NoSQL Connector Framework Overview

At Search Technologies, we are dedicated to creating really good content connectors. It is hard work. Honest work. And we work at it every day.

In the past few years, we have tackled some of the most difficult content sources. This includes legacy sources like Lotus Notes and eRooms, modern SaaS sources like Box.com, Salesforce, Jive, and big-player social networking sources like SharePoint Online (Office 365) and IBM Connections.

And we have learned a lot.

When I first met with the developers working on connectors for Search Technologies, we had done a few connectors and were in the process of ramping up the team. What I told them was this:

> It will get easier. Right now, every new connector is new. Each one requires new methods and new techniques to be effective. But at some point, connectors will start looking to be mostly the same. New connectors will get easier, because we will have seen most of the variation and we will have techniques in the framework to handle them all.

What's next?

- NoSQL Connector Framework Implementation
- NoSQL Connector Framework Security Implementation
- MongoDB Collections Description
- Failed Documents Handling

- Write Your Own Connector from Scratch

What I wasn't able to appreciate at the time was how much variety exists in the world of content management. Just when we think we have the problem licked, along comes a brand new technique that causes us to rethink everything. Recently we've been dealing with things like:

- Update tokens (which identify which content has changed)
- Databases of update tokens (separate tokens for each content type or site collection)
- Incomplete deletes (sometimes deletes only come for parent documents and so we have to delete the children ourselves from a recursive traversal of a stored hierarchy)
- Hybrid hierarchies (update tokens for some items, snapshots for other items)
- Content management system plug-ins (often required to provide the security rights information we need)

This is the fourth major refactor of the Connector framework since we started working on connectors. Why another refactor? Well, we have learned so much that it's time to synthesize all of that knowledge into a new framework that allows us to create better, faster, testable, more robust connectors than ever before.

Some of the new connector features that we expect to release soon include:

Connectors that are Easier to Write and Maintain

More code moved from the connector to the framework.

All of the learning (see above) on the various ways in which connectors scan for and process updates has been incorporated into the new framework structure. This means less lines of code for connector writers, which means less time to write and test connectors and more reliable connectors overall.

Improved Scalability and Performance

Crawling will scale linearly as you add machines.

Earlier versions of Aspire had a distributed communications method, but it was never integrated into the connector framework and it was never very high performance (it depended on an awkward method of 'return receipts'). This old code is now officially deprecated and will be removed from the next release of Aspire.

The latest framework is built to be multi-machine scalable from the start. Crawls will run as naturally on one machine as on five machines and will scale linearly as machines are added to the cluster.

Automatic Elasticity

Scalability occurs automatically. Just add machines.

Our 2.0 series of releases of Aspire were a huge step forward as we shifted to using Apache Zookeeper to hold and distribute crawler configuration. Now all Aspire machines in the cluster contain the same configuration. A change to any machine is automatically mirrored to all machines.

Now we are doing the same with crawler data. We will now use a NoSQL server to hold crawler state to allow for large-scale distributed crawling. Now you will be able to add a machine, it will automatically connect to the cluster, download all configuration, acquire crawl jobs from the NoSQL server and start crawling.

Leaderless = No Single Point of Failure

blocked URL*No complex system configuration or single point of failure. Just a cluster of machines which are all the same.*

Another cool feature about the new framework is that there is no “leader” or “controller.” The full system state is held in the NoSQL server and all of the crawlers cooperate to get crawler jobs done and acquire content.

This further simplifies configuration and set up. All Aspire nodes are exactly the same. Any node can do anything (start, stop, pause, resume, statistics, etc.). All configuration is shared across Zookeeper. There is no leader or set of workers which require special topology configuration.

Distributed Scanning

All parts of the process are now distributed, threaded, and parallelized.

Scanner jobs (those which fetch directory listings and lists of documents to crawl) are among the most difficult to distribute. Our new architecture now treats a scanner job as any other job, allowing for multiple parts of the content source to automatically divided up and scanned by multiple machines, automatically, for most connectors and crawl types.

No Shared Drives, No Local DBs = More Stable

All storage moved to a well-supported NoSQL server.

