Data Quality Assessment and Anomaly Detection Via Map / Reduce and Linked Data: A Case Study in the Medical Domain

Stephen Bonner *, Andrew Stephen McGough *, Ibad Kureshi *, John Brennan *, Georgios Theodoropoulos *,
Laura Moss †, David Corsar ‡ and Grigoris Antoniou §
*School of Engineering and Computing Sciences
Durham University, Durham, UK
{s.a.r.bonner, stephen.mcgough, j.d.brennan, ibad.kureshi, georgios.theodoropoulos}@durham.ac.uk
†University of Glasgow, Glasgow, UK
‡University of Aberdeen, Aberdeen, UK
§University of Huddersfield, Huddersfield UK

***Abstract*—Recent technological advances in modern healthcare have lead to the ability to collect a vast wealth of patient monitoring data. This data can be utilised for patient diagnosis but it also holds the potential for use within medical research. However, these datasets often contain errors which limit their value to medical research, with one study finding error rates ranging from 2.3% – 26.9% in a selection of medical databases.**

Previous methods for automatically assessing data quality normally rely on threshold rules, which are often unable to correctly identify errors, as further complex domain knowledge is required. To combat this, a semantic web based framework has previously been developed to assess the quality of medical data. However, early work, based solely on traditional semantic web technologies, revealed they are either unable or inefficient at scaling to the vast volumes of medical data.

In this paper we present a new method for storing and querying medical RDF datasets using Hadoop Map / Reduce. This approach exploits the inherent parallelism found within RDF datasets and queries, allowing us to scale with both dataset and system size. Unlike previous solutions, this framework uses highly optimised (SPARQL) joining strategies, intelligent data caching and the use of a *super-query* to enable the completion of eight distinct SPARQL lookups, comprising over eighty distinct joins, in only two Map / Reduce iterations. Results are presented comparing both the Jena and a previous Hadoop implementation demonstrating the superior performance of the new methodology. The new method is shown to be five times faster than Jena and twice as fast as the previous approach.

***Keywords*-RDF; Medical Data; Map / Reduce; Joins;**

I. Introduction

A. Big Data and Healthcare

Big data is a rapidly growing area of interest which offers the potential for great advances but also comes with significant challenges. Both the academic and industrial communities are generating and collecting data at an unprecedented rate and scale. Analysis of these datasets represents a huge opportunity to advance domain knowledge and informed decision making [7]. However, despite a few early successes, there remain many significant challenges in realising the full potential of the knowledge buried within these datasets. The volume and variety of the data generate technical challenges in storing and processing. Additionally, assessing the quality of the data – often referred to as the veracity of the data, for example to identify any abnormal artefacts or missing data, can be a daunting but crucial task to ensure that the data remains useful for its intended task.

The biomedical and healthcare fields have the potential to be one of the biggest contributors to, and benefactors from, the big data phenomenon. The increasing digitalization of healthcare has led to our ability to collect and store vast amounts of diverse and complex biomedical data from a plethora of sources and devices. The volume of available data in these fields is increasing exponentially; ranging from structured and unstructured clinical records to data streamed from medical devices and social media. In 2012 the overall volume of worldwide healthcare data was estimated to be 500 petabytes; this figure is predicted to grow 50-fold by 2050 to 25,000 petabytes [1]. Analysis of this data has the potential to lead to advances in medical research, healthcare management, service provision and patient treatment.

Computer science techniques provide a great opportunity to deliver such advances and enable healthcare practitioners to leverage clinical data. However, the size of the generated datasets, coupled with a limited capacity to store and analyse them to extract knowledge, hinders their use within healthcare. Additionally, inaccuracies are introduced due to the nature by which the data is collected and can manifest as unintentional patterns recorded in datasets. In clinical settings, these artefacts are often a result of nursing or medical interventions which are caused by patient movement or by the administration of treatments [2]. It is important to understand and remove such irregularities from the data so that they do not adversely affect interpretation and analysis. Previous studies have estimated that error rates in medical databases can range from 2.3% to 26.9% [12]. Without due consideration, low quality data can lead to sub-standard patient treatment and have substantially adverse affects on research findings. Existing approaches to the identification of anomalous values are based around the identification of

extreme values [3]; however, these are often only based on simplistic threshold checks. Although this goes some way to avoiding certain errors (e.g. physiologically impossible values), detailed domain knowledge is required in order to identify other errors (e.g. cerebral perfusion pressure readings taken from the neuro-intensive care unit that have not been corrected for the position of a patient's head).

Due to the size of the datasets generated, combined with the complexity inherent within the medical domain, manual curation of these datasets is unrealistic and automated approaches to engender a level of quality and trust within the data are required.

B. Assessing Medical Data Quality

Numerous knowledge sets are now being made available for equipment and sensors used to collect data from disperse datasources. While several models have been defined for representing sensor data, the W3C Semantic Sensor Network (SSN) ontology [17] has emerged as 'best practice' for publishing data online about sensors, the observations they generate, and their deployments. Using the SSN ontology, sensor datasets can be published according to Linked Data principles [16] and integrated with relevant data described using the Resource Description Framework (RDF) [18]. This machine interpretable encoding allows machines to access, browse, and reason about this data.

The medical dataset used in this study contains physiological patient data taken from the critical care field and was created by expressing the data against ontologies developed in prior work [4]. Critical care medicine is one of the most technology-led medical domains and sophisticated patient monitoring equipment provides constant patient monitoring. Generating large volumes of high frequency physiological patient data. For example, monitors for vital signs can record data such as arterial blood pressure, central venous pressure and ECG (electrocardiogram) signals. Sampling rates can range from once every couple of minutes (or hours) to waveform quality (many times a second).

