Improving the Diagnosis of Mild Hypertrophic Cardiomyopathy with MapReduce

Pantazis Deligiannis
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh, U.K.
pd85@hw.ac.uk

Hans-Wolfgang Loidl
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh, U.K.
H.W.Loidl@hw.ac.uk

Evangelia Kouidi
Laboratory of Sports Medicine
Aristotle University of Thessaloniki, Greece
ekouidi@med.auth.gr

ABSTRACT

Hypertrophic Cardiomyopathy (HCM), an inherited heart disease, is the most common cause of sudden cardiac death in young athletes. Successful diagnosis of mild HCM presents a major medical challenge, especially in athletes with exercise-induced hypertrophy that overlaps with HCM. This is due to a wide spectrum of non-specific clinical parameters and their complex dependencies. Recently, medical researchers proposed multidisciplinary strategies, defining differential diagnostic scoring algorithms, with the goal of identifying which parameters correlate with HCM in order to achieve faster and more accurate diagnosis. These algorithms require extensive testing against large medical datasets in order to identify potential correlations, and assess the overall algorithmic quality and diagnostic accuracy.

We present a prototype data-parallel algorithm for improving the diagnosis of mild HCM, by refining the set of parameters contributing to the main diagnostic function. To this end, we employ a rule-based, machine-learning approach and develop an iterative MapReduce application for applying the diagnostic function on large data-sets. The core component of the algorithm, including the diagnostic function, has been implemented in Java, Pig and Hive in order to identify potential productivity gains by using a high-level MapReduce language specifically for medical applications. Finally, we assess the algorithmic performance on up to 64 cores of our Hadoop (version 0.20.1) enabled Beowulf cluster, managing to achieve near-linear speedups while reducing the overall runtime from over 9 hours to a couple of minutes for a realistic dataset of 10,000 medical records.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming; I.2.6 [**Artificial Intelligence**]: Learning; J.3 [**Life and Medical Sciences**]: Health

General Terms

Algorithms, Experimentation, Performance

Keywords

MapReduce, Hadoop, Machine Learning, Medical Diagnosis, Hypertrophic Cardiomyopathy

1. INTRODUCTION

Hypertrophic Cardiomyopathy (HCM), an inherited heart disease, is commonly acknowledged by the medical community as the primary cause of sudden cardiac death (SCD) during physical activity [9]. SCD of a young athlete is unarguably the most tragic event in sports, with an annual incidence rate of 2.3 per 100,000 athletes [2]. Successful medical diagnosis is currently achieved by health screening, using highly expensive and sometimes invasive procedures such as gene testing and cardiac biopsy. This approach is very time consuming as it is based on monitoring a wide spectrum of clinical parameters with many complex dependencies between them. Especially in the case of athletes with "grey zone" cardiac hypertrophy (overlap of the benign adaptation to exercise training with HCM — as defined in Section 2.1), the diagnosis is even more challenging as the cardiologists have to differentiate between the malignant cases of particular clinical heterogeneous mild HCM and the exercise-induced "athlete's heart" [2, 15].

Today, medical experts focus their research efforts in developing state-of-the-art multidisciplinary strategies with the aim to identify which medical parameters are related with HCM in order to potentially achieve much faster and more accurate diagnosis. A promising approach is the use of novel differential diagnostic scoring algorithms in order to efficiently detect HCM, particularly in "grey zone" cardiac hypertrophy cases [12]. A major challenge behind the successful development of such techniques is that HCM is not only a highly complex heart disease, but also has a very low prevalence in the available medical datasets. These two facts converge to a need in using large-scale medical datasets in order to test the differential diagnostic algorithms, to identify the correlation between the parameters and the disease, and assess the overall algorithmic quality and accuracy.

The primary focus of this paper is the application of a prototype, massively data-parallel, MapReduce [3] algorithm to improve the diagnosis of mild HCM. This is a typical problem in the medical domain as it involves data-intensive computations, high-dimensional data and unclear parameter dependencies [13]. The computational core is the calculation of the correlations between the algorithmic score and known cases of HCM. In an iterative, rule-based approach the algorithm is improved building on a database of expert knowledge. Our algorithm is implemented in Hadoop [20], an open source MapReduce implementation provided by Apache. The massive computational size that derives from applying the algorithm on large-scale datasets justifies the choice for a massively data-parallel implementation.

