Assessment Plan - Performance

Performance Testing Scenarios

Performance testing may be simplest if real-world scenarios are used that are drawn from the use patterns for Fedora 4. In other words, performance testing should be informed by the expected way Fedora will be used. No single software product can perform well for every possible use (a.k.a pattern of use). We need to define the expected uses for Fedora, and acknowledge the trade-offs that are being made. While we need to recognize that there will be unanticipated uses, we can only test for the cases that best characterize how we expect Fedora to be used. This sets expectations for those building systems using Fedora and guidance for Fedora committers. When there is a use is identified during system design, developers can decide if Fedora is suitable as a part of their implementation. Fedora committers can decide either Fedora can be extended to support that use or suggest how Fedora can be used in combination with other tools to support that use.

The scenarios are broken into major categories of use that give us examples of how Fedora is used to aid in constructing a realistic performance test suite. These are intended to be pragmatic categories since they present different loads on the system and should not be mechanically linked one-to-one with any API.

Definitions

- Use - Canonical action in a use case from the user's view
- User Operation - Users view of a logical, single operation
- Repository Operation - Repositories view of a logical, single operation for our purposes as the result of one or more API calls
- Performance - The number of units of work that are accomplished during an operation
- Concurrency - The number and kinds of operations being performed at the same nominal time (as opposed to a single repetition of an operation)
- Measurement - Metrics (to be defined) that are appropriate for measuring the results of a performance test of the Fedora Repository.
  - There may be more than one metric
  - Working ideas:
    - The amount in bytes of content and metadata for an operation, or per unit of time
    - The number of operations per unit of time
    - The time between when an operation is started and when it is completed
    - Count of operations performed possible eliminating content transfer time (normalized)

To Be Done

- Are these a good list of usage categories?
- Validate/Add user operations for each category (to the likely limits we can do). Prioritize short term goals. - This tells us how we expect Fedora 4 is to be used.
- Add user operations to tests
- Add repository operations to test
- Choose performance units for each test
- Choose performance expectations for each test
- Describe the test
- Construct concurrency goals for each test - Note: single thread tests will be use as a baseline as informed by Single Node tests
- Write a test script (for each test)
- Describe the Fedora 4 configuration
- Set up a test infrastructure using The Grinder
- Prepare data
- Test

Usage Categories

The following categories and uses are drawn from the Fedora 4 Roadmap and any new items especially expectations that have surfaced during development. Only performance related uses are included, specific function details are dropped for simplicity and that they are likely not need specific performance testing.

Authoring

Authoring is the activity of creating or assembling new content. This includes both constructing wholly new content and referencing existing content. It is different from ingest in that is characterized by incremental assembly of the content which many rapid write/read cycles to accomplish what the user considers a single operation. It is often performed as part of some sort of authoring workflow commonly with multiple actors performing both overlapping and different roles. During this phase the content (and metadata) is rapidly changing. The time between operations in a workflow can vary considerably so Authoring needs to handle the shortest period between user operations.

Simple Ingest

Simple Ingest consist of upload of single or small amounts of content and metadata. It can be accomplished with a single atomic operation or a short series of operations, usually RESTful, without required intermediate reads prior to completion. The duration of the small ingest is expected to be approximately the time it takes to upload the content and metadata starting with the beginning of the connection, where the connection is terminated after the operation. It is expected that the ability to read (access) the uploaded content and metadata should happen fairly soon after the upload is complete.

Bulk Ingest

In bulk ingest, a large quantity of content and metadata is ingested as a logical unit or is continuous. This may be accomplished using number of repository operations or may utilize methods that are optimized for bulk ingest. It is characterized by the expectation that there may be a defined delay between when the ingest is started and part or all of the content and metadata becomes available for read.

Simple Access

Simple access (a.k.a simple read or download) is the download of content and metadata (a.k.a representation of a resource) as a single user operation and one or a small number of repository operations. It usually RESTful, and usually contained a single request. Simple Access must not require any concurrent writes to accomplish the single user operation. The content and metadata stays fixed from the beginning to the end of the access.

Conditioned Access

When streaming media, dropouts present a significant problem. The user expects to be able to access the contents without interruption. This may require a front end tool for buffering so the stream need not be perfect but good enough for the buffering tool.

Mediated Access

Not all of the content is managed by Fedora but some resources are is provided by reference from a remote web service. Fedora would retrieve the representation (content and metadata) from the web service and present it as if it was a resource in Fedora.

Bulk Access

Download of large amount of content as single user operation. This may require any number of repository operations to accomplish. Whether content and metadata stays fixed from the beginning to the end of the operation is to be defined. This is needs consideration a whole intellectual entity, graph or DIP is considered the unit. Also we need to consider what this means for continuous access operations.