Previous work described an approach and supporting framework, written in the Jena toolkit [23], for the identification of errors in the medical dataset outlined above, using linked data and semantic web technologies [4]. This work enhanced the dataset with additional provenance metadata. Including details of the machines used to collect the data, the setting used for the recording and any data exportation processes that may have been performed. Metadata, which would indicate the presence of a physiological condition, was also included. Eight queries were written that utilised the new metadata embedded with the medical data to assess the quality of the medical data. This provenance metadata enables checks for values that are outside of expected normal readings, abnormal reading which could be caused by a specific medical disorder, and missing readings.

While the Jena based SPARQL approach was highly successful in identifying errors in the data, it was unacceptably slow for real world data (taking several hours per patient), and unable to scale to dataset sizes over three million triples – approximate volume of RDF data generated by a patient per day. A Hadoop based approach was created with the aim of replicating the functionality of the Jena approach, but improving speed and scale [5]. This approach achieved both of these objectives, however it required a costly data pre-processing stage and was therefore inefficient.

C. Overview Of Our New Approach

In this paper we present a new approach for assessing the quality of medical data using a novel algorithm written using Map / Reduce for the Hadoop platform. The new approach utilises the inherent distributed nature of Hadoop to scale to dataset sizes of over one billion triples, on a modest eight node cluster constructed from previous generation, Commodity Off-The-Shelf components.

The approach utilises optimal query and join strategies – along with the creation of a *super-query* – made possible by studying the structure of the data and the original SPARQL queries. Traditionally, each join in Hadoop would require a complete Map / Reduce iteration. However, by using these optimisations we are able to complete over 80 joins in only two Map / Reduce iterations. By identifying and extracting a series of triple groups that allows us to safely segregate the data into smaller subsets. Additionally, the identification and distribution to worker nodes (via Hadoop distributed cache) of groups which are joined to frequently – but small in size, allows for multiple joins on different keys to be performed within the same Reduce task – massively reducing the required number of Map / Reduce iterations. The new approach is much faster and more scalable than traditional semantic web technologies; requires no additional data preprocessing and takes as input the raw RDF dataset. By designing our solution to run on a completely vanilla Hadoop install it removes the requirement for the end user to install any additional software packages.

The main contributions of this paper are as follows:

- The idea of creating a super-query, a technique of amalgamating queries which share common join elements, to avoid the needless re-computation of joins.
- A method for extracting subsets of triples from a larger dataset. These subsets are determined by analysis of the original SPARQL queries for triples which share numerous common join keys.
- A broadcast join technique to push the small, but frequently joined to subsets into the main memory of each worker node. Allowing many of the joins to be performed in-memory, without the need for multiple Map / Reduce iterations.
- Demonstrating this approach has better performance than both the Jena and previous Hadoop approach. The

new approach being over five times faster than the Jena approach and twice as fast as our previous approach.

The rest of the paper is organised as follows: Section II presents background and related work. Motivation is presented in Section III. Section IV details our theoretical optimisations before presenting the details of our Hadoop implementation in Section V. Experimental evaluation and results are in Section VI which are used to formulate our conclusions and ideas for future work in Section VII.

II. Background and Related Work

A. Linked Data Technologies

RDF is a W3C standard, specifically designed to express the relationship between resources on the World Wide Web [14]. An RDF Statement comprises three structural components: a Subject (about which the statement is being made), a Predicate (describing the relationship between Subject and Object) and an Object (an attribute of the Subject) [14]. An RDF triple, consisting of (s, p, o), is represented as a labeled graph $s \xrightarrow{p} o$.

The Simple Protocol and RDF Query Language (SPARQL) provides a mechanism for querying RDF data stored and is a W3C standard [8]. SPARQL can be compared to the Structured Query Language (SQL), used to query relational databases, as they share a similar syntax and logic. The normative syntax for a SPARQL query is [13]:

$$SELECT\ ?v1...?vn\ WHERE\ t1...tn$$

with the SELECT *?v1...?vn* component (of the query) representing distinct variables occurring in the input data and the order in which they are to be returned. The WHERE *t1...tn* component forms the Basic Graph Pattern (BGP) and is a series of triple query patterns that must match the graph of any input triples in order to be considered a match.

B. TripleStores

TripleStores are specialist data stores designed to store and query RDF data. They currently face two key challenges: system scalability and generality [21]. The most commonly used TripleStores are based on Relational Database Management Systems (RDBMS) and are only able to run on a single machine. Examples include Jena [23] and RDF-3x [24]. As the size of RDF datasets are growing exponentially, it is unlikely that a single machine will be able to efficiently store or process this *data deluge*.

Due to this scalability issue, recent work has developed TripleStores which are able to scale across multiple machines. Readers are referred to a recent literature review covering developments in the field [25]. Many of the approaches utilise the Apache Hadoop ecosystem or other NoSQL systems to enable data and query parallelism. Systems such as SHARD [19] or HadoopRDF [20], store RDF in the Hadoop Distributed File System (HDFS) and query the data via Map / Reduce tasks. Some systems, such as TrinityRDF [21] or CumulusRDF [22], use Key-Value stores as the basis for storage and querying. While these systems are similar to the approach presented here, none of them utilise the broadcast join method or perform any sort of data analytics before performing the queries.

C. Hadoop and Map / Reduce Joining Strategies

Hadoop is an open source framework for storing and processing of internet-scale data in a distributed manner. Hadoop comprises two main components: the Hadoop Distributed File System (HDFS), used for storing data across a Hadoop cluster and the Map / Reduce programming framework, used to process the data [26]. A standard Map / Reduce pass consists of two distinct phases: a Mapper and a Reducer. The Mapper function processes the input data, performs a user-defined algorithm, then outputs an intermediate set of key / value pairs. These intermediate results are grouped on the key to ensure that all values associated with that key are sent to the same Reduce function. The Reducer function then performs the final processing and outputs the resulting set of key / value pairs [26].

Any system which is designed to complete SPARQL queries must be able to compute data joins. However, performing efficient joins in a Map / Reduce environment is particularly difficult to achieve [10].