Our iterative rule-based approach is based on machine learning [10, 1, 13]: *learning MapReduce jobs* are chained in order to identify correlations between the given medical parameters and the HCM disease. Each MapReduce job in the loop investigates only a carefully selected subset of the total medical parameters. The selection of the parameter set is automatic and is based on the results from previous MapReduce iterations. This approach lessens the overall workload as subsequent parameters can be discarded if they are found unrelated. The development of the proposed algorithm was based on medical data and domain expertise provided by the Laboratory of Sports Medicine, Aristotle University of Thessaloniki, Greece.

We developed the core component of the iterative learning algorithm in Java, Pig [11] and Hive [18] with the goal to evaluate the ease-of-programming associated by using a high-level language and to assess their suitability for medical applications. Furthermore, we present comparisons between two different domain knowledge based approaches: a naïve brute force approach and an approach based on learning MapReduce iterations. Finally, we evaluate the algorithmic performance on 64 cores of our Hadoop Beowulf cluster, managing to achieve near-linear speedups while reducing the overall runtime from over 9 hours to a couple of minutes for a large-scale medical dataset of 10,000 clinical records. We freely provide a well-documented for domain experts source code in GitHub[1] with the hope that it will enable other medical researchers to get accustomed with MapReduce and use it in their future research.

The remainder of the paper is organised as follows: Section 2 contains background about the challenges in diagnosing HCM, a brief overview of the MapReduce execution framework and a presentation of the Hadoop ecosystem; Section 3 presents our diagnostic scoring function and its parallelisation in MapReduce; Section 4 provides an in-depth discussion about the iterative learning MapReduce algorithm we developed in order to increase the accuracy and efficiency of the HCM diagnosis; Section 5 presents the extensive experiments we conducted and the evaluation of our algorithm on our Hadoop enabled Beowulf cluster; and Section 6 concludes with a discussion about the contributions of this paper, limitations and future work.

2. BACKGROUND

2.1 Challenges in diagnosing HCM

Diagnosing HCM is a challenging process, especially in the case of athletes with "grey zone" cardiac hypertrophy: an above average, 12–16 mm cardiac septum thickness that overlaps with HCM. The main reason behind this is that no established clinical, non-invasive, diagnostic criteria exist for distinguishing hypertrophies, such as the physiological

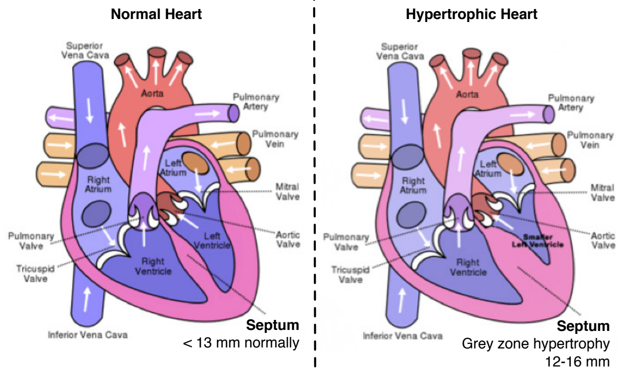


Figure 1: Difference in septum thickness between a normal and a "grey zone" hypertrophic heart (based on original illustration by Eric Pierce that is freely available under a Creative Commons licence)

adaptation of the heart to exercise training and the pathological cardiac hypertrophy (see Figure 1). Gene testing and detraining are possible but not feasible options as they are either highly expensive and not always informative or require an athlete to stop training [2, 9].

Successful diagnosis in elite athletes can be even more difficult, as the prevalence of HCM is extremely low [15]. It is suggested that 10-15% of HCM patients have a "grey zone" cardiac hypertrophy, but the exact prevalence of HCM in "grey zone" cardiac hypertrophy is currently unknown. During the last two decades cardiovascular pre-participation screening was performed in more than 22,000 young athletes in the Sports Medicine Laboratory, Aristotle University of Thessaloniki, Greece. HCM was diagnosed in only twenty of those athletes [5].

To overcome these diagnostic challenges, researchers shift their focus towards the development of novel multidisciplinary methods for diagnosing HCM by using mass pre-participation screening. As an example, Pagourelias et al. suggested a novel differential diagnostic scoring algorithm comprised of indexes from multiple clinical examinations to diagnose HCM in "grey zone" cardiac hypertrophy cases [12]. Such an algorithm, that takes into consideration the various parameters that directly or indirectly play a role in the presence of HCM, could prove to be a feasible, efficient and inexpensive tool for diagnosing HCM and identifying patients with high SCD risk. Our diagnostic algorithm directly builds on this expert knowledge.