Tests

Baseline and Simple Concurrency Testing Matrix

The tests in the table will start with a single load injector and worker to use as a baseline. Then each of the tests are executed to test concurrency with increasing numbers of load injectors and workers until performance declines and/or error rates become large.

Category	User Operation	Repository Operation	Test Metric	Test	Priority	Notes
Simple Ingest	Small Files		Rate		1	Increase Load Injectors until max rate is found. Synthetic data is acceptable for this test
Simple Ingest	Medium Files		Rate		1	Increase Load Injectors until max rate is found. Synthetic data is acceptable for this test

Simple Ingest	Large Files		Rate		1+	Increase Load Injectors until max rate is found. Synthetic data is acceptable for this test
Simple Ingest	Media Files					
Simple Ingest	Large File Count		Rate - Normalized Count		1+	Ingest files to a substantial number to explore maximum file count. Normalized to ignore size of a given file.
Bulk Ingest	Small Files					
Bulk Ingest	Medium Files					
Bulk Ingest	Large Files					
Bulk Ingest	Large File Count		Rate - Normalized Count			Ingest files to a substantial number to explore maximum file count
Simple Access	Small Files		Random Access		1	Increase Load Injectors until max rate is found. Site should contain a set of files of uniform size. Tests can vary download mix by URL. It is essential that caching be avoided. Synthetic data is acceptable for this test.
Simple Access	Medium Files		Random Access		1	Increase Load Injectors until max rate is found. Site should contain a set of files of uniform size. Tests can vary download mix by URL. It is essential that caching be avoided. Synthetic data is acceptable for this test.
Simple Access	Large Files		Random Access		1	Increase Load Injectors until max rate is found. Site should contain a set of files of uniform size. Tests can vary download mix by URL. It is essential that caching be avoided. Synthetic data is acceptable for this test.
Simple Access	Mixed Files		Random Access		1+	Increase Load Injectors until max rate is found. Site should contain a set of files of all three sizes. Tests can vary download mix by URL. It is essential that caching be avoided. Synthetic data is acceptable for this test.
Simple Access	Large File Count		Random Access		1	Increase Load Injectors until max rate is found. Access a large number of files until time is exhausted.
Bulk Ingest	Small Files					
Bulk Ingest	Medium Files					
Bulk Ingest	Large Files					
Bulk Ingest	Mixed Files					

Concurrency Tests

The tests in the table combine two or more different tests from the table above. It starts with concurrent operation of at least two workers, one for each simple test. Then each of the tests are executed with increasing numbers of load injectors and workers until performance declines and/or error rates become large.

Category	User Operation	Repository Operation	Test Metric	Test	Priority	Notes
Concurrent Test #1	Simple Access Mixed Files Simple Ingest Mixed Files		Rate Random Access		1+	Number of Load Injectors TBD Number of Load Injectors TBD
Concurrent Test #N	Authoring Simulation		Rate Directed Access			

Testing Tools and Configuration

The suggested testing tool is The Grinder. It is a load testing framework written in Java, that uses Java, Jython and/or Clojure for writing the tests. A more complete discussion of our testing infrastructure may be found on this page.

Testing Considerations

These scenarios expand on the previous single load injector tests to use multiple read, write, and read-write tests via the REST api.

Note: all tests need to be taken until:

- a steady state is achieved
- a declining state is achieved
- Fedora 4 exhibits high error rates or stops responding

Load Injectors

- 1 Load Injector to provide a baseline similar to the previous testing regimen
- 3 Load Injectors
- 6 Load Injectors (since this is where Fedora 3 starts to exhibit limits)
- 12 Load Injectors (since this is where Fedora 3 always exhibits limits)
- 24 Load Injectors

Payloads

- 100K file (Minimal File, Also used in the single node baseline tests)
- 2.5 M file (Avg Hi-Res Image)
- 50M file (Avg Video, Also used in the single node baseline tests)
- 2.6G file (DVD)

Rates

- Step up rates X2 until flat line
- Then proceed to declining response, high error rates or non-response

Note: Single node test indicated a sensitivity to have large numbers of items as children of a single node. How should we deal with this?

Fedora Configurations

- Not Transactional
- Transactional
- Not Clustered
- Clustered
- Not Replicated
- Replicated

Resources

- https://github.com/pinterest/bender/blob/master/http/TUTORIAL.md
- https://github.com/fcrepo4-labs/fcrepo-test-grinder
- http://grinder.sourceforge.net/index.html