Map / Reduce was designed to process large datasets by looking at each element in isolation – processing each sequentially. Thus joining two potentially massive datasets falls well beyond Hadoop's design remit. While joins in Map / Reduce are complicated, several strategies have emerged that make them possible [6]. These join strategies include: Reduce-Side, Broadcast and Cascade joins.

The Reduce-Side join is a simple approach for joining data and is implemented using a complete Map / Reduce iteration. The basic workflow for the Reduce-Side join is as follows: the Map function iterates through all records, tagging them with their source dataset and sets the Map output key as the join key. This will ensure that all values featuring that key are sent to one Reduce function. The Reduce function can then join the required elements and produce the final output. The logic for a Reduce-Side join for two datasets is shown below:

$$(P(a, b) \bowtie Q(b, c)),$$

where P and Q are two separate datasets each with two records (a, & b and b & c) and a single common record – b. The join can be achieved by first iterating through both datasets in the Map function and emitting *b* as the key and *a* as the value for *P*, and *b* as the key and *c* as the value for dataset *Q*. The values would be grouped via the common key and the Reduce function would be presented with the key / value pair of $b(a_n, c_n)$. The reduce function would then join the values together to create the final join result. The

major disadvantage of this method is that all data required for the join will be passed through the Map / Reduce shuffle phase, thus incurring a costly network transfer stage.

While this approach is suitable for joining two elements, joining multiple elements that depend upon the output of previous joins is not possible. However, using a technique called a Cascade join it is possible to process such situations [6]. The Cascade join is a method that allows multiple dependant relations to be joined via Map / Reduce. A Cascade join is effectively a pre-determined set of Reduce-Side joins, processed in an iterative manner [6]. The logic behind the Cascade join can be demonstrated by joining of three datasets:

$$(R(a,b) \bowtie S(b,c) \bowtie T(c,d)).$$

Using a Reduce-Side join, it is possible to join dataset *R* and *S* via the join key *b*:

$$(R(a,b) \bowtie S(b,c) \rightarrow IntermediateResult(a,b,c)).$$

Then by employing a second Reduce-Side join it is possible to join the Intermediate result to dataset *T* via the join key c to create the final result:

$$(IntermediateResult(a,b,c) \bowtie T(c,d) \rightarrow FinalResult(a,b,c,d)).$$

One alternative and less used join approach is the Broadcast join. This can be used if one of the datasets to be joined is of a small enough size, such that it could be stored in memory. A Broadcast join functions by loading the smaller dataset into memory of all machines on which a join will be performed [9]. Each machine can then probe the dataset loaded in-memory to check for possible joins from data either in the Map or Reduce phase. This approach removes the need for costly cascade joins.

III. Motivation and Analysis

This section will explore some of the motivations behind why a new approach is required. We use graph theory methods to analyse both the RDF data and SPARQL queries. We then use the results to aid the creation of the highly optimised query solution detailed in Section IV and V.

A. Analysis Of The RDF Graph Structure

As RDF – here used to store the knowledge sets for sensors, physiological conditions and the collected patient data – is a graph-based data model, network and graph analysis may be performed upon it. Any understanding gained from this analysis can be used to guide optimisation techniques, thus leading to more optimal query execution. In way of motivation for our data segregation approach we analyse one of our anonymised sample datasets.

Let $G = (V, E)$ be a graph which contains all of the RDF triples representing our RDF dataset, where the set of vertices V is a number of RDF subjects or objects and the set of edges E are a number RDF predicates. The graph G comprises RDF triples (s, p, o), with the subject and object being nodes and the predicate being edges $s \xrightarrow{p} o$. This graph will be explored throughout this section.

The RDF dataset used for this analysis contained 2,733,290 unique triples, which resulted in a graph of 595,597 unique vertices and 2,733,290 edges. Analysis of this dataset reveals a number of interesting properties. Using graph theory we can analyse the vertex degrees within the dataset, where a vertex degree value is defined as the number of edges connecting to a particular vertex (subject or object). The degree value d_v of a certain vertex v, provides a quantification of how connected v is within the overall graph. As a graph created from RDF is inherently directional, both the in-degree d_v^{in} and out-degree d_v^{out} should be considered independently. Further the degree distribution is the proportion of vertices $v \in V$ with a certain degree value $d_v = d$. The degree distribution provides an overall view of the level of connectivity within a graph.

Table I illustrates the top ten nodes when ordered by d_v^{in}. With the most connected node being the patient's unique identifier. This is followed by a series of nodes which have identical d_v^{in} values. It is interesting to note that all the nodes with a high d_v^{in} have a low or nonexistent d_v^{out}. This implies that these nodes are RDF objects, which have many unique subjects connecting to them. In the case of the top ten nodes illustrated here these are all *metadata* nodes describing the subject they are related to. We can later exploit this for creating our Broadcast joins.

Table I
Nodes Sorted By d_v^{in}

Node Name	d_v^{in}	d_v^{out}
<Patient/15>	158300	1
<PhysiologicalObservation>	158299	0
<ssn/Observation>	158299	0
<PatientData/Reading>	158299	0
<PhysiologicalObservationValue>	158299	0
<ObservationValue>	158299	0
<PhysiologicalSensorOutput>	158299	0
<SensorOutput>	158299	0
<ObservationCollection>	39536	0
<Timepoint>	39536	0

B. Implications Of Performing SPARQL Queries

In SPARQL variables are represented via a $? < name >$ syntax. A SPARQL query will take the general format of:

$$SELECT\ ?v1...?vn\ WHERE\ t1...tn$$

where *?v1...?vn* represent the variables we wish to return from the query. The WHERE clause defines the the patterns which need to be matched within the dataset. If a variable exists more than once within the WHERE clause then all must be satisfied. If multiple variables exist within the WHERE clause then they will be joined together via their common variable(s). As an example, consider the five patterns –

commonly referred to as Basic Graph Patterns (BGP) – related to $?obs$ in the example SPARQL query (Listing 1). Each $?obs$ variable found in the dataset, would be compared to the five patterns and only if it satisfies all five patterns would it be considered 'complete' and therefore retained. For example $?obs\ ssn : observedProperty\ ?p$ would only match an $?obs$ variable where a triple exists with $?obs$ as the subject, $?p$ as the object and $ssn : observedProperty$ is the predicate. Further complete $?range$ variables would be joined with complete $?obj$ variables if a common $?p$ variable exists between both.