2.2 MapReduce

The MapReduce framework, introduced by Google during 2004, is a distributed programming model for analysing internet-scale data sets in acceptable time bounds [3]. It enables powerful large-scale data analysis, while negating the data processing bottlenecks of the past. Today, MapReduce is a key Cloud computing technology empowering hundreds of industrial and academic projects. As an example of its success, Google runs more than 100,000 MapReduce tasks in a daily basis [4]. Although MapReduce is commonly used for search engine indexing, data warehousing and log processing, lately it has gained acceptance by the wider scientific community and is being increasingly used in a wide range

[1]https://github.com/pdeligia/mapred-hcm

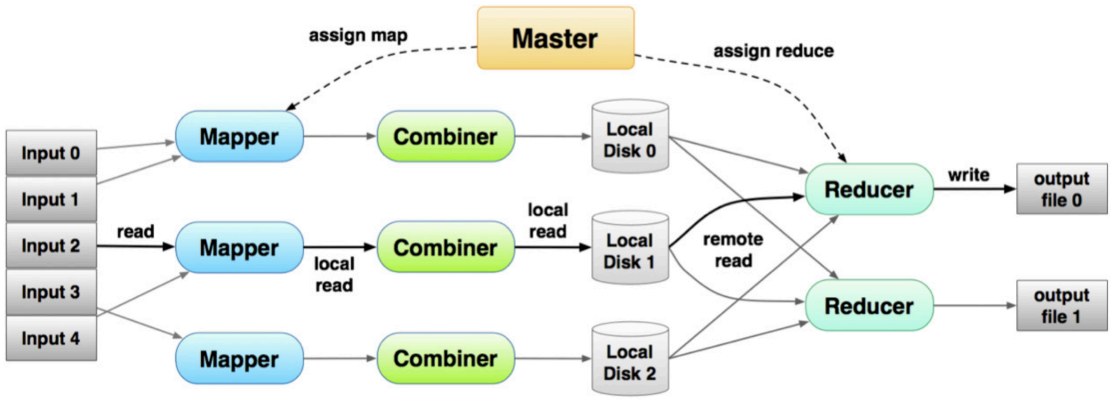


Figure 2: The MapReduce execution workflow. Map tasks are assigned for each input record. The map phase results can be combined locally (reduce-like operation) to enhance the overall efficiency of MapReduce. The intermediate results are then gathered by the reducers for the final aggregations. (based on work created and shared by Google and used according to terms described in the Creative Commons 2.5 Attribution License)

of data-intensive applications such as genome analysis [14] and medical research [19].

The MapReduce model is based on Map and Reduce, two parameterised functions largely inspired by the map and reduce higher order functions in functional programming. The power of MapReduce derives from the generality of its abstraction combined with a carefully tuned parallel implementation. Programmers no longer have to worry about all the underlying complexities associated with large-scale parallelism (e.g., coordination and synchronisation of tasks). They only need to define implementations of Map and Reduce and the run-time system will automatically parallelise the MapReduce job by distributing the individual computational tasks among the available processing elements [3].

Figure 2 presents the overall MapReduce execution workflow. Initially, the input files are split into multiple chunks. Afterwards, the master processor initialises the map phase and assigns a map task for each input record. Map tasks are executed in parallel on the available map workers. The map task results can be optionally combined while they are in memory (not yet stored in a local disk), thus raising the overall efficiency as less disk read and writes will be required. Finally, these intermediate results are gathered by the corresponding reducer workers and are aggregated to produce the final results.

2.3 The Hadoop ecosystem

Hadoop[2] [20] is an open source implementation of Google's MapReduce pattern. It is maintained by the Apache Software Foundation and among its main code contributors are major companies such as Yahoo!, Facebook and Cloudera. The Hadoop ecosystem consists of a large number of subprojects (e.g., Pig and Hive) and a vibrant open source community that supports them.

