Clood CBR: towards microservices oriented case-based reasoning.

NKISI-ORJI, I., WIRATUNGA, N., PALIHAWADANA, C., RECIO-GARCIA, J.A. and CORSAR, D.

2020

Clood CBR: Towards Microservices Oriented Case-Based Reasoning

Ikechukwu Nkisi-Orji[1], Nirmalie Wiratunga[1], Chamath Palihawadana[1], Juan A. Recio-Garcia[2⋆], and David Corsar[1]

[1] School of Computing Science and Digital Media, Robert Gordon University, Aberdeen AB10 7GJ, Scotland, UK
{i.o.nkisi-orji,n.wiratunga,c.palihawadana,d.corsar1}@rgu.ac.uk
[2] Department of Software Engineering and Artificial Intelligence, Universidad Compluetnse Madrid
jareciog@fdi.ucm.es

Abstract. CBR applications have been deployed in a wide range of sectors, from pharmaceuticals; to defence and aerospace to IoT and transportation, to poetry and music generation; for example. However, a majority of these have been built using monolithic architectures which impose size and complexity constraints. As such these applications have a barrier to adopting new technologies and remain prohibitively expensive in both time and cost because changes in frameworks or languages affect the application directly. To address this challenge, we introduce a distributed and highly scalable generic CBR system, Clood, which is based on a microservices architecture. This splits the application into a set of smaller, interconnected services that scale to meet varying demands. Microservices are cloud-native architectures and with the rapid increase in cloud-computing adoption, it is timely for the CBR community to have access to such a framework.

Keywords: Cloud CBR, Mircoservices, Elasticsearch, CBR framework

1 Introduction

Several case-based reasoning (CBR) development frameworks and toolkits have been introduced to the CBR community [12–14]. These have been extended for recommender systems [7] and textual CBR [11] and more recently for self-management systems [1]. However many of these CBR systems are mostly implemented with monolithic architectures such as desktop standalone applications, with heavy demands due to siloed in-memory batch processing. This is not compatible with recent software development trends, which are increasingly using REST APIs for communication with cloud computing platforms.

Cloud computing is a term used to describe the use of remote hardware and software to deliver on-demand computing services through a network (usually the Internet). In the past, applications or programs were run from software

⋆ Supported by the Spanish Committee of Economy and Competitiveness (TIN2017-87330-R) downloaded on to a physical computer or server. In contrast cloud computing lets users access these applications through the internet. Implementing software applications in the cloud offer several benefits which include efficient/cost reduction, scalability, mobility, and disaster recovery. Distribution of CBR applications and cases enables, MapReduce type algorithms to exploit the parallelism opportunity that is to be had with pair-wise similarity computations [18]. Interestingly, CBR has also been applied to support cloud provisioning, whereby similar Amazon Web Services (AWS) [3] configurations are recommended given a characterisation of a user's compute task [8]. This helps the user to make decisions about the types of cloud services for the given task. But having to monitor resource utilisation and change service requirements accordingly is a challenge which in turn has paved the way for microservice based architectures.

A CBR framework using a microservice based architecture provides (amongst other things) flexibility in both the technology being used (e.g. programming language) as well as dynamic scalability that can adapt to user application demands (e.g. spikes in casebase querying, seasonal effects). This is because, individual microservices are independently scaled and developed such that the overall system architecture is a scalable distributed application [5]. Importantly, the computation of services are stateless since they are automatically provisioned only when needed and then stopped when no longer required. This is particularly advantageous to CBR in situations where there is in-memory demand due to its inherent nature of being a lazy learner.

In this paper we discuss how the CBR cyle can be organised into multiple microservices and how service discovery is facilitated between these independent components using rest communications. A microservice is considered efficient when the system is loosely coupled and highly cohesive [9]. Identifying which functionalities within the CBR cycle should be decoupled and organising them into microservices is a key design challenge that we address in this paper. We do this by introducing, Clood, a generic open-source CBR cloud-based microservice framework, and make the following key contributions:

- create a novel design using the microservice paradigm for CBR;
- introduce, Clood, an extensible open source microservice CBR framework[4];
- evaluate the scalability of the retrieval phase on a recommender task; and
- identify areas of future development that are essential for the sustainability of Clood CBR.

Rest of the paper is organised as follows; in Section 2 we discuss existing frameworks, jcolibri and *myCBR*. The design paradigm appears in Section 3 and the Clood implementation is discussed in Section 4. Results from a scalability experiment with half a million cases is presented in Section 5 followed by conclusions and future directions in Section 6.

