CM-Well: A Data Warehouse for Linked Data

Dan Bennett[1], Jason Engelbrecht[2], and Dudi Landau[3]

[1] Thomson Reuters, 610 Opperman Drive, Eagan, MN, 55123, USA
[2] Thomson Reuters, 30 South Colonnade, Canary Wharf, London, E14 5EP, UK
[3] Thomson Reuters, 94 Derech Em Hamoshavot, Petach Tikva 49527, Israel
`{dan.bennett,jason.engelbrecht,dudi.landau}@tr.com`
`http://www.thomsonreuters.com`

Abstract. In this paper we present CM-Well, a clustered, horizontally scalable data store for linked data. We discuss the key architectural principals of the system and introduce distinguishing features. Thomson Reuters has been running CM-Well in production for over two years, across multiple data centers. The system provides data, via API, to a number of our online products.

Keywords: Data Warehousing, Linked Data, SPARQL, Distributed

1 Introduction

A traditional data analytics approach is to use a centralized data warehouse that collects relevant information before onwards distribution to data marts[4]. These marts then structure and index data in forms appropriate for the desired consumption pattern. We observed that a similar model could work well for linked data. However, we did not find industry solutions that would serve this need well, particularly optimized for incremental flow of linked data.

In this paper, we introduce CM-Well[5]: an open source application for managing large volumes of linked data. Within Thomson Reuters, we run multiple clusters; for warehousing of our Knowledge Graph in addition to work-in-progress editorial systems. The focus of CM-Well is on content management and distribution: higher-level functions, such as reasoning, are left to other systems. While CM-Well includes a browser based user interface, typically most interactions are via REST API.

2 Architecture

CM-Well is based on a clustered architecture (Fig 1). Philosophically, the application is similar to a Linux distribution in that it consists of a well tested

[4] https://en.wikipedia.org/wiki/Data_mart
[5] https://github.com/thomsonreuters/CM-Well

integration of a number of open source packages including Akka[6], Cassandra[7], ElasticSearch[8], Jena[9] and Kafka[10].

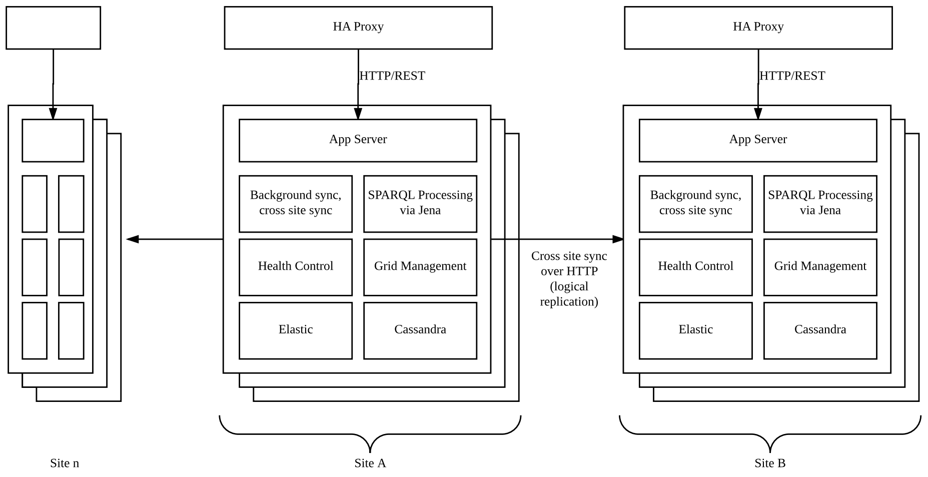
HA Proxy
HTTP/REST
App Server
Background sync, cross site sync
SPARQL Processing via Jena
Health Control
Grid Management
Elastic
Cassandra
Cross site sync over HTTP (logical replication)
Site n
Site A
Site B

Fig. 1. CM-Well Architecture

Each node in the cluster has the same hardware configuration and runs a set of processes with no single point of failure. Singleton control roles are moved between nodes on failure, implemented on a "self healing" principle. In a typical deployment, the cluster is fronted by an HTTP load balancer such as haproxy[11]. The majority of the application is written in the Scala language.

2.1 Grouping Triples as immutable "Infotons"

Individual triples are grouped by subject to form an "Infoton": the basic unit of storage. These information objects are stored in Cassandra and invert indexed by ElasticSearch. While writes require a full rewrite of the infoton, reads by subject can be served from one node and return all triples for the subject. This grouping of triples forms a fundamental design tradeoff between read and write performance but has been found to offer good horizontal scale performance: on our production cluster, API response time for all information on a subject is around 10ms.