Hadoop is accompanied by the Hadoop Distributed File System (HDFS), a high-throughput storage system that is responsible for distributing large-scale data sets across the available computing nodes of a cluster. HDFS is an important aspect of the Hadoop ecosystem as it also contributes towards fault tolerance by replicating the available data sets (either automatically or on-demand) and by providing a heartbeat mechanism for checking if workers have failed. Thus, if a cluster node fails during a MapReduce job (a very common event in large clusters setups) the corresponding data will not be lost but will be accessed from another node [16].

Notably, the Hadoop ecosystem also includes Pig Latin [11] and Hive QL [18], two high-level data-query languages that aim to enhance productivity by providing powerful, general purpose, programming language abstractions. Pig Latin, introduced by Yahoo! in 2006, resides inside the Pig platform and combines high level abstractions with a lower level procedural style, making it an ideal language for programmers familiar with Java. Pig not only provides a wide range of data manipulation functions (e.g. SORT, FILTER, GROUP and JOIN), but also allows the programmers to embed their own special purpose user-defined functions. Nowadays Pig has been widely adopted by an increasing number of data analysis companies, such as Twitter, and is widely used in their every day data intensive tasks [8].

Hive QL, initially developed in Facebook, is an SQL-like declarative language defined by Hive, a data warehouse platform for Hadoop. Hive users can perform ad-hoc querying that allows them to manipulate and analyse large-scale data sets stored in an HDFS cluster. Similar to Pig, Hive also provides great extensibility, as it allows the programmers to use their own user defined functions. Hive style of programming is closer to that of traditional database management systems as it allows the definition of tables, columns and records. Furthermore, Hive allows the programmer to abstract over the defined schema through views, a method that can potentially enhance query processing.

Recently, YARN[3] (Hadoop version 0.23) was released, introducing a complete overhaul of the Hadoop MapReduce framework. With this major update, Hadoop is now capable of not only running classic MapReduce jobs but also more complex patterns of computation, specified as directed acyclic graphs of jobs (similar to Microsoft's Dryad [7]).

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

[3]http://hadoop.apache.org/common/docs/r0.23.0/index.html

YARN aims not only to achieve scalability towards hundreds of thousands of cores that will probably become the main high performance computing platform in the near future, but also to enhance Hadoop as a general purpose data parallel computing framework. YARN, though, is currently in alpha testing, and thus not recommended for production environments yet.

3. HCM DIAGNOSIS WITH MAPREDUCE

The technique we use to diagnose mild HCM is based on previous work by Pagourelias et al. [12], who defined a diagnostic score for differentiating "grey zone" cardiac hypertrophy cases into physiologic and pathologic. We extend this concept by prototyping a diagnostic scoring function in Hadoop MapReduce for diagnosing mild HCM in large-scale datasets of athletes with "grey zone" cardiac hypertrophy. Towards this purpose, we cooperated with experts from the Laboratory of Sports Medicine, Aristotle University of Thessaloniki, Greece, who provided the required medical data and domain expertise. The data consisted of clinical results from eleven non-invasive diagnostic tests (involving 35 medical parameters in total) that were performed on 40 young athletes with "grey zone" cardiac hypertrophy. Twenty of them were diagnosed with HCM, confirmed by cardiac Magnetic Resonance Imaging. Six of these 20 athletes were also found with high-risk factors for SCD. The HCM diagnosis for the rest 20 athletes was negative. This is only the core data initially used for designing the scoring function.

The diagnostic scoring function matches each of the 35 medical parameters against their established diagnostic value ranges (cut-off limits), found with ROC curve analysis [21], and their weights (measure of importance for the diagnostic process). Table 1 presents a small sample of these parameters. Five out of the 35 parameters are medically accepted as high-risk factors for SCD in subjects with HCM [6]. These are: (i) personal history of unexplained syncope during effort, (ii) family history for SCD, (iii) extreme left ventricular wall thickness in echocardiogram, (iv) presence of non-sustained ventricular tachycardia on 24-h Holter electrocardiographic recordings and (v) blood pressure decrease or inadequate increase during upright exercise.

The result from mapping the scoring function against the input medical dataset is two intermediate vectors of values, representing the contribution of one parameter to the overall score. The first vector encompasses the high-risk SCD parameters, the second vector encompasses all parameters. These values are calculated based on the corresponding parameters' weights and cut-off limits: if the value of a medical parameter is found above (or below accordingly) its cut-off limit, it will add its weight in the corresponding vector, else it will add a 0. Afterwards, the values in each vector are summed, resulting into two distinguishable intermediate scores (high-risk and general score). As an example, a medical record that includes findings of all five high-risk SCD factors will have a high-risk score of 5, whereas a medical record without high-risk SCD findings will have a high-risk score of 0. The high-risk and the normal scores are then summed into an overall score, effectively assessing the HCM and SCD risk in the corresponding medical record.

