Comparison of Triple Stores

Abstract
In this report we present evaluation of triple stores. We present load times and discuss the inferencing capabilities of **Jena SDB backed with MySQL, Sesame native, Mulgara and Virtuoso.** We also discuss the suitability and performance of the triple store when used as a backend for Bioportal. Our evaluations are performed using the UNIPROT data and ontologies from Bioportal. We finally conclude by suggesting a hybrid approach which combines a native triple store with the fine level access of an API.

Introduction
A triple store is framework used for storing and querying RDF data. It provides a mechanism for persistent storage and access of RDF graphs. Recently there has been a major development initiative in query processing, access protocols and triple-store technologies. The number of triple stores being actively developed has increased from Jena and Sesame in the early 2000s to Garlik JXT, YARS2, BigOWLIM, Jena TDB, Jena SDB, Virtuoso, AllegroGraph, BigData, Mulgara, Sesame, Kowari, 3Store and RDF Gateway. Among these some like Garlik and YARS2 are propriety and not distributed. Others like BigOWLIM and AllegroGraph are commercially available. The rest are all free & open source. The main purpose of our study was to find suitable backend for Bioportal. Majority of our effort was to concentrate on freely available open source triple stores and hence the choice of Jena SDB, Sesame, Mulgara and Virtuoso.

Triple stores can be divided into 3 broad categories – in-memory, native, non-memory non-native - based on architecture of their implementation. In-memory triple stores store the RDF graph in main memory. Storing everything in-memory cannot be a serious method for storing extremely large volumes of data. However, they can act as useful benchmark and can be used for performing certain operations like caching data from remote sites or for performing inferencing. Most of the in-memory stores have efficient reasoners available and can help solve the problem of performing inferencing in persistent RDF stores, which otherwise can be very difficult to perform. A second, now dominant category of triple stores is the native triple stores which provide persistent storage with their own implementation of the databases, for eg, Virtuoso, Mulgara, AllegroGraph, Garlik JXT. The third category of triple stores, the non native non memory triples stores are set up to run on third party databases for eg. Jena SDB which can be coupled with almost all relational databases like MySQL, PostsgreSQL, Oracle. Recently native triple stores due to their superior load times and ability to be optimized for RDF have gained popularity.

In our evaluations we use a non memory non native store – Jena SDB, a native store – Sesame native which has an API to provide fine level access, Virtuoso and Mulgara which are native stores but without an extensive API. Virtuoso and Mulgara are also claimed to be more scalable than Sesame native [Give reference].

In the next section we present a brief overview of the triple stores being evaluated. The following sections describe the dataset being used, results and the conclusion.

Jena

Jena is a java framework for building semantic web applications. Jena implements APIs for dealing with Semantic Web building blocks such as RDF and OWL. Jena's fundamental class for users is the `Model`, an API for dealing with a set of RDF triples. A `Model` can be created from the filesystem or from a remote file. Using JDBC, it can also be tied to an existing RDBMS such as MySQL or PostgreSQL. SDB is a component of Jena. It provides for scalable storage and query of RDF datasets using conventional SQL databases for use in standalone applications, J2EE and other application frameworks. The storage, as mentioned, is provided by an SQL database and many databases are supported, both open source and proprietary. An SDB store can be accessed and managed with the provided command line scripts and via the Jena API. In addition on the fine grain access provided by the Jena API, SDB can be coupled with the web-server
- 'Joseki' - which is SPARQL query server. This enables an SDB store to be queried over HTTP. Jena recently introduced a non-transactional native store called TDB. For our current evaluation we have Jena SDB backed with MySQL.

Jena in addition to the above mentioned features also provides access to a reasoned that configured to perform reasoning of various degrees. It can be used to perform simple RDFS reasoning to the more memory intensive OWL-DL reasoning. Popular external reasoners like the Pellet reasoner can be also be used with it. All the reasoners made available by the Jena API perform the reasoning in-memory, In other words they require the data to be present an in-memory model.

Sesame

Sesame is an open source framework for storage, inferencing and querying of RDF data. Sesame matches the features of Jena with the availability of a connection API, inferencing support, availability of a web server and SPARQL endpoint. Like Jena SDB it provides support for multiple backends like MySQL and Postsgre.
Sesame Native is the native triple store offering from Sesame. As compared to be Jena's native triple, TDB, it is less scabable. [1] mentions that Sesame native has been tested with upto ~70 Million triples while Jena TDB is known to work with 1.7 Billion triples.