[3] https://aws.amazon.com/

[4] Clood CBR repository: `https://github.com/RGU-Computing/clood`

2 Related Work in CBR Development Architectures

There are two well-established open-source frameworks for building CBR applications: *myCBR* and colibri, though they follow different approaches and support different phases of the CBR application development.

myCBR has been a tool for researchers and practitioners over the last ten years [15]. This framework is focused on the developing of a knowledge model for representing cases and computing similarity through the myCBR-workbench tool [2]. This knowledge model can be instantiated through the building blocks and functionality provided by the myCBR-SDK, that is a Java library following a classical monolithic software architecture. However, their authors have recently presented the *myCBR* Rest API which exposes the functionality of both myCBR-SDK and myCBR-workbench through a RESTful API [1]. Instead of forcing users to integrate their *myCBR* systems into a Java environment, this novel API enables users to model a CBR system using myCBR's workbench and then deploying the application as a web service. The goal is to make it easier to build, test, compare and deploy CBR applications.

colibri, on the other hand, is focused on the development of a wide range of CBR applications [10]. As a platform, colibri offers a well defined architecture for designing CBR systems, a reference implementation of that architecture: the jcolibri framework [12], and several development tools that aid users in the implementation and sharing of new CBR systems and components. These tools have been integrated in the colibri *Studio* development environment [13]. Both tools make up the COLIBRI platform following a two layer architecture. jcolibri is the white-box layer of the architecture: a framework for developing CBR applications in Java. This framework represents the bottom layer of the platform. It includes most of the code required to implement a wide collection of CBR systems: Standard, Textual, Knowledge-Intensive, Data-Intensive, Recommender Systems, and Distributed CBR applications. It also includes evaluation, maintenance and casebase visualisation tools. All this functionality has established jcolibri as a reference CBR framework with more than 35K downloads. However, jcolibri still follows the same monolithic Java architecture like *myCBR* and is not suitable for modern web environments.

The need for both these platforms to evolve into web services architecture is clear. However, there are different approaches to implement this evolution. *myCBR* proposes wrapping its existing java components as web services. It is a straightforward option but has several drawbacks. Mostly, the wrapping of the existing java components does not allow to take advantage of the capabilities of cloud architectures regarding availability or scalability. The alternative option is to create a cloud-based CBR framework from scratch in order to exploit the features of modern cloud architectures. This is the option adopted by Clood, that can be considered as a re-implementation of the functionalities provided by the jcolibri and *myCBR* frameworks, but instead of wrapping its existing java components, it redesigns entirely the CBR architecture for the cloud. In this manner, Clood adopts the CBR architecture defined in colibri based on a pre/post-cycle to load/release required resources. Clood also reproduces the case structure representation based on a composite pattern, and the similarity computation through global/local similarity functions that both jCOLIBRI and *myCBR* implement.

In summary, our goal is to create a cloud architecture that is able to provide the same functionalities using familiar methods currently being used in jCOLIBRI and, thereafter, further integrate existing web services found in *myCBR*. As we will present in the following section, CLOOD re-implements jCOLIBRI's methods using modern web services technologies such as Elasticsearch or JSON-based communications that extend the existing capabilities of the framework regarding flexibility and data-intensive processing.

3 Microservices Design Paradigm for CBR

A microservice is an independent process which can carry out specific tasks in isolation [5]. These should be deployed, tested and scaled independently for a single functional responsibility; such as similarity, ranking, casebase editing, etc. Key to this architecture are the concept of serverless functions also referred to as *Function-as-a-Service* (FaaS)[3] - logic that is split into small code snippets and executed in a managed compute service. Well known examples include AWS's Lambda[5] and Google's Cloud Functions[6].

3.1 Clood Architecture

Figure 1 shows a high-level overview of the system's design consisting of 3 components: REST API; Serverless functions; and the ElasticSearch (ES) service. The core CBR tasks – retrieve, reuse, revise, retain – are implemented as serverless computing functions. Functions can interact with external interfaces (e.g. dashboard) and internally with other functions through REST APIs. Decomposition of the CBR cycle into smaller functions provides flexibility to introduce similarity functions and deploy them independently. Such functions will also include relevant knowledge container provisions. The post-cycle or maintenance tasks, like forgetting cases or recomputing footprint cases can be confined to the Retain service. Other management services for CLOOD project management and configurations are also included as microservices accessible through REST. The ES service is used as the casebase which allows the serverless functions to query and retrieve. Data sources and connectors forming the pre-cycle communicate with the Casebase once they are synced with the ES. Data source's can either be external or within the cloud platform which gives flexibility for the community to use existing data sources. An important distinction here with the pre-cycle is that it remains lean (as compared to jCOLIBRI, or *myCBR*); in that it does not involve loading cases into memory once cases are made persistent.

[5] `https://aws.amazon.com/lambda/`
[6] `https://cloud.google.com/functions`

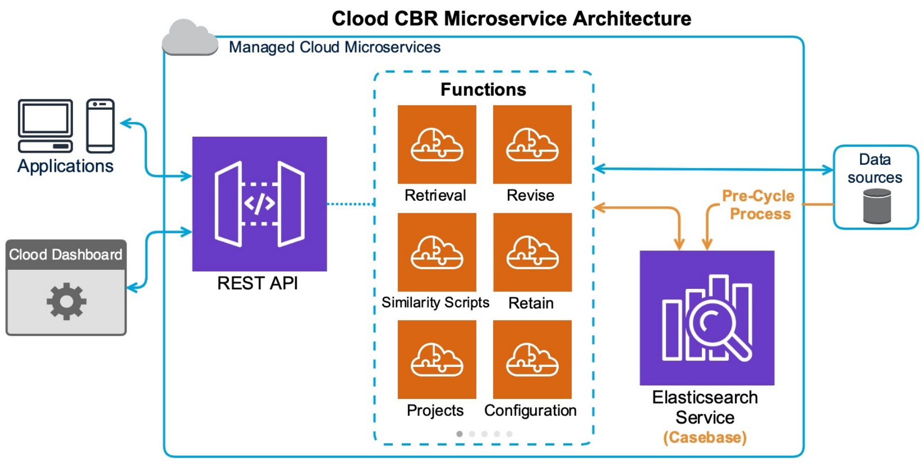


Fig. 1. Proposed Clood CBR Architecture diagram

3.2 The Casebase

Popular CBR systems like jCOLIBRI keep the casebase in memory during operation. An in-memory casebase guarantees speed when interacting with the casebase but will incur massive costs to scale up for big data. Also, using the CBR system in a distributed manner can be problematic with in-memory casebase as memory is an expensive resource even on the cloud.