The positive diagnosis of HCM is confirmed if the overall score is equal or greater than 23. Furthermore, a diagnosis results in high-risk SCD findings when the high-risk score is equal or greater than 1. Therefore, the diagnostic scoring function uses the high-risk score and the overall score, in order to classify the medical records into: HCM and high-risk SCD (overall score ≥ 23 and high-risk score ≥ 1); high-risk SCD without HCM (overall score < 23 and high-risk score ≥ 1); mild HCM (overall score ≥ 23 and high-risk score $= 0$); and healthy with "grey zone" cardiac hypertrophy (overall score < 23 and high-risk score $= 0$).

Table 1: Sample of HCM related medical parameters with their cut-off limits and weights.

Medical Parameter	Cut-Off Limit	Weight
Number of symptoms	> 1	2
Positive family history (y/n)	> 0	2
Pathological ECG	> 1	2
IVS (mm)	> 13.52	1
pVO_2 (ml/kg/min)	< 49.67	2
BNP (pg/ml)	> 9.22	2

Implementation aspects:

Our implementation for classifying large-scale datasets into pathologic and physiologic cases exploits massive data parallelism provided by the MapReduce pattern. The mapper function uses the above diagnostic scoring function. Depending on the outcome of each map task, one of the aforementioned classifications is chosen as the map key and the value is assigned with the integer 1. Importantly, multiple map outputs with the same key can effectively be combined by the combiner, thus reducing the overall computational size that the performance-critical reducer has to perform. The reducer function computes the component-wise sum over a four-tuple of all possible classes. It is important to mention that the reduce phase is based on unequal load balancing (much more healthy records than unhealthy in a dataset based on the general population), but this does not hinder performance as the reduction is very cheap compared to the mapping.

4. IMPROVING THE HCM DIAGNOSIS

The accurate diagnosis of mild HCM in "grey zone" cardiac hypertrophy cases is of critical importance as it can lead to the prevention of sudden cardiac death incidents. Although the above diagnostic scoring function provides an accurate diagnosis when tested against the initial small-scale dataset (40 medical records), for much larger datasets ($> 10^3$ medical records) there is loss of 10-20% accuracy. Domain experts believe there are two main reasons behind this: first, many of the medical parameters required for successful diagnosis of HCM do not have completely understood dependencies between them and they need to be taken into account to gain higher diagnostic accuracy; and second, the artificial large-scale medical datasets we used for the experimentation (created in cooperation with medical experts) can showcase some extreme — but possible — combinations of medical values that can cause uncertainty in the classification.

Our approach towards providing a more feasible and accurate diagnosis of mild HCM is to use optimisation techniques based on machine learning [10, 1, 13], in particular a rule-based approach, in order to identify the relevant combina-

