


A decorative graphic consisting of two overlapping squares: a smaller, solid olive-green square on top-left and a larger, semi-transparent light-green square on bottom-right.

What is MarkLogic Server?

An overview

A decorative graphic consisting of several overlapping squares in various shades of green and olive, arranged in a cluster.

By Jason Hunter
October 2010

Table of Contents

- 3 | What is MarkLogic Server?
- 3 | Document Centric
- 3 | Transactional
- 4 | Search-Centric
- 4 | Structure Aware
- 5 | Schema Agnostic
- 5 | XQuery and XSLT Driven
- 6 | High Performance
- 7 | Clustered
- 7 | Database Server
- 7 | Conclusion

What Is MarkLogic Server?

MarkLogic Server fuses together database internals, search-style indexing, and application server behaviors into a unified system. It uses XML documents as its data model, and stores the documents within a transactional repository. It indexes the words and values from each of the loaded documents, as well as the document structure. And, because of its unique Universal Index, MarkLogic doesn't require advance knowledge of the document structure (its "schema") nor complete adherence to a particular schema. Through its application server capabilities, it's programmable and extensible.

MarkLogic Server (referred to from here on as just "MarkLogic") clusters on commodity hardware using a shared-nothing architecture and differentiates itself in the market by supporting massive scale and fantastic performance — customer deployments have scaled to hundreds of terabytes of source data while maintaining sub-second query response time.

MarkLogic Server is a document-centric, transactional, search-centric, structure-aware, schema-agnostic, XQuery- and XSLT-driven, high performance, clustered, database server.

Let's look at all of this in more detail.

Document-Centric

MarkLogic uses documents, written in XML, as its core data model. Financial contracts, medical records, legal filings, presentations, blogs, tweets, press releases, user manuals, books, articles, web pages, metadata, sparse data, message traffic, sensor data, shipping manifests, itineraries, contracts, and emails are all naturally modeled as documents. In some cases the data might start formatted as XML documents (for example, Microsoft Office 2007 documents or financial products written in FpML), but if not, it can be transformed to XML documents during ingestion. Relational databases, in contrast, with their table-centric data models, can't represent data like this as naturally and so either have to spread the data out across many tables (adding complexity and hurting performance) or keep this data as unindexed BLOBs or CLOBs.¹

Transactional

MarkLogic stores documents within its own transactional repository. The repository wasn't built on a relational database or any other third party technology. It was purpose-built with a focus on maximum performance.

Because of the transactional repository, you can insert or update a set of documents as an atomic unit and have the very next query able to see those changes with zero latency.

¹ In addition to XML, you can store binary and text documents in MarkLogic.

MarkLogic supports the full set of ACID properties: *Atomicity* (a set of changes either takes place as a whole or doesn't take place at all), *Consistency* (system rules are enforced, such as that no two documents should have the same identifier), *Isolation* (uncompleted transactions are not otherwise visible), and *Durability* (once a commit is made it will not be lost).

ACID transactions are considered commonplace for relational databases but they're a game changer for document-centric databases and search-style queries.

Search-Centric

When people think of MarkLogic they often think of its text search capabilities. The founding team has a deep background in search: Chris Lindblad was the architect of the Ultraseek Server, while Paul Pederson was the VP of Enterprise Search at Google. MarkLogic supports numerous search features including word and phrase search, boolean search, proximity, wildcarding, stemming, tokenization, decomposing, case-sensitivity options, punctuation-sensitivity options, diacritic-sensitivity options, document quality settings, numerous relevance algorithms, individual term weighting, topic clustering, faceted navigation, custom-indexed fields, and more.

Structure-Aware

MarkLogic indexes both text and structure, and the two can be queried together efficiently. For example, consider the challenge of querying and analyzing intercepted message traffic for threat analysis:

Find all messages sent by IP 74.125.19.103 between April 11th and April 13th where the message contains both "wedding cake" and "empire state building" (case and punctuation insensitive) where the phrases have to be within 15 words of each other but the message can't contain another key phrase such as "presents" (stemmed so "present" matches also). Exclude any message that has a subject equal to "Congratulations". Also exclude any message where the matching phrases were found within a quote block in the email. Then, for matching messages, return the most frequent senders and recipients.

By using XML documents to represent each message and the structure-aware indexing to understand what's an IP, what's a date, what's a subject, and which text is quoted and which isn't, a query like this is actually easy to write and highly performant in MarkLogic. Or consider some other examples.

Find hidden financial exposure:

Extract footnotes from any XBRL financial filing where the footnote contains "threat" and is found within the balance sheet section.

Review images:

Extract all large-format images from the 10 research articles most relevant to the phrase "herniated disc". Relevance should be weighted so that phrase appearance in a title is 5

times more relevant than body text, and appearance in an abstract is 2 times more relevant.

Find a person's phone number from their emails:

From a large corpus of emails find those sent by a particular user, sort them reverse chronological, and locate the last email they sent which had a footer block containing a phone number. Return the phone number.²

Schema-Agnostic

MarkLogic indexes the XML structure it sees during ingestion, whatever that structure might be. It doesn't have to be told what schema to expect, any more than a search engine has to be told what words exist in the dictionary. MarkLogic sees the challenge of querying for structure or for text as fundamentally the same. At an index level, matching the XPath expression `/a/b/c` can be performed similarly to matching the phrase "a b c". That's the heart of the Universal Index.

