar to



Statistics exported to /tmp/ml-medium-stats.zip

The easiest way to import to another system is to unzip to the ‘data’ directory in the Roxy project, and execute ‘ml <ENV> deploy content’.

13 Troubleshooting

If your job seems to get ‘stuck’, as with most things, try stopping and starting MarkLogic. This can happen if permissions have not been set correctly on the shares being used for Forests for example.

If you are changing the share that the Sample-01 forest is written to, then you may need to remove the forest manually, and possibly re-start the server.

Be aware that changing the number of task server threads may result in a re-start. If this happens, you will need to re-create your job, unless running in ‘scheduled mode’.

You can log in as the ‘application user’ io-test-user – but you will have a reduced set of privileges – you will not be able to run tests, but you will be able to browse reports. You will see a warning message on the home page if so. You need to log in as a user with admin privileges to run tests.

If running a study involving changing thread counts, the time between tests will be 10min rather then 5 min. This is because the scheduling results first in a restart due to modified configuration and then the job being executed.