To better visualise the logical processes involved in joining, Figure 1 shows a graph which would complete the sample SPARQL query in Listing 1. In this graph a vertex is either a subject or an object, with the directed edges between any two nodes indicating a predicate between them. The size of a vertex is determined by its total degree value. The graph is coloured to illustrate the various common matching patterns of variables across the query. For example all variables which are joined on the $?obs$ subject are coloured black, whilst variables joined on the $?range$ subject are blue. Key vertices which serve as bridges or joins between the variable groups are coloured in red. Computationally, each edge represents a single join – note that each pattern match is performed as a single join – and all must be completed before the final query logic can be assessed.

Listing 1. An Example SPARQL Query

```
SELECT ?obs ?p ?htime ?max ?value WHERE{
?range a med:AcceptableRange .
?range med:clinicalRangeMax ?max .
?range pd:hasParameter ?p .

?obs a mo:PhysiologicalObservation;
?obs ssn:observedProperty ?p .
?obs ssn:observationResultTime ?time .
?obs pd:atHumanTime ?htime .
?obs ssn:observationResult ?a1

?a1 ssn:hasValue ?a2 .
?a2 pd:readingValue ?value .

FILTER (?value > ?max)}
```

The FILTER clause within the query (Listing 1) is checking if a particular recorded value is above the maximum permitted value by the hardware which recorded it. This clause is only enacted once all the completed variables have been extracted. This whole query process is computationally expensive as the number of attempted joins is a multiple of the number of triples within the dataset being queried. In our running example there would be eleven joins per query. Note that the dotted line in Figure 1 segregates the joins into those for nodes with low d_v^{out} values – which can be performed using in-memory joins – and nodes with high d_v^{out} values – which require Reduce-Side joins.

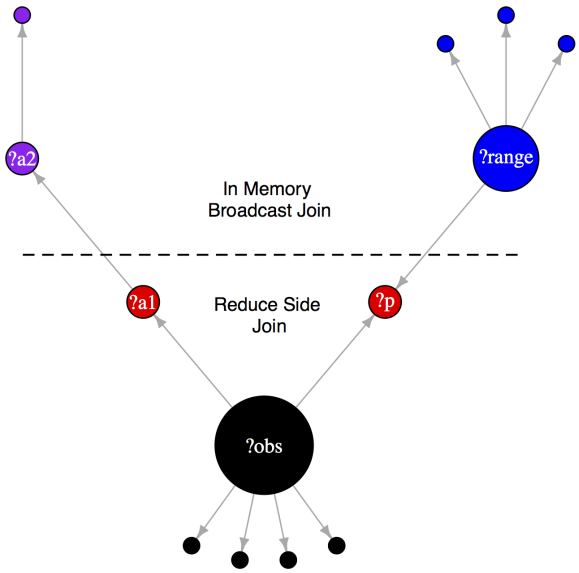

Figure 1. A Graph Of A Single Complete SPARQL query

Although we only present one query here we need to perform many such queries over our dataset. Though many of these queries are very similar in nature. For example, a second query may only vary the FILTER line and associated BGP from Listing 1. However, conventional approaches would perform both queries independently – thus repeating all of the joins.

IV. Theoretical Model

In this section we will explore some of the theoretical ideas underpinning our modified Hadoop based approach. We start with a brief overview of this Hadoop approach. Firstly, as the set of queries share many common elements, a super-query was created to avoid the re-computation of joins common among all the queries. This super-query maps out, for the algorithm, which RDF triples need to be extracted from the larger dataset and how they should be joined together to create the final results. Secondly, the RDF data is segregated into smaller sub-sets based on the triple group size and degree. This enables groups which are small in size, but are joined to frequently, to be broadcast to all the nodes in the Hadoop cluster. Here we quantify that these groups must be small enough in size to fit into the Hadoop distributed cache, which has a default maximum size of 10 Gigabytes. These groups can then be loaded into memory on all the worker nodes and can be joined to the large datasets as needed. This allows multiple joins to be performed within the same Map / Reduce iteration.

A. Generation of Super-Queries

As discussed in section I-B, the original functionality of the Jena based approach comprises eight SPARQL queries

which assess the data quality against several metrics. Many of these queries are almost identical, with only one or two elements changing. For example, the only difference between the query assessing if the value is below or above the values permitted, is one line [15]. Currently each query is run independently against the data. This is highly unoptimised as all of the numerous common joins, between any queries, will be recomputed, as seen in section III-B. A more optimal solution would be to combine the queries so that the join is performed only once. Both the min and max queries could be completed by just adding an extra vertex to the graph in Figure 1, which contains the information for the minimum permitted value. Two separate conditional logic steps could then be computed to complete both queries. This method, effectively the creation of a super-query, would allow completion with the addition of just one extra join.

B. Optimisation of Super-Query Through Segregation

The super-query solves the issue of recomputing identical joins across a set of queries, however it creates a new problem as it is inherently a large and complex SPARQL query. Which would be difficult to complete using Hadoop without further optimisations.