In the serverless architecture, we maintain the casebase in a NoSQL full-text distributed search engine for all types of data. ES and Solr are popular examples of such distributed, scalable open-source search tools for textual, numerical, geospatial, structured, and unstructured data. These tools provide a significant improvement regarding the representation of cases in previous CBR frameworks, because the case structure does not need to be fixed. Therefore, the cases in the casebase can have different attributes, and similarity metrics are applied according to each particular data types.

Moreover, as these search tools are built on Apache Lucene, they are extendable, allowing users to write custom similarity metric scripts against a data index. Accordingly, the type of operations that would normally occur in-memory can be done in the data store index which is usually file-based[7]. While there are several databases with search capability to choose from, we focus on ES because of its popularity and close integration with existing cloud service providers.

3.3 Local Similarity

A subset of the serverless functions are used to generate similarity scripts to measure local similarity. These metric functions perform retrieval from the case-

[7] `https://www.elastic.co/guide/en/elasticsearch/reference/7.6/index-modules-store.html`

base at the attribute level. Each generated similarity function script depends on the data type of the attribute. Supported data types include string, numeric, Boolean, date and object. Similarity metrics to retrieve exact matches are inbuilt in ES or similar services. And custom similarity metrics functions have been implemented to support other local similarity functions that are used for CBR retrieval in the jCOLIBRI and *myCBR* frameworks.

3.4 Global Similarity

The global similarity function which aggregates local similarities determines the order in which cases are retrieved from the casebase and their ranking. Both a weighted and non-weighted form can be used to identify the nearest neighbours and is managed directly by the ES index service. Each local similarity function script is executed in memory, in response to a single query, to obtain the global similarity as a sum. Custom scripts can be created as needed to vary the weights associated with different attributes. These weights can be dynamically modified for each retrieval task or alternatively remain static for all queries. The latter corresponds to learning a attribute weighting scheme that is used unchanged with every casebase query; whilst the former provides the opportunity to change attribute weights to suit the query context. The default global aggregation can be replaced with a custom aggregation script; whilst this does offer greater flexibility it will also incur greater computing memory when working with medium to large casebases since all the cases that are returned by the local functions will be held in memory (as with the monolithic organisation of jCOLIBRI and *myCBR*).

3.5 Implication for CBR Cycle

The major improvement over the architectures used by jCOLIBRI and *myCBR* is the lack of a two-layer persistence strategy. In previous frameworks there is a need to load cases into memory from a persistence media such as a data-base, text file, etc. However, the use of ES services allows to manage cases directly from its internal index.

Absence of the two-layer persistence strategy, has an immediate impact on the application structure because unlike previously where a *precycle* step was needed prior to the CBR cycle itself for loading cases into memory, this is no longer required. However CLOOD maintain the possibility of executing a precycle (or its complementary postcycle) in order to perform additional pre/post-processing of the data, if the CBR system requires it.

Another significant benefit of cloud-based technologies is concurrency, which directly creates the opportunity to execute CBR processes in parallel. This feature is quite limited in current frameworks and is also very relevant in order to parallelise time-consuming algorithms such as kNN or noise removal methods such as BBNR (Blame-based noise reduction), CRR (Conservative Redundancy Removal), RENN (Repeated Edited Nearest Neighbour), RC (Relative Cover), ICF (Iterative Case Filtering), etc.

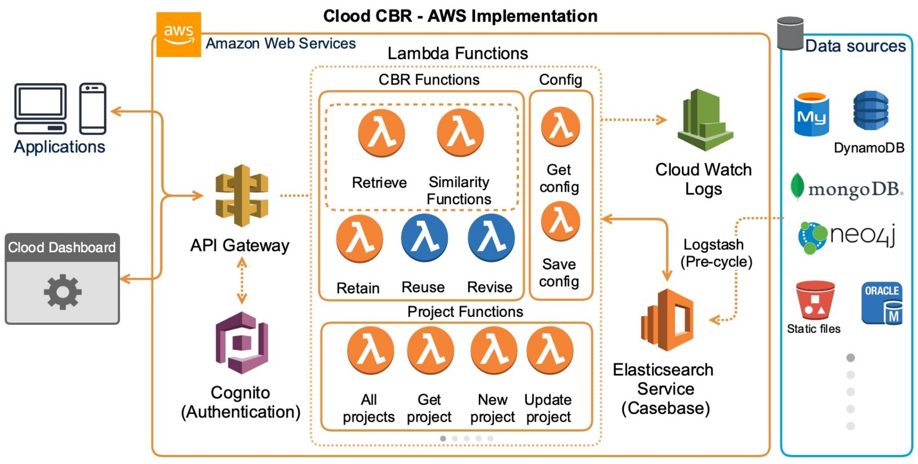


Fig. 2. Clood CBR Implementation on AWS

4 Clood CBR System

Clood is implemented using python functions following the design paradigm presented in Section 3. These functions run on Amazon Web Services (AWS) Lambda, which is the severless event-driven computing service of AWS. The casebase uses the AWS ES service and the client application is implemented with JavaScript and HTML using the AngularJS framework[8]. Using a test application provided by jCOLIBRI[9] we describe the Clood implementation (see Figure 2) and discuss how CBR functionality is achieved with cloud capabilities.

