RESTful Triple Space Management of Cloud Architectures

Antonio Garrote Hernández and María N. Moreno García

Universidad de Salamanca
agarrote@usal.es, mmg@usal.es

Abstract. In this paper we present a job coordination service for distributed applications being executed in a Hadoop cluster based on the use of a RDF backed triple space and a RESTful web services interface. The system provides an efficient and simple coordination mechanism to resolve data dependencies between applications and it can be used at the same time as a rich source of information about the state and activity of the cluster that can be processed to build additional services and resources.

1 Introduction

Cloud infrastructure and distributed data processing frameworks like Apache's Hadoop[1], where storage and processing capacity can be dynamically expanded, have made it easier to build applications capable of processing sheer amounts of data in a scalable way. These applications are built from a large collection of distributed applications that must coordinate their execution in order to process incoming data usually stored as plain data files in a distributed file system that can be easily processed by Hadoop's line oriented data interface. The final output of this process is refined information that can also be stored in the distributed file system, in a relational database system or any other data store. The drawback of this architecture is the increasing complexity of the application coordination task as well as the increasing number of unstructured data sources that must be tracked and warehoused. In this paper we present a triple space [7] coordination system[2] that uses a RDF graph as a blackboard system with a RESTful [8] interface that applications in

[1] http://hadoop.apache.org/

[2] http://github.com/antoniogarrote/clusterspace

L. Uden et al. (Eds.): 7th International Conference on KMO, AISC 172, pp. 571–579.
springerlink.com

the cluster can use to track their data dependencies and publish application's execution state information using using HTTP standard methods to assert and retract this information from the RDF graph. The use of RDF and a shared ontology provides a snapshot of the distributed system execution that can be queried and processed to provide higher level cluster management services on top of the basic coordination system.

1.1 Problem Description

Complex data processing scenarios on a distributed system using frameworks like Apache's Hadoop often involve building a graph of individual applications with data dependencies among them. Raw data is inserted on the boundaries of the application graph and then loaded into the HDFS file system, either periodically or on demand. These data are processed by different applications following a data path through the application graph where intermediary data outputs are stored in temporary locations of the HDFS file system. Finally the multiple possible outputs generated must also be stored in the distributed file system. Managing this complex network of data dependencies is the main goal of the system presented in this paper, this task is accomplished capturing the state of the distributed application graph as RDF graph available through a HTTP RESTful interface. Managing dependencies is only one aspect to be considered in distributed data processing systems. Errors in application execution must be tracked and all kind of notifications must be generated, for example, notifying users that the output of a data processing path has finished its computation. The system here described addresses this problem using the RDF graph as a triple space, a blackboard style system where queries over the graph can be registered and will be automatically triggered where certain conditions are met. Distributed systems must also be easy to extend. Applications must be as loosely coupled as possible, so the configuration of the data processing graph can be rearranged, adding new applications or removing existing ones. The use of a plain HTTP interface, where RESTful semantics for HTTP methods are observed, along with the extensible capabilities of RDF make it possible for applications in the data processing graph to only share a small RDF ontology to describe data dependencies while additional RDF terms can be used by groups of application for more complex coordination tasks specific to that subset of applications. Finally, an important problem in distributed data processing systems is managing the knowledge generated and stored in the system. Big distributed systems evolve along time with different teams building new applications, adding data sources and consuming output from other applications whose authors might be unaware of this fact. Using a single RDF graph as a coordination mechanism allows to capture an important portion of the implicit knowledge about the system, like data sources, data dependencies, information not being recently accessed, etc. All this information can be used to build a layer of high level services on top of the coordination system that can be used to effectively manage the execution of the data processing graph.

2 Components

The term "cloud architecture" [3] is a broad term that can refer to very different systems. In this section we will narrow its scope describing in a precise way the different components in the architecture where the triple space coordination system is embedded and the additional services built on top of it. The figure 1 shows the relation between these building blocks.