Mulgara

Mulgara is a native RDF triple store written in Java. It provides a Connection API that can be used to connect to the Mulgara store. Being a native triple store it has a 'load' script which can be used to load RDF data into the triple store. In addition to supporting SPARQL queries through the connection API, these can be performed through the TQL shell. The TQL shell[Ref] is a command line interface that allows queries on models present in the store.

Virtuoso

Virtuoso, is a native triple store available in both open source and commercial licenses. It provides command line loaders, a connection API, support for SPARQL and web server to perform SPARQL queries and uploading of data over HTTP. A number of evaluations have tested virtuoso and found it to be scalable to the region of 1B+ triples.
In addition to this, a Virtuoso provides bridges for it to be used with Jena and Sesame.

Data Sets
To facilitate the evaluation of the triple stores we use 2 datasets. UNIPROT[1] is a publicly available dataset which contains information on proteins. It is a central repository of protein sequence and function created by joining UniProt Knowledgebase (UniProtKB), the UniProt Reference Clusters (UniRef), and the UniProt Archive (UniParc). Since our primary use case is to evaluate feasibility of a triple store as backend for Bioportal, our second dataset comprises ontologies from Bioportal.

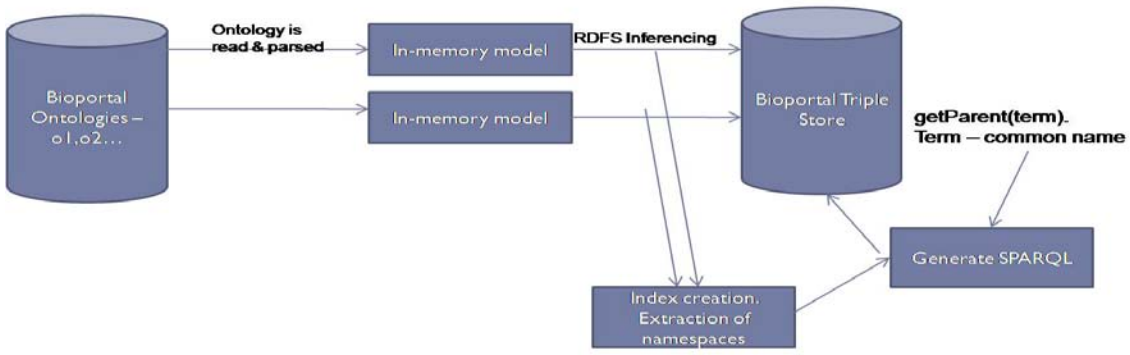
The two datasets serve different purposes and help ensure that the triple stores are compared on a number of different axis. We use two datasets from UNIPROT, the first has 1 Million triples and the second 10 Million. Even though 10 million is a small number in the world of datasets, it helps quantify scalability of the RDF triple stores.

Ontologies from Bioportal
The ontologies from bioportal help evaluate feasibility of using a triple store as a backend for Bioportal. In addition to this, most of the use-cases in bioportal require some level of inferencing over the ontologies. Thus using this data helps measure the support for inferencing in the RDF triple stores.
If a triple store is used as backend of the bioportal the queries need to be performed as they are by the current implementation of bioportal. Common name of the term is used in the current queries. However, when used with triple store as the backend, the namespace to which the term belongs needs to be made available for execution of queries. Since currently ontologies in the bioportal do not have a common namespace, the namespaces of each of the ontology needs to be extracted and stored as the ontology is being loaded into the triple store. This preprocessing step helps measure the effectiveness of the API with the triple store.
Since the user only supplies the common name of the term to query, the SPARQL query with the namespace of the term needs to be created. The namespaces extracted in the above preprocessing are used in this step. Since the namespace to which the term originally belongs is not known, combination with all the namespaces extracted is performed. This causes a single query to fan-out by a factor of ~100 and puts strain on the triple store and helps measure the ability of the triple store to perform a large number of queries in a short duration. More information about this is available in the methodology section.

Evaluation Methodology

All the triple stores were accessed through their suggested APIs i.e. Jena SDB through the Jena API, Sesame Native and Mulgara through Sesame and Virtuoso through its connection API.
Even though the UNIPROT data can be loaded into the triple store without use of an API, we preferred testing loading using an API since it was a requirement for the ontologies of bioportal. As already mentioned, the ontologies of bioportal don't have a common namespace. The namespaces associated with the ontologies are extracted before loading them. This enables us to transform a user query in 'common name' into a SPARQL query with complete namespaces. Figure 1 illustrates the process of loading the data into a triple store.