4.1 Casebase using Elasticsearch

ES is an open-source highly distributed and horizontally scalable full-text search engine with various capabilities built on Apache Lucene [6]. ES uses RESTful interfaces to manipulate its schema-free JSON document store and performs searches at very high speeds maintaining an index that is about 20% the size of the indexed documents [17]. Compared to traditional database management systems, the ES "index" is somewhat like the database table as queries are executed against the index. Although it is "schema-free", ES internally generates a schema based on the field (attributes/columns) values of documents to be indexed. Relying on an ES-generated schema can be problematic in some cases. For example, a field for storing alphanumeric values can be designated as numeric by ES if the first documents to be indexed have numeric values only for that field. In order to avoid undesirable field properties, we create an explicit mapping

[8] `https://angularjs.org/`

[9] `http://gaia.fdi.ucm.es/research/colibri/jcolibri/doc/apidocs/es/ucm/fdi/gaia/jcolibri/test/test1/package-summary.html`

Data type	Similarity metric	Description
All	Equal	Similarity based on exact match
String	EqualIgnoreCase	Case-insensitive string matching
	BM25	TF-IDF similarity with TF normalisation based on Okapi BM25 ranking function
	Semantic USE	Similarity based on the similarity of vector representations
Numeric	Interval	Similarity of two numbers inside an interval
	INRECA	Similarity following the INRECA More is Better and Less is Better
	McSherry	Similarity following the McSherry More is Better and Less is Better
Enum	EnumDistance	Similarity of values based on their relative positions within an enumeration
Date	ClosestDate	Similarity depending on the extent two dates are to each other

Table 1. Clood's Local Similarity Metrics

which indicates the data type to be stored for each field in the casebase. The ES index "mapping" is comparable to the database schema as it describes the fields (columns) in the JSON documents along with their data types.

An explicit index mapping supports the specification of how a field's values should be indexed and the local similarity metric to be used for retrieving the values of that field. Where possible, we delay specifying the local similarity function for a field until retrieval time for greater flexibility. This is because the index specification for a field cannot be modified once data is added to the index. With query script similarity functions supplied at retrieval time, the method of retrieval can be varied without having to modify the underlying index mapping. Introducing a new attribute to an existing casebase can be done by extending the index mappings with the new field. The structure of cases that do not have values for newly created fields will remain unchanged. Clood's severless functions interact with ES by HTTP requests and responses using a python Elasticsearch client, elasticsearch-py[10]. The casebase is a separate service which can be hosted anywhere with exposed API end-points further highlighting the distributed nature of Clood.

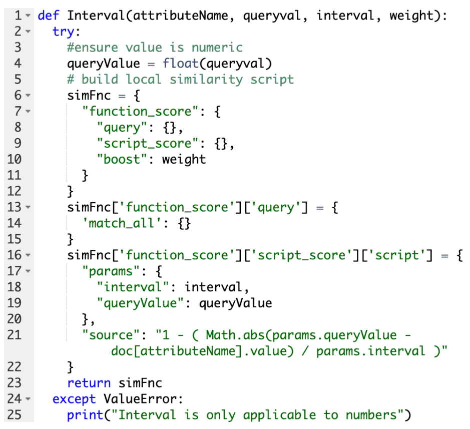
4.2 Clood Similarity Functions

Table 1 shows the local similarity metric functions that are currently implemented on Clood, reproducing some relevant functions available in jcolibri and *myCBR*. Although several similarity metrics are currently missing in Clood,

[10] https://elasticsearch-py.readthedocs.io/en/master/

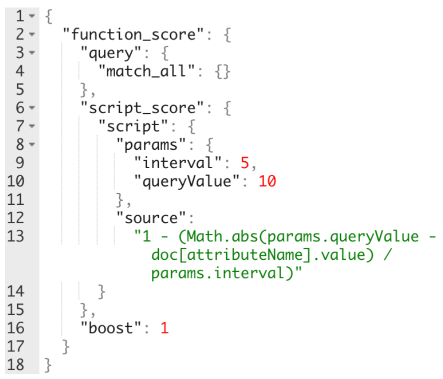
the goal here is to demonstrate the potential of the framework and to encourage code contributions in the future. Each similarity metric is implemented as a python function which generates and returns a *Painless* script. Painless is the scripting language that is specifically designed for writing inline and stored scripts on ES. Alternative languages for writing ES scripts are Java, Lucene expressions language[11], and Mustache template[12].