2.1 Distributed File System

The distributed file system is the main data repository in the cluster, available for all applications being executed. In our case Hadoop Distributed File System (HDFS)[3] [4] is the underlying implementation used. HDFS offers a reliable storage mechanism implementing tasks like data replication and distribution. It also offers an scalable solution whose storage capacity can be expanded as required adding more nodes to the cluster.

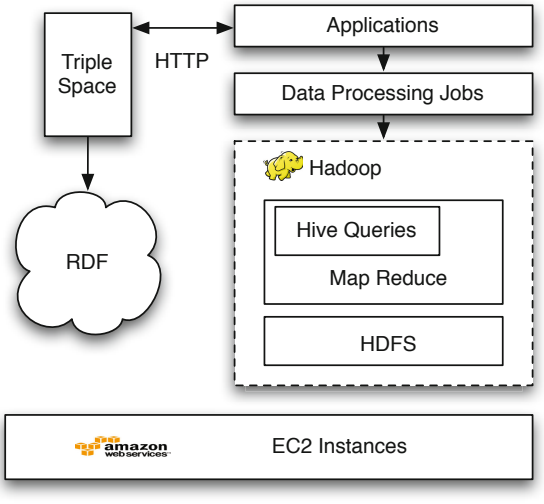

Fig. 1 Components of the distributed system and how they are related to the triple space coordination service

2.2 Data Assets

Every data unit stored in the cluster that is available for applications being executed is referred to as a data asset. The most generic data asset in the cluster are files stored

[3] http://hadoop.apache.org/hdfs/

in the distributed file system. These files are imported as raw unstructured data by applications and then transformed into different data assets.

2.3 Data Processing Jobs

Data processing jobs are groups of batch operations being executed by applications in the cluster, transforming input data assets and generating new data assets as results. In our case, data processing jobs consist of Hadoop map-reduce [5] jobs processing files available in the Hadoop distributed file system. Job operations with a higher level of abstraction, like Apache Hive's[4] SQL-like queries that are finally executed as a collection of map-reduce jobs are also counted as data processing jobs. Data processing jobs may have dependencies on other jobs that must be met before the job can begin its execution.

2.4 Applications

Applications designate every unit of business logic being executed in the cluster. The main task performed by applications is the execution of data processing jobs. Applications in our model are autonomous and direct communication between applications is avoided. However, applications are addressable using the HTTP protocol by the triple space coordination service. The triple space coordination service makes use of the HTTP protocol to notify applications when a certain event in the triple space have taken place. These notifications can be used to trigger data processing jobs when their dependencies are met. Additionally, applications have an associated state that is maintained by the triple space coordination service.

3 Triple Space Coordination Service

The triple space coordination service is an implementation of a blackboard style tuple space communication system backed by a RDF graph and accessible through a RESTful HTTP interfaces mapping classic tuple space operations [8] to HTTP protocol methods. Applications in the cluster can request three main operations in the coordination service:

- Application registration
- Assertion of data as RDF triples
- Data hook registration

Applications must first register into the service providing some application details as structured RDF information, like the accessible location of the application as an URL or the repository where the application source code is stored. This application information is transformed into a set of RDF triple assertions and then added to the

[4] http://hive.apache.org/

RDF graph. Once registration is finished, the RDF sub-graph encoding the application information will be accessible as an HTTP resource that can be read or updated by applications in the cluster using regular HTTP verbs for resource manipulation.

During its execution, applications may add arbitrary new data assertions to the RDF graph mapped as HTTP resources. Additionally, applications can add RDF assertions to the triple space using specific HTTP entry points for data assets. Triples added through the data assets HTTP interface are validated by the triple space service and additional information is added, including time-stamps, RDF type assertions and a updated execution state predicate. Applications in the cluster can obtain the RDF graph associated with a data asset dereferencing the associated resource URI. Applications updating any resource in the HTTP API or adding assertions to the triple space RDF graph must identify themselves using the HTTP User-Agent header and the URI of the API HTTP resource created by the application when registering into the service. This information is used by the service to track changes on the RDF graph.