To enable more efficient computation, a large graph representing many RDF triples can be broken down into more manageable sub-graphs. To guide the creation of the sub-graphs, a certain criterion is needed to partition the graph. The criteria chosen for this work is that of triple groups (TG). Our definition of a triple group is a selection of triples taken from the larger dataset which, guided by the SPARQL query, have more than one join via a common element. From the example query in Listing 1, it is possible to visually identify 3 unique triple groups; these being the $?range$, $?obs$ and $?a2$ groups. As an example, the $?range$ group would be a subset of triples which matched any of the patterns beginning with $?range$ in the SPARQL query.

Formally, n TGs can be identified as follows:

$$TG_n = \{(v, *, *) \in J \ \cup \ (*, *, v) \in J : |\Phi_v| > 1, v \in V\},$$

where TG_n is a subgraph of the complete SPARQL query, J is the set of all Base Graph Patterns within the WHERE clause of the SPARQL query, V is the set of all variables found within the WHERE clause, $(v, *, *)$ denotes that v is the subject of a BGP, likewise $(*, *, v)$ implies v is the object, and Φ is defined as:

$$\Phi_v = \{v \in V : (v, *, *) \in J \ , \ (*, *, v) \in J\},$$

i.e. the set of all BGP's which contain v. Thus partitioning the RDF dataset into smaller subsets based on the triple groups and satisfying our second goal.

Once an identification of the TG's has been performed it is now necessary to determine which should become in-memory broadcast joins and which should be Reduce-Side joins. This can be selected based on the number of unique triples containing each $v \in V$ – small numbers indicating an in-memory broadcast join. The determination of the number of unique triples can be performed using a simple Map / Reduce algorithm. For example, we ran this against two real-world datasets which were combined with the instrument knowledge set and physiological knowledge set. Table II depicts the results for the three main TG groups – those of $?range$, $?obs$ and $?cs$ which reflect the triple groups used in all the SPARQL queries [15]. The $?cs$ group has not previously been mentioned but it contains information pertaining to physiological conditions. Like the $?range$ group, it is constant across datasets.

Table II
TRIPLE GROUP DISTRIBUTION

Triple Group Name	Number Of Elements
$?obs$	3,426,188
$?range$	518
$?cs$	446

The Table shows that the distribution of the triple groups is massively skewed towards the $?obs$ group, whilst both the $?range$ and $?cs$ groups contain very few instances. This is key from an optimisation point of view, as it means the $?range$ is reused and joined frequently against many unique $?obs$ elements. We exploit this skewed trait in the data by pushing smaller triple groups, such as $?range$, into main memory to massively reduce the number of joins required.

V. HADOOP IMPLEMENTATION

In this section, details of the approach we developed to query medical datasets via Hadoop are explored. There were many reasons why Map / Reduce was chosen as the basis for the new approach. Two of the most compelling reasons being that it is inherently parallel, whilst also reducing the complexity for developing a distributed application.

Previous state-of-the-art systems which query RDF via Hadoop are often limited to just performing SPARQL operations [19], [20]. However, there are a wide range of other useful data metrics and operations which can not be represented or extracted using SPARQL logic. These include metrics such as the number of times a certain query was completed, or operations such as creating a new subset of data conforming to a certain query. Also the number of joins required by complex queries makes the approach of systems like SHARD [19] unsuitable – as they rely on a series of cascade reduce-side joins where each join requires a complete Map / Reduce iteration.

To perform these complex queries on massive volumes of medical data, a new approach, based on our theoretical optimisations, has been created. This was intentionally designed with the constraint to work on a vanilla Hadoop install, requiring no additional software – maximising compatibility and portability. The algorithm is also compatible with Map / Reduce v1 and v2.

The approach required the RDF data to be stored on the HDFS in N-Triple serialisation. Previous work has shown

the N-Triple serialisation of RDF to be the most suitable for processing via Hadoop, as it represents a complete triple via a single line of text [11].

A. Algorithm Design Overview

This approach requires no pre-processing of the RDF data and takes the raw unaltered medical RDF dataset as input. Our previous work showed how costly the data upload stage can be [5] – requiring a data pre-processing stage, comprising two additional Map / Reduce iterations, before the query stage. As the data pre-processing stage requires traversing the entire dataset, leading to large portions of the dataset being exchanged over the network, this lead to an extremely slow approach.

Removal of the data pre-processing stage along with the realisation that repeated (almost) identical SPARQL queries consumed a significant proportion of the execution time formed the motivation for this work. Our new algorithm is able to perform all of the joins required in just two phases, these being the data selection and the join phase.

Where each phase can be realised as a single Map / Reduce iteration as outlined below:

- **Selection Phase** is used as a data reduction phase along with the identification of the triples which will become part of the in-memory Broadcast join. These triples are exchanged between nodes as part of the join phase.
- **Join Phase** performs the final joins of the in-memory data with the locally held data, along with the generation of the final results from the query.

This reduction to just two Map / Reduce iterations is achieved through a combination of our super-query and query segregation approaches.

The use of the flexible Map / Reduce platform also means that we have greater control over the low quality data once it has been identified. For example any data which has failed one of the data quality assessment metrics is split into a file on the HDFS based upon the failed metric. This allows the low quality data to be separated from the high quality data. Additionally, we can insert new RDF triples back into the original dataset which will identify any bad data so that it can be excluded from future use. Alternatively, we can delete the data entirely from the original, leaving a clean dataset for future analysis.

B. Selection and Triple Group Creation Phase

This phase performs two main functions and is implemented as a single Map / Reduce iteration. Firstly it traverses all of the triples stored on the HDFS and selects only those required – determined by the super-query. Secondly, the selection stage creates the triple groups and stores them as separate files on the HDFS ready for use in the join stage of the query algorithm. The key advantage of creating the approach using Hadoop, is that this map stage utilises data locality to run in parallel across a cluster.

The selection of the required triples is performed in the map task. Here triples are selected if they conform to any of the patterns in the super-query. To reduce the amount of data being transferred over the network via the shuffle and sort phase, only the required part of the triple is passed to the reduce phase. The output key of the map function is set as the join key and the output value is set as the required triple element plus a value denoting the join key to be used.