Inferencing Approach
A bioportal installation based on triple stores requires inferencing atleast to the level of RDFS i.e. we need an inferencing mechanism that performs transitive closure on the sub-property and sub-class hierarchy. All the off the shelf reasoners available expect the data to be cached in-memory to perform the reasoning. However, in our case due to size of the data it is stored in persistent storage.
This inferencing limitation further precipitates the need for the triple store to have an available API. In our current implementation, we perform inferencing as the ontologies are being loaded. The ontologies are cached into an in-memory model, their namespaces extracted and RDFS inferencing performed on them before pushing them into the triple store.



Results

Table 1, 2 and 3 present the time taken to load the three datasets. 'Couldn't load' in the time field denotes that an out-of-memory error started appearing. These results are discussed in the conclusion section.

Triple Store	Time(min)
Jena SDB	164
Sesame Native	7
Mulgara	12
Virtuoso	Couldn't load

Table 1: Time Taken to load the bioportal ontologies

Triple Store	Time(min)
Jena SDB	8.19
Sesame Native	4.46
Mulgara	Couldn't load
Virtuoso	Couldn't load

Table2: Time Taken to load the UNIPROT 1M

Triple Store	Time(min)
Jena SDB	175
Sesame Native	123
Mulgara	Couldn't load
Virtuoso	Couldn't load

Table3: Time Taken to load the UNIPROT 10M

Query Results

The queries on bioportal mainly extract the parent and child of a given term. Since these are first order simple queries, not much difference is seen in the performance. More complex queries that recursively calculate the path to root of a term should be used to get a better indication of the querying capabilities.

Child queries

The child queries are used to extract the children of a given term. As already mentioned the user only supplies the 'term', without the associated namespace for which he needs the children. To create the SPARQL query we plug in the term with all the known namespaces extracted. This created query is then executed against the triple store. For eg. if the user wants to find the children of the term **'Size',** a concept mentioned in the amino-acid ontology, the following query is created:

*PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> SELECT ?x FROM <bioportal:amino-acid> WHERE { ?x rdfs:subClassOf <http://www.co-ode.org/ontologies/amino-acid/2006/05/18/amino-acid.owl#**Size**>*

This query is in addition to the other queries generated using different namespaces like *http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#.* Using this namespace another query(shown below) is generated. Similarly queries for other namespaces are generated.

Thus, for each term we need to generate as many queries as the number of namespaces present in Bioportal. This results in a fanout of the order of ~300.
*PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> SELECT ?x FROM <bioportal:Thesaurus > WHERE { ?x rdfs:subClassOf < http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#**Size**>*

Parent queries

A similar procedure of generating multiple queries for extracting the parents of a term is used. Multiple SPARQL queries with different namespaces are created and executed against the triple store. The only difference from the child queries, is that the variable is at object position instead of the subject. For eg. to extract the parents of the term '**Size',** one of the generated queries is:
*PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> SELECT ?x FROM <bioportal:amino-acid> WHERE { <http://www.co-ode.org/ontologies/amino-acid/2006/05/18/amino-acid.owl#**Size** rdfs:subClassOf ?x>*

Similar to the case of children queries, the fanout for parent queries is of the order ~300 or the number of namespaces in Bioportal.

Get All Children/Parents

To extract the all the children or parents of a term, we recursively call the children or parents with the results from the previous step. However, unlike the first children query there is no fanout for the succeeding children queries. This is because, at the first step we have no information regarding the namespace that the term belongs to. However, at the second step, when we extract the children of the terms extracted at the first level, we already know the namespace. Thus we don't have to generate the queries with other namespaces, unlike in the first step.

However, the Parent/Children queries provide us no information on the completeness of the support of SPARQL by a triple store. Both these queries only use the 'where' clause and don't test whether the triple store supports more complicated clauses like 'filter' and 'optional'.
Due to the simplicity of the queries and the fact that they use only one SPARQL construct – 'where' not much difference is seen in the performance of the triple stores. The difference is of the order of a few seconds(<2s) and even that gets reduced when the cache gets warm.

Conclusion

The above results indicate a superior performance of native stores like Sesame native, Mulgara and Virtuoso. This is in coherence with the current emphasis on development of native stores since their performance can be optimized for RDF. However, these native triple stores especially Mulgara and Virtuoso are constrained by the absence of an API. For certain use cases in Bioportal, presence of an API is necessary. Further, Table 2 and

Table 3 point out that the inability of Mulgara and Virtuoso to load large datasets (>1M triples). It is important to note that the problem is not of lack of scalability with the two mentioned triple stores, but the inability of the third party APIs to handle large datasets. These APIs work perfectly fine when used to access and query existing stores but throw up memory errors when used to load the data through them. Both Mulgara and Virtuoso are able to load the same datasets without any problems when used through their loaders (scripts or command line utilities like load). The documentation of both Mulgara and Virtuoso notes that the preferred way to load large datasets is through the supplied loaders instead of the third party APIs.