The last kind of triple space operation is the registration of data hooks. Data hooks make possible for applications to be notified by the triple space coordination system when changes in the RDF data graph occur. Data hooks are created as a different kind of HTTP resource in the triple space HTTP interface. They are described using a small ontology including predicates to specify a SPARQL [6] query and a callback URL. The triple space service registers the SPARQL query associated to every data hook created in the system and evaluates the query every time new data added to the RDF query may change the result set returned by the query. When the query is evaluated results are sent back to the application using a HTTP PUT request to the callback URL associated to the data hook. The application can use this notification to trigger a data processing being hold awaiting for its dependencies to be fulfilled or to update its internal state.

3.1 Cluster Ontology and RDF Encoding

Communication between applications and the triple space consist in the exchange of two types of RDF graphs: RDF data encoding the representation of the HTTP API services like applications, data hooks and data assets, and arbitrary RDF triples added by applications to the coordination service graph. In the case of RDF data containing the representation of a HTTP API resource, a shared ontology is used by applications to ensure easy inter-operability. This ontology includes a small set of around 20 RDF properties used to describe certain types of information:

- State of applications, resources, etc.
- Distributed file system paths.
- Time format, creation and update time-stamps.
- Resource RDF types.

Correct use of this vocabulary is enforced by the triple space coordination service running validations on the received RDF data. This vocabulary is also extended by

certain types of applications and resources. For example, data assets consisting of Hive tables declare and additional sub-type of the `cs:DataAsset` RDF type and information about the location of the HDFS file, table name and Hive partitions.

On the other hand, use of differnt RDF ontologies is possible on arbitrary RDF data added to the graph by applications. This allows for subsets of applications to coordinate their execution through the triple space service and to extend the coordination service through the use of new RDF vocabularies.

RDF data in HTTP requests and responses are encoded using JSON-LD. JSON is a popular format for web APIs and in contrast with other RDF serializations, numerous libraries supporting the format exist for most programming languages. It also makes it possible for application developers using the HTTP API of the coordination service to interact in most cases with the service using familiar JSON objects without having to deal with the complexity of the RDF data model.

3.2 Implementation Details

The triple space service is implemented as a JavaScript application being executed in the Node.js V8 JavaScript platform[5]. Node.js offers an excellent platform for developing scalable web services thanks to its implementation of an asynchronous evented HTTP server. Specific support for Node.js is also available in different cloud infrastructure providers.

RDF and SPARQL implementations are provided by RDFStore-js[6], a pure JavaScript implementation of a RDF and SPARQL database we have developed. This library provides an events API for RDF graphs that is used to by the coordination service to perform the evaluation of SPARQL queries associated to data-hooks. RDFStore-js can also be executed in client applications using node, Java applications using Mozilla's Rhino JavaScript implementation as well as in the browser. This offers a convenient mechanism for client applications to process RDF data received from the coordination service when processing JSON-LD [1] encoded RDF as plain JSON objects is not enough for their functionality.

In the triple store coordination service the underlying persistent storage layer for RDFStore-js is a MongoDB[7] cluster where RDF triple assertions are stored as JSON documents. A replica set of two MongoDB instances to provide better availability to the system on the event of the coordination service node failure.

4 Higher Level Services

The RDF graph maintained by the triple space coordination service can be understood as a snapshot of the cluster state at any given moment of time, including

[5] http://nodejs.org/

[6] http://github.com/antoniogarrote/rdfstore-js

[7] http://www.mongodb.org/

information such as available data sources, the dependency graph between applications and data assets or cluster health information. The use of an extensible semantic technology like RDF makes it possible to store this information provide an makes it possible to develop an additional layer of services processing this information implementing higher level cluster coordination services.

4.1 Application Monitoring Service

The application monitoring service registers a data hook in the triples space requesting the value of the `cs:state` property for all `cs:Application` RDF resources registered in the graph. Every time an application associated to one of these resources is updated, the application monitoring service is notified by the triple space coordination application with the updated list of states for the applications in the cluster. The application monitoring service uses this information to provide a web interface where the state of cluster applications can be browsed, implementing functionality like email alerts or the required logic to restart failed applications.