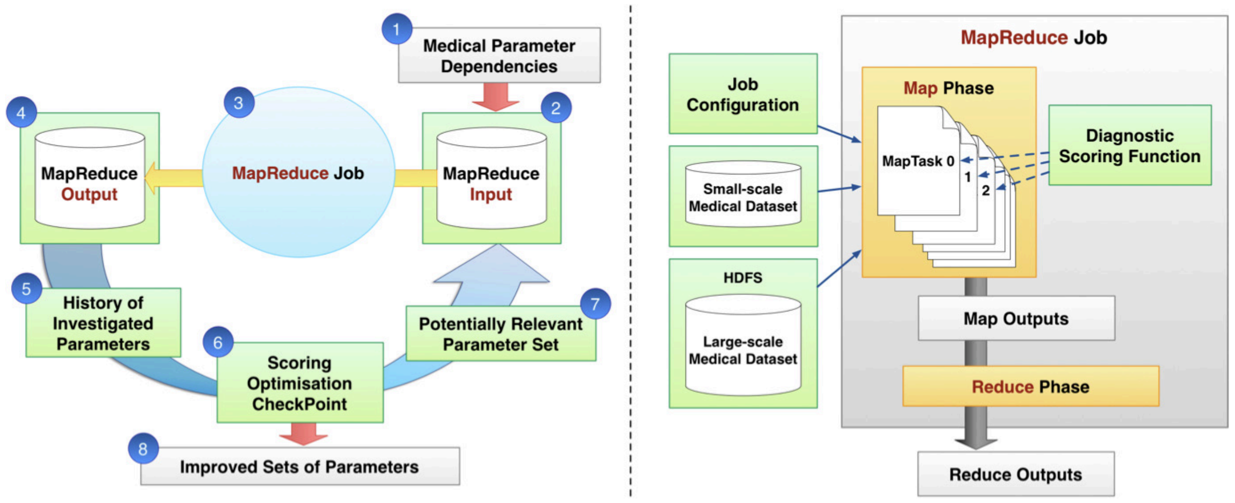

1
Medical Parameter Dependencies
2
MapReduce Input
3
MapReduce Job
4
MapReduce Output
5
History of Investigated Parameters
6
Scoring Optimisation CheckPoint
7
Potentially Relevant Parameter Set
8
Improved Sets of Parameters
MapReduce Job
Job Configuration
Small-scale Medical Dataset
HDFS
Large-scale Medical Dataset
Map Phase
MapTask 0
1
2
Diagnostic Scoring Function
Map Outputs
Reduce Phase
Reduce Outputs

Figure 3: Iterative MapReduce learning (left) — MapReduce job during each learning iteration (right).

tions of medical parameters that: (i) can successfully classify a large-scale medical dataset into pathological and physiological cases; (ii) satisfy any known or speculated dependencies between the parameters; and (iii) return acceptable medical results, i.e., medical records classified as unhealthy can not be more than 0.1–0.2% of the overall dataset. To achieve this efficiently, our algorithm goes through a number of learning MapReduce iterations: each iteration improves the parameter set based on the results from the previous MapReduce jobs and based on a set of rules encoding likely dependencies between some parameters.

Conceptually, the learning algorithm iterates over a set of potentially relevant parameters that should be considered by the diagnostic function, starting with those identified in the core diagnostic function in Section 3. Based on this set, a sequence of rules is applied, generating a modified parameter set. Each rule in the sequence is of the form $p \leftarrow p_1 \oplus \cdots \oplus p_n$, where $p, p_1, \ldots, p_n$ are medical parameters and $\oplus$ is a boolean combinator, one of: logical *and*, logical *or*. Thus, the right hand side represents a logical formula that is true, iff the combination of parameters is contained in the current parameter set. Iff the formula is true, the parameter on the left hand side is added to this set.

The implementation operates on two levels as shown in Figure 3. On the outer level (left hand side) the algorithm iterates over the parameter set and in the inner level (right hand side) the algorithm runs MapReduce jobs, applying the diagnostic function. Initially, the learning algorithm creates a MapReduce job configuration object that includes all the user defined dependencies between the medical parameters (Step 1 in Figure 3). After the job configuration has been successfully defined and the medical parameters to be investigated have been included in the MapReduce input (step 2 in Figure 3), the MapReduce computation can begin (Step 3 in Figure 3). While the iterations of our learning algorithm are executed in a sequential fashion, the actual MapReduce computation is massively parallel and well-tuned for execution in large-scale Hadoop clusters. The MapReduce job has three inputs: the job configuration; the small-scale medical dataset of 40 medical records used for validation; and the large-scale medical dataset stored in HDFS.

The iterative structure of the algorithm is problematic from a performance point of view, because it introduces synchronisations between the iterations, significantly deteriorating the available parallelism. Indeed we observed fairly poor performance in an initial, straightforward implementation. To overcome this problem, we had to increase the computational amount performed in one MapReduce iteration, without losing the learning aspect of the algorithm. We achieve this by investigating all possible permutations of parameters from the same potential parameter set in one iteration. This approach lessened the relative overhead associated with initialising a Hadoop job in each iteration, and led to significant performance gains as showcased in the evaluation section.

The map phase, thus, proceeds by generating all possible combinations of the parameters defined in the job configuration. Then, in a qualifying step, the diagnostic scoring function is applied on the small-scale medical dataset for each combination to filter out the combinations that do not diagnose HCM correctly. Finally, in a classification step, the diagnostic scoring function is applied to the large-scale dataset to classify the medical records. At the end of the map phase, the classified medical records (for each validated combination of medical parameters) are sent to the reducer as intermediate outputs.