McSherry, INRECA, Interval and EnumDistance are re-implementations of local similarity metrics found in jcolibri. Figure 3 shows an example of the python code used to dynamically generate scripts for measuring Interval similarity on numeric attributes. Figure 4 is the generated script when the Interval function is called with parameters "queryval" = 10, "interval" = 5, and "weight" = 1. At retrieval, generated scripts for each case attribute, are combined into a single multi-match query script. For textual CBR, we specifically implemented the Semantic local similarity metric (Semantic USE) for text content, using the Universal Sentence Encoder (USE) which embeds texts in a dense vector space of 512 dimensions [4]. This vector representation is generated using a lite version of USE based on the Transformer architecture[13] [16] and is stored as a dense vector field on ES. Textual retrieval follows the same process of generating the vector representation of a query string. Afterwards, the Semantic USE local similarity function measures the cosine similarity between query vectors and documents' vectors to identify the most semantically similar content.



```python
def Interval(attributeName, queryval, interval, weight):
  try:
    #ensure value is numeric
    queryValue = float(queryval)
    # build local similarity script
    simFnc = {
      "function_score": {
        "query": {},
        "script_score": {},
        "boost": weight
      }
    }
    simFnc['function_score']['query'] = {
      'match_all': {}
    }
    simFnc['function_score']['script_score']['script'] = {
      "params": {
        "interval": interval,
        "queryValue": queryValue
      },
      "source": "1 - ( Math.abs(params.queryValue -
          doc[attributeName].value) / params.interval )"
    }
    return simFnc
  except ValueError:
    print("Interval is only applicable to numbers")
```

Fig. 3. Python severless function for generating Interval local similarity script



```json
{
  "function_score": {
    "query": {
      "match_all": {}
    },
    "script_score": {
      "script": {
        "params": {
          "interval": 5,
          "queryValue": 10
        },
        "source":
          "1 - (Math.abs(params.queryValue -
            doc[attributeName].value) /
            params.interval)"
      }
    },
    "boost": 1
  }
}
```

Fig. 4. Generated Painless script for Interval local similarity

[11] https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-scripting-expression.html
[12] https://mustache.github.io/
[13] https://github.com/tensorflow/tfjs-models/tree/master/universal-sentence-encoder

4.3 REST API

REST APIs are stateless in that the API server does not remember the state of its clients and every call to an end-point is independent of other calls. REST API uses existing protocols such as HTTP for Web APIs. As a result, client applications do not need additional software to use the service. REST improves portability to different types of platforms since all interactions are completed through universally understood interfaces. With Clood, each REST API end-point is a serverless function. The replication of an end-point and the resources allocated to it vary to meet changing demands without affecting the other end-points. Table 2 summarises the major REST API end-points of Clood.

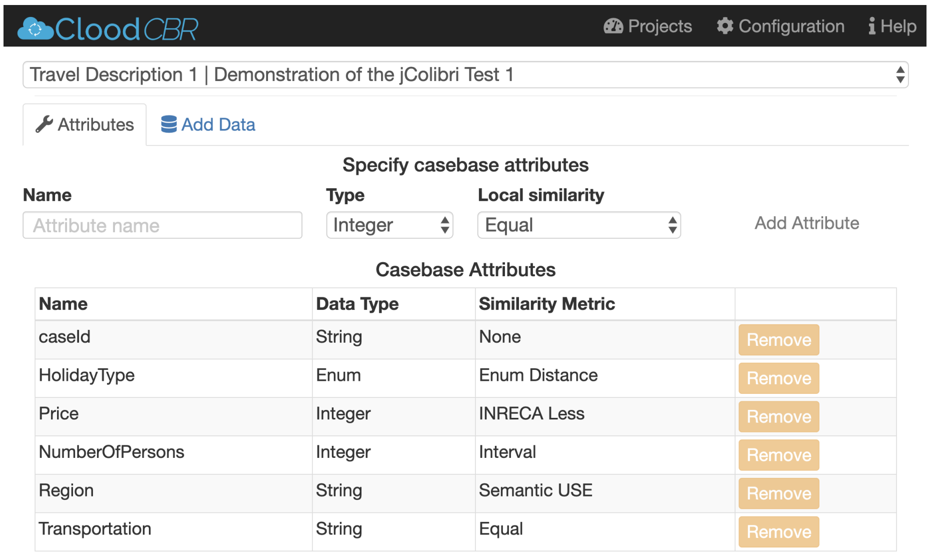
Clood is able to concurrently manage multiple CBR projects (use-cases) referred to as "project" in Table 2. The system's capabilities can be easily extended by introducing new serverless functions (e.g. similarity functions, reuse functions, revise functions). Functions that will become part of the REST API are specified in a YAML file along with their access protocols.

End-point	Request method	Description
/project	HTTP GET	Retrieves all the CBR projects
/project/{id}	HTTP GET	Retrieves a specific CBR project with specified id
/project	HTTP POST	Creates a new CBR project. The details of the project are included as a JSON object in the request body.
/project/{id}	HTTP PUT	Updates the details of a CBR project. Modifications are included as a JSON object in the request body.
/project/{id}	HTTP DELETE	Removes a CBR project with specified id
/case/{id}/list	HTTP POST	Bulk addition of cases to the casebase of the project with specified id. Cases are included in the request body as an array of objects
/retrieve	HTTP POST	Performs the retrieve task.
/retain	HTTP POST	Performs the retain task.
/config	HTTP GET	Retrieves the system configuration.
/config	HTTP POST	Adds or updates the system configuration.

