1. One thing to keep in mind is that we need to generate data + tests (e.g. Updates, deletions and queries). The tests, especially the queries, should yield a predictable (known) number of results so that we can measure accuracy.

The generation tool uses a seed for each instance. The same seed defined in the config file will always yield the same generated data. This means we can re-run the tool for each of our tests and it will produce the same data.

1. Can same data be generated dynamically even with some bad condition like service/system crash or interruption?

The tool is built with interruption in mind, it has a mechanism to see where it is up to before a crash and saves the current seed value. Restarting the tool will continue where it left off. For example having a crash at 50 million items and then continuing will produce the same data as running the tool for 100 million items with no interruption.

1. If we have to store files, what’s the best way?

The tool can generate data on the fly or persist onto disk. On the fly would be faster (less I/O to disk). However, in order to push into our service layer we must expose some method (WCF Service) that accepts some bytes, a stream etc. The EV tool has a layer which can push to other destinations, in our case the new service layer.

1. What’s the test data/query structure like?

Test data will be based on the spreadsheet created by Eduardo and completed by Red. This will determine what elements to include in the generated XML. We will most likely have all data stored in plaintext within the XML including the attachment text where necessary.

Essentially all queries will boil down to some Lucene Query as that’s the core indexing technology all our POC’s are using (for now). Our search tool will be responsible for passing these query onto the new search service layer. The search tool will be very simple, passing the query as is to the new layer, and this layer will then forward the query to the correct technology (Datastax, Solr or Elastic Search). This means the search tool will just accept the required query language, we will not complicate it at this stage by introducing a generic query language which gets mapped to either Datastax, Solr or Elastic Search. The tool will know the number of expected results thanks in part to the below mentioned ‘Quantity Tagging’ feature.

1. How we measure query complexity to test?

EV Gen has a cool mechanism known as Quality Tagging. These tags are defined in the config and state how many times a specific word or phrase should appear within the generated data. When executing searches we then know how many results should be returned. By setting phrases we can test complicated proximity searches (“hello world”~2). There’s also an ability to combine tags for AND + OR queries etc and other queries. Essentially you can go on to test a comprehensive set of queries.

1. We need to test multitalented environment and search for multiple groups (simulates loading and unloading of various Indexes, nodes etc).

I’ve spoken with the EV Gen author and we can alter the ‘Quantity tagging’ feature a bit to stipulate the GroupID element of our generated XML. This means we can state exactly how many XML’s should be generated for each group, mimicking a multi-tenant environment.

1. The tool must stress the data stores to some extent (i.e. generating on the fly we must still be able to stress the system). May involve having multiple processes running to push data simultaneously into the new service layer.

EV Gen is split into two parts. The main EV Generator tool and additional DropProcessors (each its own Process). The tool is where everything is configured and generated, this is then pushed to a queue which the DropPrcessors work off. We can spawn multiple DropPrcessors to concurrently ingest into the service layer therefore stressing it.

1. Test data to consist of a single XML/YAML/JSON whatever file which contains all the data.

We will stick to XML. If we have a WCF service layer we can stream an XML across the network and WCF internally does lots of compression and clever stuff. No need to experiment with YAML and JSON etc.

EV/Keith (the author) has already put a lot of effort into building the Environment Generator, its inception was in 2006! It is used by EV and more recently Clearwell. We should also look into using this now and going forward. EV used this tool heavily during their Indexing Engine POC’s back in the day, since then the tool has only become better!