A weakness of the prior architecture was that snapshots and crawler state were stored in files and local (on-disk) databases. An upshot of this architecture is that these files and local DBs had to be stored on shared drives to allow for any machine to perform a crawl. These files and local DBs were responsible for most of the stability problems which we have encountered with Aspire.

All of this crawler state has been moved to a shared NoSQL server. This means that a professional NoSQL server (with backup, replication, and scalability) is now responsible for all shared state. This architecture will be much more stable.

Robust to Hardware Instability

Any set of machines can be shut down and restarted at any time.

Since all state is now held inside the NoSQL server (including all job, queues, and snapshot information), this means that any machine can be shut down at any time, and nothing is lost. When the machines start back up, they simply pick up exactly where they left off. We can even shut down everything, and the crawler will just pick back up where it left off when the system restarts.

This sort of robustness is required in modern cloud environments, where machines may be randomly restarted at any time by Amazon, Microsoft, or Google. If this happens in the new architecture, the crawl simply picks up where it left off.

Transparency

View the full progress and current state of the crawler at any time.

Again, because all state is in the NoSQL server, this provides an unprecedented amount of transparency and visibility into the current progress of the machine. By looking into the NoSQL server you can determine: the number of directories (waiting, completed, or in progress), the number of files to fetch (waiting, completed, or in-progress), and any and all snapshot information (the entire hierarchy of items and the ‘last touch time’ of every item).

This level of transparency will make it much easier to monitor and debug the crawlers and connectors.

The Master User Database = Security Monitoring

A single place for all user information across all your systems.

blocked URLWe’re going to start storing all user/group membership together for a user (rather than in separate files, as is currently the case). This means that you will be able to fetch a user from Aspire see all groups from all content sources [plus, eventually, other user attributes] together in a single object.

This is the “Master User Database.” The idea is a single master database where all user attributes and security information from around your organization across all of your content sources is gathered together and kept up-to-date all day long. Further, as user information changes from around your organization, it can be tracked and monitored, providing, essentially, a way to track all user security changes across your organization.

Imagine this: In the future, Aspire will be able to track all changes to a user’s security. Any change to group membership, any change to access rights, for systems and across all content. Pretty cool.

Backwards Compatible

Old connectors will still work in the new framework.

The new connector framework will require some (modest) refactoring of our connectors. While this is happening, old connectors will continue to work inside the new framework. This is important because we have a lot of connectors.

Announcing: Connector Libraries

We're getting more and more requests to use our connectors outside of the Aspire framework. Of course we like our Aspire framework, but we have absolutely no problem if someone wants to use our connectors in some other framework.

And so, we are simplifying our connector interfaces so that all connectors will support the same AbstractScanner and AbstractFetcher interfaces. This will make the connectors easier to program, easier to test, and more usable inside of external content processing frameworks. As an example, we have a demonstration of using Aspire connectors inside of "Kettle," the open source data integration framework from Pentaho.

Writing Connectors is Difficult, Tedious Work

Writing connectors is tedious, thankless work. Every connector is different and requires unique solutions. This is made even more complex because all corporate IT environments have different network topologies, content dimensions, feature usage, and security structures.

But we keep at it, working on connector after connector, working out each and every issue, working on performance, working to provide the best possible coverage, highest performance, and most features – working closely with our customers to get the connectors working reliably in production-scale systems.

We do this because connectors are vital. After all, if there is no data, there is no search. You have to get the data into the search engine, or nothing else matters.

The Dawning of a New Age of Search Engine Connectors

But there are also moments of joy. Our core engineering team is now fifteen people, and we are excited about this new framework. We hope to have early versions of it for release by the end of the calendar year, and I think that we all understand how this new framework will make everything better, all the way up and down the connector production chain.

Search Technologies is fully committed to connectors and to solving the unstructured data acquisition problem. We work on it day and night. Further, we understand that the problem will never truly be "solved." It will require on-going and unending work.

Will there be another framework refactor in two years' time? Of course! This is what it means to be committed: Constantly evolving and improving and forever working to make things better.