Table 2. Clood's REST API end-points

4.4 Clood CBR Dashboard

Client applications can perform CBR operations through the RESTful API end-points of Clood. The Clood CBR client application is a light-weight HTML and JavaScript implementation that is able to manage multiple CBR projects through API calls. Figure 5 shows the interface for specifying the attributes of a project's casebase. Clood system's configuration provides guidance on allowed operations when specifying attributes. For example, it indicates that the Interval local similarity metric only applies to numeric attributes. Once the attribute specifications are completed, Clood generates an index mapping for the case representation on ES.



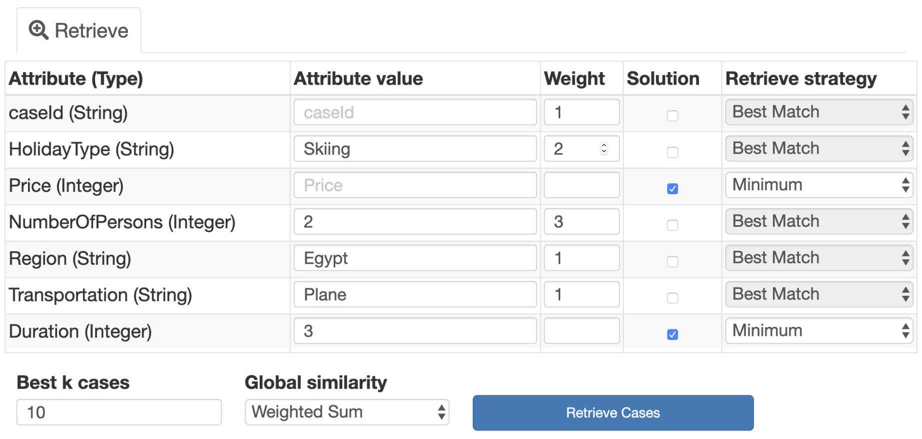
Name	Data Type	Similarity Metric	
caseId	String	None	Remove
HolidayType	Enum	Enum Distance	Remove
Price	Integer	INRECA Less	Remove
NumberOfPersons	Integer	Interval	Remove
Region	String	Semantic USE	Remove
Transportation	String	Equal	Remove

Fig. 5. Specifying attributes for a casebase.

Logstash is an open-source data processing pipeline from the ES stack for ingesting data into ES[14]. Using Logstash, cases can be added to a Clood's casebase from multipel data sources including files (e.g. CSV file), databases with JDBC interfaces (e.g. MySQL, PostgreSQL, Oracle), and NoSQL databases (e.g. MongoDB, CouchDB). However, we also include a file upload utility for adding cases from CSV files through a RESTful end-point and which should be sufficient for file sizes that will not overwhelm the Web browser.

The retrieve operation begins with specifying some attribute values along with weights for aggregating the local similarity measures. Attributes with known values become part of the problem space while attributes with unknown values form the solution space. Furthermore, a retrieve strategy can be specified per attribute as shown on the user interface in Figure 6. For example, the Best match can be retrieved for one attribute while the Mean of the k best matches

[14] `https://www.elastic.co/guide/en/logstash/current/input-plugins.html`



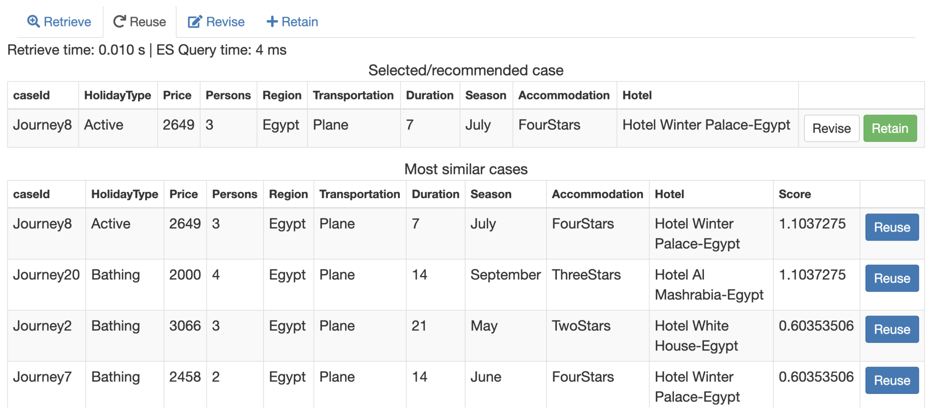
Attribute (Type)	Attribute value	Weight	Solution	Retrieve strategy
caseId (String)	caseId	1	☐	Best Match
HolidayType (String)	Skiing	2	☐	Best Match
Price (Integer)	Price		✓	Minimum
NumberOfPersons (Integer)	2	3	☐	Best Match
Region (String)	Egypt	1	☐	Best Match
Transportation (String)	Plane	1	☐	Best Match
Duration (Integer)	3		✓	Minimum

Best k cases	Global similarity
10	Weighted Sum

Fig. 6. Retrieve stage query specification.

retrieved for another attribute. The k nearest neighbours to retrieve and the global similarity method can also be specified at the retrieve phase.