As an example, the selection phase for the small SPARQL query shown in Listing 2 would search the input data for any triples which matched any of the three required patterns. For the second pattern, the algorithm would search the input dataset and check each triple to see if the predicate is equal to $med : clinicalRangeMax$.

Listing 2. A small SPARQL query

```
?range a med:AcceptableRange .
?range med:clinicalRangeMax ?max .
?range pd:hasParameter ?p
```

Once a suitable triple has been located, the subject element matching $?range$ would be set as the Hadoop output key and the object element matching $?max$ would be set as the Hadoop output value. As a reduce function is spawned for each unique Hadoop key value this ensures that all the values for a particular instance of ?range are present in the same reduce function and thus can be joined together via the reduce side join method. Once the reduce of the selection phase has finished, all of the individual triple groups have been created and are stored on the HDFS ready for the join phase.

C. Join Phase

The join phase has two main functions and is implemented as a complete Map / Reduce iteration. Firstly it performs the required joins between the various triple groups in order to compete the queries. Secondly, it formats and returns the final output. The input to this stage is the data emitted by the selection stage. Using a broadcast-join enables numerous elements to be joined together within the same task.

To perform the broadcast-join, the smaller triple group files created in the selection phase are distributed to all the reducers running in the join phase. This uniform distribution of files is achieved by making use of the distributed cache feature, allowing files under 10 Gigabytes to be made accessible by any Map or Reduce task running upon any node within the cluster. The files pushed to the reduce phase via the broadcast join, are extracted into Hashtables, which allows for extremely fast lookups when checking for element membership. The broadcast join method is particularly applicable in this case, since the data is massively skewed towards one triple group. This group, which always forms the $?obs$ part in any query, is far too large to be stored in memory so must be joined via a reduce-side join. However, the smaller triple groups can be joined to the larger $?obs$ group the

broadcast join technique within the same reduce function. The use of the broadcast join to access the previously created triple groups and then load them into memory is the third main contribution of this paper.

These concepts are implemented in the following manner. Firstly, the map phase, which takes as input a file containing the $?obs$ results. The map phase decides if the current input is an $?obs$ record or a $?a2$ record based on length. These two triple groups are then joined via the common element and passed to the reduce phase. In the reduce phase, the two groups will be available in the same job due to the shuffle and sort phase. The system then performs the rest of the joins via the broadcast method. As explained above the files which are to be joined via the broadcast join method are all extracted from the Hadoop distributed cache and loaded into Hash-tables in main memory. Then the different triple groups can be joined via any common elements. Each of the joins performed via the broadcast method would otherwise have been completed via a separate reduce-side join, as they are all performed upon individual elements. Once the required joins are complete, the algorithm then performs the conditional logic determined by the original queries. The logic conditions include checking values against a predetermined range. Once the algorithm finds a value that does not meet the requirements, it will emit the required value and the other associated values back onto the HDFS. This stage again makes use of multiple outputs, to split the output from each conditional logic check into its own file on the HDFS. This allows a user to more easily see and access the values which have failed a particular logic check.

VI. Experimental Evaluation and Results

In this section we present the results from running our new approach on a small Hadoop cluster. The results show the work to be both faster and more scalable than the Jena and previous Hadoop implementation. This validates the effectiveness of the work detailed in this paper and is the last major contribution of this paper.

A. Testing Environment and Methodology

The different approaches were tested on a small development Hadoop cluster, comprising a head node with eight data nodes. All machines ran an identical software stack of CentOS 6.5 64-Bit, Java OpenJDK 1.7.0_51 and Hadoop 1.2.1. All nodes had identical hardware – an Intel Q8400 quad-core processor, 8GB of Memory, a 250GB (7200 RPM) HDD and communicated via a dedicated Gigabit switch.

Due to the sensitivity of the original datasets, only a small medical dataset was available so additional records were synthetically generated [5]. To produce the new data, a Map / Reduce algorithm was developed which generated new RDF data based upon an input of real world medical RDF data. The algorithm retains the structure and distribution from the real world dataset, but inserts new randomly generated values for the variable triple elements. The algorithm exploits knowledge of the datasets so that it does not alter triples which are consistent across all datasets, for example data about the technical capabilities of clinical sensor equipment.

B. Phase Performance Across Eight Nodes

Figure 2 shows the query performance of our new approach across the 8 node cluster. This result represents the total run time as there is no data formatting stage required. The Figure illustrates the proportion of the total run time required for each phase, with the total runtime being a sum of the two. It can be seen that the selection phase represents the vast majority of the total run time of the approach. This phase increases nearly exponentially, whilst the join phase grows much more slowly. The lack of an index for data stored on the HDFS requires the selection phase to traverse the entire dataset and then shuffle the required elements over the network to a reduce task. This explains why the time required to complete the phase increases much faster than for the join phase and is the main limitation of using Hadoop.

C. Performance Scalability Across Cluster Size

Figure 3 shows how our new approach scales across a range of cluster sizes. Only the eight node cluster was able to complete the queries on 1000M tripes, with all other cluster sizes having insufficient storage space available on the HDFS. The runtime for four and eight nodes is quite similar, with an increase to eight nodes only resulting in a modest decrease in runtime. This result can be explained by the extra nodes over-saturating the network connection, as more nodes result in an increase in the amount of data being shuffled over the network.