The reduction phase begins with the reducer calculating the sum of the medical records that were classified as pathological for each potential combination of parameters. This sum is expected by the medical experts to be between 0.1–0.2% of the large-scale medical dataset. Thus, whichever combination of parameters does not achieve the expected ratio is filtered out. The output of the reduce phase is the list of all combinations of parameters that achieved the expected ratio (Step 4 in Figure 3).

In order to guarantee termination of the iterations and to avoid duplication of work, the implementation keeps a record of all parameter sets seen so far (Step 5 in Figure 3). Since this record increases in each iteration, and is finite, termination is guaranteed. Using the successful parameter-sets as input, the given rule-set is interpreted, delivering a new set of parameters that becomes input to the next

MapReduce iteration (Step 7 in Figure 3). Finally, when the iterative MapReduce learning loop terminates (Step 6 in Figure 3), the application ends by returning as results all possible combinations of parameters that achieve the three aforementioned criteria (Step 8 in Figure 3).

As a baseline for our performance comparisons, and in order to assess the effectiveness of our learning approach, we also provide the option for brute-force execution of our algorithm. If the user decides not to provide any dependencies, the learning algorithm will not perform any iterations, and will attempt to investigate all possible combinations of parameters in a single MapReduce job. This brute-force approach, though, suffers one major problem: the computational size for the map phase increases exponentially with the number of investigated parameters. In contrast, by providing parameter dependencies, the learning algorithm can split the workload into multiple iterations providing much faster results as presented in the evaluation section.

5. EVALUATION

We implemented the MapReduce iterative learning algorithm in Hadoop 0.20.1. The iterative learning components were developed in Java, while the diagnostic scoring function was also developed in Pig 0.8.1 and Hive 0.7.0, two very popular high-level data-query languages for developing Hadoop MapReduce applications. For the evaluation, we executed our code on the Beowulf cluster of Heriot-Watt University, Edinburgh, UK, running Cloudera's Hadoop distribution. Each node of the Hadoop cluster consisted of 4 cores at 2.0 GHz, 12 GB of RAM and 150 GB of available disk space. We conducted the experiments using up to 20 nodes (4 tasks per node) of our cluster during off-peak hours in order to minimise interference of external processes.

Language comparison:

For the comparison of the Java, Pig and Hive versions of the diagnostic scoring function, we used $2 * 10^8$ auto-generated realistic medical records (roughly 30 GB). In order to ensure the validity of our data, we compared the medical results with the original set of 40 medical records provided by the Laboratory of Sports Medicine, Aristotle University of Thessaloniki, Greece.

Figure 4 plots the speedup performance for each language implementation (Java, Pig and Hive) of the diagnostic scoring function. The figure shows that for all configurations, Java performed much better than Pig and Hive. For example, the Java version of the function using 80 cores achieved a speedup of 61.15, whereas both Pig and Hive were found slower than Java by a factor of approximately 1.27. This finding was expected because although Pig and Hive, as higher level languages, provide many programming abstractions leading to ease-of-programming, Java is usually better for raw performance and optimisations [17].

We can further notice in Figure 4 that although Hive initially yields higher speedups than Pig, the gap gradually closes between the two implementations as the number of cores increases. After 64 cores this gap is eliminated, and from that point and onwards both languages achieve the same speedup performance. While investigating for further differences between the performance of the three languages, one could notice that although Pig and Hive exhibit a steady speedup growth, this is not the case with Java, which exhibits irregular speedup growth trends, especially between 32 and 64 cores.