The reuse interface in Figure 7 displays the retrieval results for reuse. The recommended case (candidate solution) mixes the user-supplied attribute values with the retrieved values for unknown attribute values. The k most similar cases to the query case are also presented for possible reuse. The reuse button against a retrieved case is used to make it the recommended case. The recommended case can be revised by adjusting it as required. Afterwards, the case can be retained by adding to the casebase.



Retrieve time: 0.010 s | ES Query time: 4 ms

Selected/recommended case

caseId	HolidayType	Price	Persons	Region	Transportation	Duration	Season	Accommodation	Hotel	
Journey8	Active	2649	3	Egypt	Plane	7	July	FourStars	Hotel Winter Palace-Egypt	Revise Retain

Most similar cases

caseId	HolidayType	Price	Persons	Region	Transportation	Duration	Season	Accommodation	Hotel	Score	
Journey8	Active	2649	3	Egypt	Plane	7	July	FourStars	Hotel Winter Palace-Egypt	1.1037275	Reuse
Journey20	Bathing	2000	4	Egypt	Plane	14	September	ThreeStars	Hotel Al Mashrabia-Egypt	1.1037275	Reuse
Journey2	Bathing	3066	3	Egypt	Plane	21	May	TwoStars	Hotel White House-Egypt	0.60353506	Reuse
Journey7	Bathing	2458	2	Egypt	Plane	14	June	FourStars	Hotel Winter Palace-Egypt	0.60353506	Reuse

Fig. 7. Reuse stage where k most similar cases are returned.

5 Evaluation

A scalability test is conducted to evaluate Clood based CBR application, to examine how resource demands both on the casebase and the serverless CBR functions are met. We expect a fairly consistent compute performance for different CBR tasks across different project sizes (compared to a jCOLIBRIapplication). We focus on case retrieval for evaluation since it is the most commonly performed and time-consuming stage of the CBR cycle.

5.1 Experimental Setup and Dataset

Six CBR projects of increasing casebase sizes (10, 10^2, 10^3, 10^4, 10^5, and 540, 394) were created from a used cars dataset[15] (1.35GB CSV file), and case retrieval efficiency compared with Clood and jCOLIBRI . A case has 25 attributes describing the physical features of a car (e.g. manufacturer, model, colour), car location (e.g. region, state, coordinates), and the listing price. In the comparative study, 10 nearest neighbours (NN) are retrieved using the following query.

```
{ 'year':'2017', 'manufacturer':'ford', 'model':'focus',
'condition':'good', 'fuel':'gas', 'title_status':'clean',
'transmission':'automatic', 'drive':'4wd',
'size':'compact', 'paint_color':'grey' }
```

Time taken by the Retrieval function (Retrieve time) is recorded which for Clood, consists of: the time spent to dynamically generate a query using the appropriate similarity functions for the query case, retrieve the 10 NN of the query case from the casebase, generate a recommended case for reuse using specified reuse strategy, and generate a response through the API. We do not include the time lapse between the client application and the API endpoints as that is very dependent on the network connection speed and client's platform resources. For jCOLIBRI, Retrieve time is measured in the cycle phase consisting of: the time spent to retrieve the similarity configuration, perform NN scoring over the cases (in-memory), and select the 10 best cases. jCOLIBRI was run on a Windows 10 PC having 6th generation Intel core i7 processor and 16GB RAM with 2GiB Java heap size. Clood uses AWS Lambda functions for its operations while the casebase was hosted on a single cluster of the AWS ES Service with 2GiB and 1 vCPU (t2.small.elasticsearch instance).

5.2 Results and Discussion

Figure 8 shows the average case retrieve times for Clood and jCOLIBRI on log scales with standard deviations as error bars. jCOLIBRI was marginally faster on the smallest casebase (10) but the superior performance of Clood is apparent with increasing casebase sizes. Similar case retrieval times were obtained at at about 100 cases; however at casebase size of 1,000, Clood was 5.5 times faster

[15] `https://www.kaggle.com/austinreese/craigslist-carstrucks-data/data`

than jCOLIBRI and at casebase size of 540,394, CLOOD was 3,737 times faster than jCOLIBRI. Close examination of CLOOD's Retrieve time spent on the ES casebase when measured separately (Query time) shows to have increased due to time spent querying the casebase (see Figure 9).

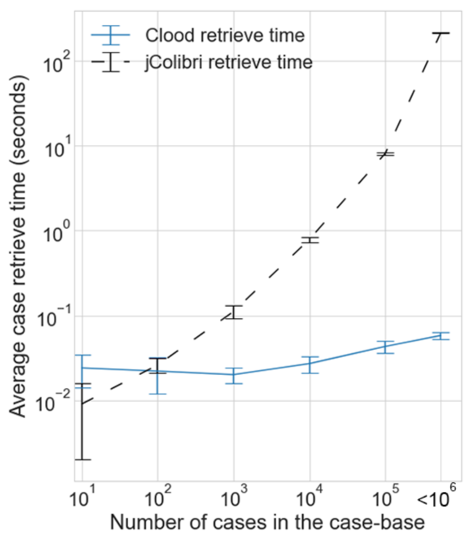


Fig. 8. Case retrieve times as casebase size increases. Both axes are log scales.