Tables 1,2 and 3 indicate that Sesame native gives the best performance of the 4 triple stores being compared. In addition to this, Sesame native comes with an inherent API and thus doesn't suffer from the same problem as the other 2 native stores Mulgara and Virtuoso do. However, Sesame native was only tested with only 10 Million triples. This size currently suffices the need of Bioportal. However[Give reference] mentions that the largest dataset known to be loaded into Sesame native is only 50Million triples, both Mulgara and Virtuoso are said to be lot more scalable than this. This can be a serious limitation when the dataset becomes in-order of a few billions.

Jena SDB is able to load all the three datasets but is the slowest. This was expected since SDB is simply a loader which relies on third party relational databases which are formatted by it before loading the data. It is unable to be optimized for a particular database.

A hybrid Approach

As noted above native triple stores provide a superior performance but are hampered from the inability of third party APIs to load large datasets into them. Sesame's native store gives a performance comparable to the other native stores with an inherent in-house API, however there are doubts about its scalability beyond 50 Million triples. Keeping these issues in mind we suggest a hybrid approach which combines the best of native triple stores and fine level access provided by triple stores with in-built APIs.

In the hybrid approach, the ontologies are first loaded and preprocessed in an in-memory RDF model using a popular API like Jena or Sesame. In the preprocessing step the namespaces are extracted and inferencing using the API is done. These inferenced ontologies are then written out to disk. These inferenced ontologies are read from disk and loaded into a native triple store like Mulgara using the load scripts provided by the triple store and without use of any API. Figure 2 illustrates this process.

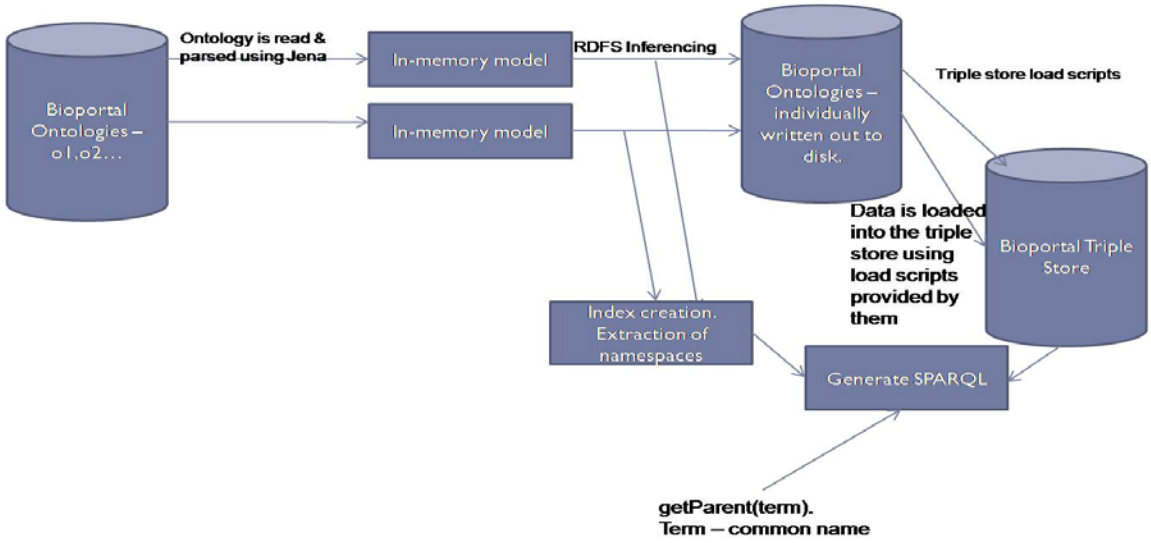


Figure 2

Results using the hybrid Approach

The total time to load the data using the hybrid approach was approximately 13 minutes. Out of this, Jena API took 3.65 minutes to do the preprocessing (inferencing and the extraction of namespaces). The time taken to load the data into Mulgara using load scripts supplied by Mulgara was approx. 9 minutes. This is comparable to the time taken by Mulgara in Table 1, but with the added benefit of inferencing and extraction of namespaces.