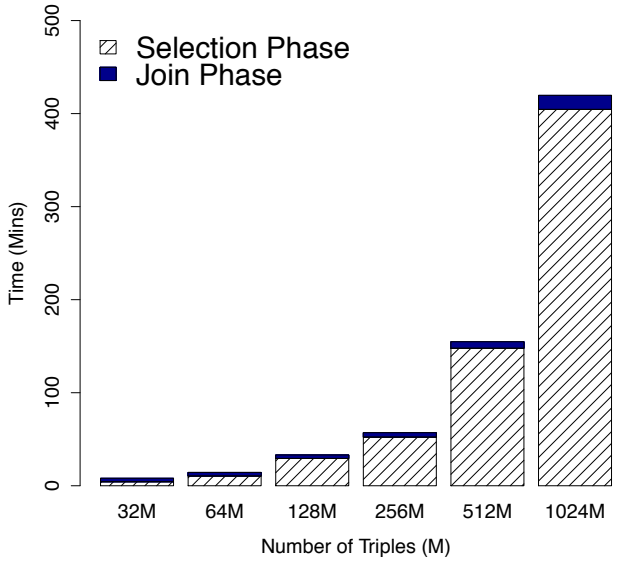

Figure 2. Query Performance Across Eight Nodes

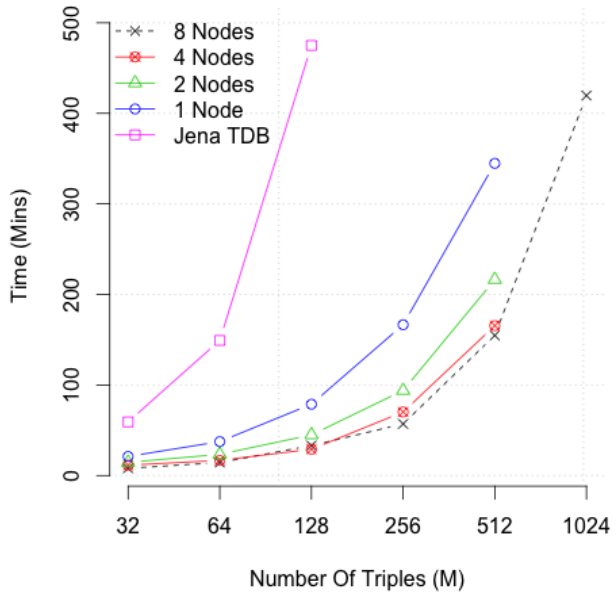

Figure 3. Performance Scalability Across Cluster Size

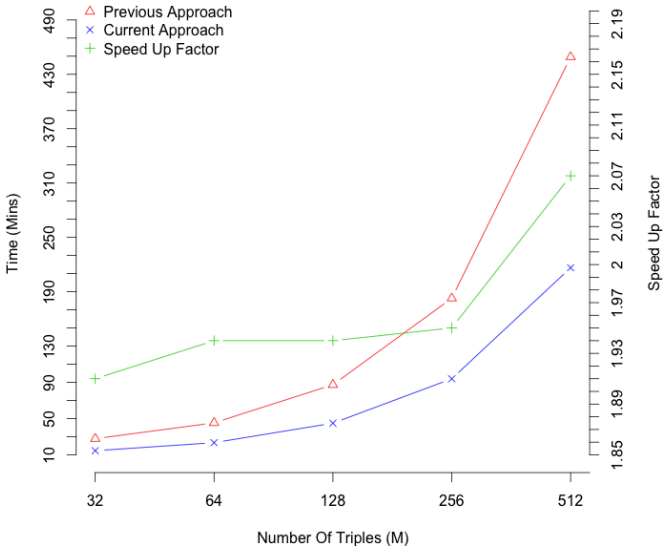

Figure 4. Comparison Between Approaches

For comparison, the results from running Jena on a single node from the cluster is also presented. Jena was unable to process dataset sizes over 128M triples as an upload request caused the system to hang. It can be seen that the Hadoop approach, run on any cluster size, is considerably faster than Jena. With the Hadoop approach able to query over 1000M triples in less time taken by Jena to query just 128M. It is worth noting that all the results were obtained from previous generation hardware and thus do not accurately represent performance we would expect when running on a modern dedicated Hadoop cluster with fast network – as Hadoop is know to be network limited.

D. Approach Comparison

Figure 4 shows the total run time on the two node cluster for both the new and previous approach [5]. The Figure also plots the speed-up factor between the previous and current approach. Our new approach (Current Approach) does not require the upload stage required in our previous approach. It can be seen that this is consistently faster across all dataset sizes than the previous approach. With the current approach demonstrating a speedup of 2 at a dataset size of 512M triples. Promisingly, the speedup factor appears to be increasing marginally, proportional to dataset size, demonstrating the scalability of the approach presented in this paper. The decrease in run time between the two approaches can be attributed to the optimisation techniques utilised to remove the requirement of the data being pre-processed and the new query approach.

VII. Conclusion and Further Work

A. Conclusion

With medical scientists increasing their research into predictive and prescriptive modelling it is of upmost importance that the available datasets are accurate and error free. However, it is also important that these datasets can be made available in a timely manner – especially significant for diagnosis. This work has presented a novel methodology of processing medical monitoring datasets, in linked data format, to assess for accuracy and validity. Building upon previous work, this approach is not only faster than previous implementations, in Jena and Hadoop, but also demonstrates clear potential to scale better than other approaches as the dataset size grows. This methodology involves extracting smaller groups of triples from a larger dataset and using a broadcast join technique to hold frequently called groups in memory. These are exploited as part of a super-query – merging multiple similar queries. This removes the costly data upload stage and greatly improves the algorithm's performance. Crucially, this new approach demonstrates a five times speedup over the Jenna approach and is twice as fast as our previous work.

These approaches have, to the best of our knowledge, not been attempted in current literature and can be adapted to other types of linked datasets. The clearly superior performance will allow for more and larger medical datasets to be checked for errors and inconsistencies leading to more accurate and reliable datasets for medical and diagnostic research. We hope that this work showing how utilising data analytic and graph theory techniques can inform a highly optimal query solution will inspire further research.