Being able to efficiently index and query without prior knowledge of a schema provides real benefits with unstructured or semi-structured data where:

1. A schema exists, but is either poorly defined or defined but not followed.
2. A schema exists and is enforced at a moment in time, but keeps changing over time, and may not always be kept current.
3. A schema may not be fully knowable, such as intelligence information being gathered about people of interest where anything and everything might turn out to be important.

Of course, MarkLogic also works fine with data that does fully adhere to a schema. You can even use MarkLogic to enforce a schema, if you'd like.³

XQuery- and XSLT-Driven

To interact with and program MarkLogic Server you have your choice between two W3C-standard languages, XQuery and XSLT. XQuery is an XML-centric functional language designed to query, retrieve, and manipulate XML. XSLT is a recent addition to MarkLogic, added to make it easier to transform content during ingestion and output.

² How do you identify footers and phone numbers? You can do it via heuristics, with the markup added during ingestion. You can mark footer blocks as a `<footer>` element and a phone number entity as a `<phone>` element. Then it's easy to query for phone numbers within footers limited by sender name or address.

³ See <http://www.kellblog.com/2010/05/11/the-information-continuum-and-the-three-types-of-subtly-semi-structured-information/> for a deeper discussion of why so much structured information is really semi-structured information.

Each language has its advantages; you don't have to pick. You can mix and match between the languages: XSLT can make in-process calls to XQuery and vice-versa.

MarkLogic operates as a single process per host. It opens various socket ports for external communication. When configuring new socket ports for your application to use, you can pick between three distinct protocols:

HTTP and HTTPS Web Protocols

MarkLogic natively speaks HTTP and HTTPS. Incoming web calls can run XQuery or XSLT scripts the same way other servers invoke PHP, JSP, or ASP.NET scripts. These scripts can accept incoming data, perform updates, and generate output. Using these scripts you can write full web applications or RESTful web service endpoints, with no need for a front-end layer.

XDBC Wire Protocol

XDBC enables programmatic access to MarkLogic from other language contexts, similar to what JDBC and ODBC provide for relational databases. MarkLogic officially supports Java and .NET client libraries, named XCC. There are open source libraries in other languages. XDBC and the XCC client libraries make it easy to integrate MarkLogic into an existing application stack.

WebDAV File Protocol

WebDAV is a protocol that lets a MarkLogic repository look like a filesystem to WebDAV clients, of which there are many including built-in clients in most operating systems. With a WebDAV mount point you can drag-and-drop files in and out of MarkLogic as if it were a network filesystem. This can be useful for small projects; large projects usually create an ingestion pipeline and send data over XDBC.

High Performance

Speed and scale are an obsession for MarkLogic. They're not features you can add after the fact — they have to be part of the product in its core design. And they are, from the highly-optimized native C++ code to the algorithms underpinning the system. For MarkLogic customers it's routine to compose advanced queries across terabytes of data that make up many millions of documents and get answers in less than a second. The largest live deployments now exceed 200 terabytes and a billion documents. The largest projects now under development will exceed a petabyte.

In the words of Flatirons Solutions, an integration partner: "It's fast. It's faster than anybody else. It's way, way faster. It blows you away it's so fast. It's actually so fast that it... makes it possible to do real-time queries against large XML databases; makes it possible to do large-scale personalization from XML data; makes it possible to think about classic problems in an entirely new way."

Clustered

To achieve speed and scale beyond the capabilities of one server, MarkLogic clusters across commodity hardware connected on a LAN. A commodity server in 2010 might be a box with 4 or 8 cores, 32 or 64 gigabytes of RAM, and either a large local disk or access to a SAN. On a box such as this a rule of thumb is you can put roughly 500 gigabytes to 1 terabyte of data, sometimes more and sometimes less, depending on your use case.

Every host in the cluster runs the same MarkLogic process, but there are two specialized roles. Some hosts (Data Managers, or *D-nodes*) manage a subset of data. Other hosts (Evaluators, or *E-nodes*) handle incoming user queries and internally federate across the D-nodes to access the data. A load balancer spreads queries across E-nodes. As you load more data, you add more D-nodes. As your user load increases, you add more E-nodes. Note that in some cluster architecture designs the some host may act as both a D-node and an E-node. In a single-host environment that's always the case.

Clustering enables high availability. In the event an E-node should fail, there's no host-specific state to lose, just the in-process requests (that can be retried), and the load balancer can route traffic to the remaining E-nodes. Should a D-node fail, that subset of the data needs to be brought online by another D-node. You can do this by using either a clustered filesystem (allowing another D-node to directly access the failed D-node's storage and replay its journals) or intra-cluster data replication (replicating updates across multiple D-node disks, providing in essence a live backup).

Database Server

At its core you can think of MarkLogic as a database server — but one with a lot of features not usually associated with a database. It has the flexibility to store structured, unstructured, or semi-structured information. It can run both database-style queries and search-style queries, or a combination of both. It can run highly analytical queries too. It can scale horizontally. It's a platform, purpose-built from the ground up, that makes it dramatically easier to author and deploy today's information applications.

Continuing Onward

If you wish to learn more, you can find downloads, documentation, and discussion at <http://developer.marklogic.com>.

This text is extracted from the introduction of a much longer paper describing MarkLogic Server internals: its data model, indexing system, update model, and operational behaviors. Please contact your MarkLogic representative if you'd like to obtain the extended version or visit <http://developer.marklogic.com/inside-marklogic>.

For the complete version of this paper, please download it from
<http://developer.marklogic.com/inside-marklogic>

For more information please visit www.marklogic.com

or call 1 877 993 8885