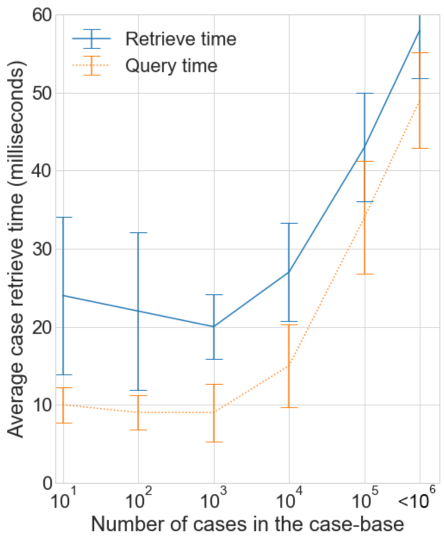


Fig. 9. CLOOD retrieve times compared to the query times. X-axis is a log scale.

6 Conclusion

We introduced CLOOD CBR, a novel microservices-oriented CBR framework which leverages the serverless architecture for CBR operations and a distributed data storage service (Elasticsearch) for CBR knowledge persistence. Implementation of the extensible CLOOD CBR framework is an ongoing opensource project. We demonstrated the robustness of CLOOD on a CBR project of half a million cases and showed how CLOOD is more scalable than existing systems like jCOLIBRI. On-going work is extending support for additional similarity and data types (e.g. *myCBR*'s table similarity and user-defined similarity functions); and include functions for reuse and revise, casebase maintenance and visualisation. In future we aim to make CLOOD a Python library to reuse the CLOOD Elasticsearch similarity functions for the community and to include seamless support for multiple cloud providers.

References

1. Bach, K., Mathisen, B.M., Jaiswal, A.: Demonstrating the mycbr rest api. In: Demo session of the 27th Int. Conf. on CBR (2019)
2. Bach, K., Sauer, C.S., Althoff, K., Roth-Berghofer, T.: Knowledge modelling with the open source tool mycbr. In: KESE@ECAI. CEUR Workshop Proceedings, vol. 1289. CEUR-WS.org (2014)
3. Castro, P., Ishakian, V., Muthusamy, V., Slominski, A.: Serverless programming (function as a service). In: 2017 IEEE 37th Int. Conf. on Distributed Computing Systems (ICDCS). pp. 2658–2659. IEEE (2017)
4. Cer, D., Yang, Y., Kong, S.y., Hua, N., Limtiaco, N., John, R.S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., et al.: Universal sentence encoder. arXiv preprint arXiv:1803.11175 (2018)
5. Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L.: Microservices: yesterday, today, and tomorrow. In: Present and ulterior software engineering, pp. 195–216. Springer (2017)
6. Gormley, C., Tong, Z.: Elasticsearch: the definitive guide: a distributed real-time search and analytics engine. " O'Reilly Media, Inc." (2015)
7. Jorro-Aragoneses, J.L., Recio-Garcia, J.A., Diaz-Agudo, B., Jiménez-Diaz, G.: Recolibry-core: A component-based framework for building recommender systems. Knowledge-Based Systems **182** (2019)
8. Minor, M., Schulte-Zurhausen, E.: Towards process-oriented cloud management with CBR. In: Int. Conf. on CBR (2014)
9. Pahl, C., Jamshidi, P.: Microservices: A systematic mapping study. In: CLOSER (1). pp. 137–146 (2016)
10. Recio-García, J.A., Díaz-Agudo, B., González-Calero, P.A.: The colibri platform: tools, features and working examples. In: Successful CBR Applications-2, pp. 55–85. Springer (2014)
11. Recio-Garcia, J.A., Diaz-Agudo, B., Gómez-Martin, M.A., Wiratunga, N.: Extending jcolibri for textual cbr. In: Proc. 6th Int. Conf. on CBR. vol. 3620, pp. 421–435. Springer (2005)
12. Recio-García, J.A., González-Calero, P.A., Díaz-Agudo, B.: jcolibri2: A framework for building CBR systems. Science of Computer Programming **79**, 126–145 (2014)
13. Recio-Garcia, J.A., González-Calero, P.A., Diaz-Agudo, B.: Template-based design in colibri studio. Inf. Syst. **40**, 168–178 (2014)
14. Roth-Berghofer, T., Recio-Garcia, J.A., Severing-Sauer, C., Althoff, K.D., Diaz-Agudo, B.: Building CBR applications with mycbr and colibri. In: Proc. 17th UK Workshop on CBR. pp. 71–82. University of Brighton (2012)
15. Stahl, A., Roth-Berghofer, T.R.: Rapid prototyping of CBR applications with the open source tool myCBR. In: Proc. 9th European Conf. on CBR. pp. 615–629. Springer-Verlag, Heidelberg (2008)
16. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)
17. Voit, A., Stankus, A., Magomedov, S., Ivanova, I.: Big data processing for full-text search and visualization with elasticsearch. Int. Journal of Advanced Computer Science and Applications **8**(12), 18 (2017)
18. Zhong, Z., Xu, T., Wang, F., Tang, T.: Text CBR framework for fault diagnosis and prediction by cloud computing. Mathematical Problems in Engineering (2018)