[6] http://akka.io/
[7] http://cassandra.apache.org/
[8] https://github.com/elastic/elasticsearch
[9] https://jena.apache.org
[10] https://kafka.apache.org
[11] http://www.haproxy.org

Every Infoton write is treated as immutable; this approach permits eventually consistent replication across distributed data centers with secondary instances subscribing to a change log and applying writes locally. Depending on read/write tradeoff, all slaves can replicate from a single master or slaves can form a branching chain.

2.2 Deployment

While we have found CM-Well capable of balancing read and write traffic, in most applications consumers wrap CM-Well with a data-mart implemented to meet the specific needs of their application. Depending on use case, this mart might be as simple as a caching reverse HTTP proxy to reduce read load or deployment of a separate search cluster which structures data to meet required performance targets.

3 Key features

One of the motivating factors for the development of CM-Well was our perception that key requirements were not met by other tools; either open or closed source. Integrating our own solution permitted us to focus on key differentiating features.

3.1 Subscription by query

The API surface for CM-Well includes a REST based API for boolean querying by predicate. Any query can be used as a (server-side stateless) subscription; returning a time ordered sequence of Infotons matching the query.

Each stream API call returns a continuation token in the HTTP response headers. When passed in a subsequent request, this token is used as a point-in-time marker with the stream continuing from the end of the previous response. Additionally, an optional parameter will instruct CM-Well to include tombstones for deleted data, effectively treating each query as a FIFO queue of content events.

This feature is key to the 'information in motion' design of CM-Well permitting downstream stores to subscribe to relevant subsets of data defined by query.

3.2 SPARQL Support

Currently, CM-Well supports two types of SPARQL query: sub and full graph. When querying, the sub-graph solution consists of two sub-steps: 1) one or more queries, routed to ElasticSearch, are first used to select a candidate set of nodes from our data; 2) Such candidate nodes are then loaded to a Jena triple store on a single machine for SPARQL execution. We are also experimenting with a second form which builds an execution plan for input ElasticSearch queries based on data statistics relevant to the SPARQL query. Both solutions are limited to the underlying characteristics of ElasticSearch, such maximum results window of 10,000. Hence, wide ranging queries can miss triples of relevance.

3.3 Triggers

Much of the data we load to CM-Well is highly normalized and often requires simplification to ease downstream consumption. Our trigger mechanism permits a sensor to be created, based on a query, which on new data invokes a SPARQL CONSTRUCT statement. This then generates new predicates.

For example, in our organization ontology, company names are modeled as their own type <OrganizationName>. Each name includes date based "from" and "to" predicates to model company name changes over time. While this temporal model is powerful, it complicates the simple retrieval of a company, since one must traverse the set of attached names. A trigger is used to create (or update) a "flattened" name predicate on the organization entity on update of related organization name entity.

4 Demonstration plan & Conclusions

In this paper we have briefly presented our approach to managing & distributing large volumes of linked data using CM-Well. We believe the application presents a cost effective way of building and distributing linked datasets and are pleased to be able to share with the wider community via open source: without which CM-Well wouldn't be possible.

A video demo to support this paper can be found on Vimeo[12]: the demo shows a subset of the data from the permid.org dataset, update via API, immutable history, retrieval by API and our implementation of "full-graph" SPARQL.

Acknowledgments. The authors would like to acknowledge the efforts of the entire CM-Well development team, both current and historical, in addition to Thomson Reuters for supporting the open sourcing of CM-Well.

References

1. Dayarathna, M., Herath, I., Dewmini, Y., Mettananda, G., Nandasiri, S., Jayasena, S., Suzumura, T.: Introducing acacia-rdf: An x10-based scalable distributed rdf graph database engine. In: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 1024–1032 (May 2016)
2. Peng, P., Zou, L., Özsu, M.T., Chen, L., Zhao, D.: Processing sparql queries over distributed rdf graphs. The VLDB Journal 25(2), 243–268 (Apr 2016), https://doi.org/10.1007/s00778-015-0415-0
3. Punnoose, R., Crainiceanu, A., Rapp, D.: Sparql in the cloud using rya. Information Systems 48, 181 – 195 (2015), http://www.sciencedirect.com/science/article/pii/S0306437913000975
4. Um, J.H., Lee, S., Kim, T.H., Jeong, C.H., Song, S.K., Jung, H.: Distributed rdf store for efficient searching billions of triples based on hadoop. The Journal of Supercomputing 72(5), 1825–1840 (May 2016), https://doi.org/10.1007/s11227-016-1670-6

[12] https://vimeo.com/226891533