4.2 Hive Data Catalog Service

`cs:HiveAssets` RDF resources are specialized data assets storing information about Hive tables and partitions. They also include temporal information about the date when a particular Hive table and partition have been modified. New tables being created and partitions being removed are also translated into HTTP requests adding and removing RDF assertions.

The Hive data catalog application registers a data hook in the coordination service consisting of a SPARQL query requesting the value in the graph of the `cs:tableName` RDF property associated to `cs:HiveAssets` whose state is updated by an application at a given time-stamp. Using this information as well as Hive's data definition language, information for those tables is retrieved and stored.

This information is used by the Hive data catalog service to build a description of the Hive tables available in the system and offer a web interface to users where the available tables as well as the applications modifying each table can be browsed. The information can also be used to determine applications affected by administrative changes in Hive tables as well as to look for data tables no longer being used.

4.3 Cluster Log Service

The cluster log service is a module of the triple space coordination service logging individual assertion and removal of RDF triples in the RDF graph. Every time an application request triggers the insertion or deletion of RDF data from the graph, the RDF data is serialized and logged to HDFS using Apache's Flume[8]. The format

[8] https://cwiki.apache.org/FLUME/

of the logged data used is a variation of the Turtle syntax [2] where a time-stamp and the URI identifying the application in the User-Agent of the HTTP triggering the RDF graph modification are added. This line oriented log format makes it easier the log processing using map reduce Hadoop jobs. This information is used to audit the performance of the cluster and compute usage statistics.

5 Conclusions and Future Work

In this paper we have presented a coordination service for distributed data processing applications based on the use of RDF and RESTful HTTP services. The main goal of the service is to coordinate the execution of data driven application work flows. At the same time, it can be used as an structured data source about the state of the cluster, including application's execution state, functional dependencies between applications and data sources. This information can be used to implement higher level services like the data catalog and application monitoring services.

The extensible design of RDF makes it possible to add new functionality to the system using additional RDF vocabularies describing new types of data sources or coordination primitives shared only by a subset of applications. Technologies used to implement the service serve a double purpose, they are based in platforms like Node.js and MongoDB well supported by cloud infrastructure providers and at the same time try to offer a simple interface to the RDF system for application developers using standard RESTful design and a JSON based data exchange format like JSON-LD.

Future lines of work include the integration of additional information into the RDF graph, specially additional data sources beside HDFS data, like relational databases as well as configuration and deployment information. This information can be retrieved from the source code repository information of the registered applications already stored in the RDF graph. The ultimate goal of this job would be to track data being processed in the cluster as higher level data entities being transformed despite their multiple possible representations, from text files stored in HDFS to tables in a relational database or documents in a MongoDB repository. Another area of research is the processing of the cluster log information being generated by the coordination service. This log consist of a stream of time-stamped RDF assertions describing the temporal evolution of the cluster configuration stored in the coordination service RDF graph in a modified Turtle RDF encoding. This source of temporal data can be processed to compute a number of valuable metrics on the historical performance of the system.

References

1. Sporny, K.: SON-LD Syntax 1.0, A Context-based JSON Serialization for Linking Data, online specification (2012)
2. Beckett, B.-L., Prud'hommeaux: Turtle Terse RDF Triple Language. W3C Working Draft (2012)

3. Kim, W.: Cloud architecture: a preliminary look. In: Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia. ACM (2011)
4. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The Hadoop Distributed File System, Yahoo. In: IEEE 26th Symposium on Mass Storage Systems and Technologies (2010)
5. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Google, Communications of the ACM (2008)
6. Prud'hommeaux, Seaborne: SPARQL Query Language for RDF. W3C Recommendation (2008)
7. Fensel, D.: Triple-Space Computing: Semantic Web Services Based on Persistent Publication of Information. In: Aagesen, F.A., Anutariya, C., Wuwongse, V. (eds.) INTELLCOMM 2004. LNCS, vol. 3283, pp. 43–53. Springer, Heidelberg (2004)
8. Fielding, R.T.: Architectural styles and the design of network-based software architectures. PhD. Thesis, University of California, Irvine (2000)