B. Further Work

The new Hadoop approach has only been tested on a small cluster using a comparatively small medical dataset size. Further investigation is required to see if this approach would continue to scale as cluster size is increased. Additional

work would also be required to assess if the approach would remain effective on dataset sizes past one billion triples.

Further research work could be performed as to the optimal size of triple groups added to the distributed cache. In this work the frequently joined to triple groups are under one gigabyte in size so it would be interesting to study how performance is affected as more triples are added to the distributed cache. In addition other metrics could have been investigated to gain a deeper insight into the performance. These include metrics related to the Hadoop join key and more detailed comparisons of the two approaches.

Currently this work has been tailored specifically to process medical RDF datasets and pre-determined SPARQL queries. However future research is needed to investigate the possibility of creating a generic framework for processing RDF datasets via Hadoop. The key aspect of this generic framework would be that the optimisations explored here would be automatically performed. Specifically this means, researching a way of automatically creating the triple groups, assessing the frequency of any intergroup joins and then pushing the small but frequently joined groups into the distributed cache for the broadcast join.

ACKNOWLEDGMENT

The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work. We would also like to thank EPSRC for continued funding. In addition we would like to acknowledge the clinical input into our earlier work from Prof. John Kinsella (Glasgow Royal Infirmary) and Dr. Ian Piper (Institute of Neurological Sciences, New South Glasgow Hospital).

REFERENCES

[1] Intel White Paper. Bigger Data for Better Healthcare 2013.

[2] Olson, D. McNett, M. Lewis, L. Riemen, K and Bautista, C. Effects of nursing interventions on intracranial pressure Am J Critical Care. 22(5):4318, 2013 Sep.

[3] Nizami, S. Green, J.R. and McGregor, C. Implementation of Artifact Detection in Critical Care: A Methodological Review In *IEEE Reviews in Biomedical Engineering*. 127-142, 2013.

[4] Moss, L. Corsar, D. Piper, I. and Kinsella, John Trusting Intensive Care Unit (ICU) Medical Data: A Semantic Web Approach In . *Proceedings of 14th International Conference on Artificial Intelligence in Medicine*, (AIME 2013), pages 68-72, Springer, 2013.

[5] Bonner, S. Antoniou, G. Moss, L. Kureshi, I. Corsair, D and Tachmazidis, I. Using Hadoop To Implement a Semantic Method Of Assessing The Quality Of Research Medical Datasets In . *Proceedings of the 2014 International Conference on Big Data Science and Computing*, BigDataScience 2014, ACM, 2014.

[6] Afrati, F and Ullman, J. Optimizing joins in a map-reduce environment. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 99–110, New York USA, ACM, 2010.

[7] Kaisler, S. Armour, F. Espinosa, J. A. and Money, W. Big data: Issues and challenges moving forward In *Hawaii International Conference on System Sciences*, HICSS 2013, pages 995-1004, IEEE, 2013.

[8] DuCharme, B. *Learning SPARQL* , O'Reilly Media Inc, 2013.

[9] Lin, J and Dyer, C. *Data-Intensive Text Processing with MapReduce*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool Publishers, 2010.

[10] Miner, D and Shook, A. *Mapreduce Design Patterns Building Effective Algorithms and Analytics for Hadoop and Other Systems*. O'Reilly, 2012.

[11] Myung, J. Yeon, J and Lee, S. Sparql basic graph pattern processing with iterative mapreduce. In *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud*, MDAC '10, New York, USA, ACM, 2010.

[12] Goldberg, S. Niemierko, A and Turchin, A. Analysis of data errors in clinical research databases. In *AMIA Annual Symposium Proceedings*, volume 2008, page 242. American Medical Informatics Association, 2008.

[13] François Goasdoué, Zoi Kaoudi, Ioana Manolescu, Jorge Quiané-Ruiz, and Stamatis Zampetakis. CliqueSquare: efficient Hadoop-based RDF query processing. In *BDA'13 - Journées de Bases de Données Avancées*, Nantes, France, 2013.

[14] W3C. Rdf primer, http://www.w3.org/TR/2004/REC-rdf-primer-20040210/, 2004.

[15] Orignal SPARQL queries. http://homepages.abdn.ac.uk/dcorsar/pages/medical/index.php

[16] Linked Data http://www.w3.org/DesignIssues/LinkedData.html

[17] W3C Semantic Sensor Network Ontology. http://www.w3.org/2005/Incubator/ssn/ssnx/ssn

[18] Resource Description Framework http://www.w3.org/RDF/

[19] Rohloff, K and Schantz, R. High-performance, Massively Scalable Distributed Systems Using the MapReduce Software Framework: The SHARD Triple-store. In *Programming Support Innovations for Emerging Distributed Applications*, New York, USA, ACM, 2010.

[20] Du, J. Wang, H Ni, Y and Yu, Y. HadoopRDF: A Scalable Semantic Data Analytical Engine In *Proceedings of the 8th International Conference on Intelligent Computing Theories and Applications* ICIC'12, Springer, 2012.

[21] Zeng, K. Yang, J. Wang, H. Shao, B and Wang, Z. A Distributed Graph Engine for Web Scale RDF Data In *Proceedings of the 39th international conference on Very Large Data Bases* PVLDB'13, ACM, 2013.

[22] Ladwig, G and Harth, A. CumulusRDF: linked data management on nested key-value stores In *The 7th International Workshop on Scalable Semantic Web Knowledge Base Systems* SSWS'11, 2011.

[23] McBride, B. Jena: a semantic Web toolkit In *IEEE Internet Computing* 2002.

[24] Neumann, T and Weikum, G. RDF-3X: a RISC-style engine for RDF In *Proceedings of the VLDB Endowment* PVLDB'08, ACM, 2008.

[25] Kaoudi, Z and Weikum, G. RDF in the clouds: a survey In *The VLDB Journal* , Springer, 2014.

[26] White, T. Hadoop: The definitive guide O'Reilly Media Inc, 2012.