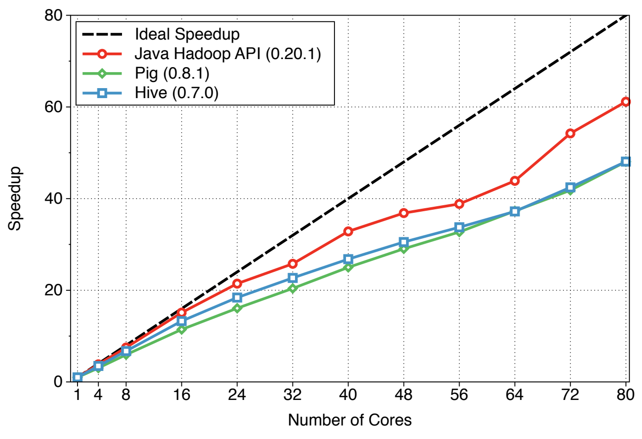


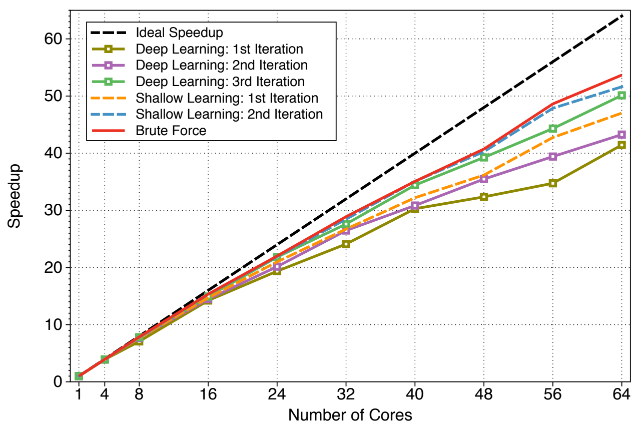
Figure 4: Speedups for each implementation (Java, Pig and Hive) of the diagnostic scoring function.

Concluding the language comparison between Java, Pig and Hive, we present the number of source code lines that were required for each implementation of the diagnostic scoring function. The Pig implementation required 72 lines of source code, which is only 55% of the corresponding Java version (132 lines of source code). Hive delivers the shortest version of the three with only 68 lines of source code. This result was expected because Pig and Hive bring many high-level abstractions to the table, which potentially leads to easier constructed parallel MapReduce applications.

Iterative learning algorithm:

For the evaluation of the iterative MapReduce learning algorithm, we compare three different learning scenarios, each realised with its own rule-set: (i) no predefined dependencies between the medical parameters leading to a single MapReduce job ("brute-force learning" approach); (ii) "shallow learning" of the medical parameters leading to two MapReduce learning iterations; and (iii) "deep learning" of the medical parameters leading to three MapReduce learning iterations. The motivation for using these artificial rule-sets derived from our extensive discussions with domain experts. Finally, we used an artificial dataset of realistic size: 10,000 medical records which is of similar scale with the 22,000 large-scale dataset monitored by the Sports Medicine Laboratory — see Section 2.1. In this setup, a single core MapReduce execution was slightly over 9 hours, thus, feasible to complete overnight while producing significant results.

Table 2 summarises the end-to-end runtime results for each of the three aforementioned learning scenarios from 1 to 64 cores of the Hadoop cluster. We can easily observe that as the number of learning iterations increases, the end-to-end execution time significantly drops. As an example, the brute-force learning scenario resulted in an overall runtime of 33,598 seconds for a single-core execution and 626 seconds for 64 cores. As expected, the learning algorithm performed much faster under three MapReduce learning iterations: 345 seconds end-to-end runtime on 64 cores and 16,233 seconds for single-core execution, justifying the design of our algorithm. Despite the more challenging nature of iterative MapReduce jobs these results make intuitive sense, because



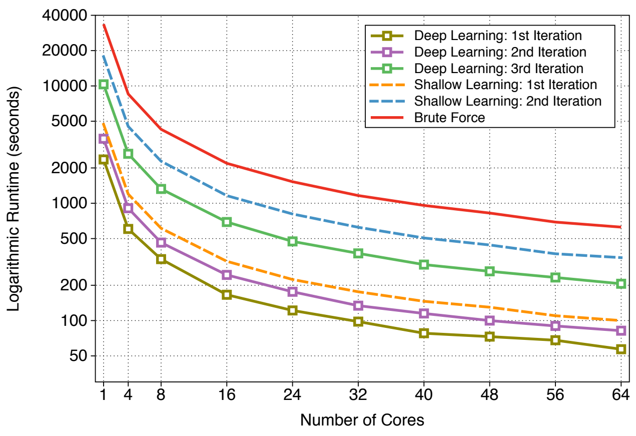


Figure 5: Speedup (left) and logarithmic runtime (right) comparisons for the three rule-sets.

Table 2: End-to-end runtime and speedups for all three rule-sets.

Learning Approach	Runtime (sec)		Speedup		
(Rule-Set)	1C	64C	16C	32C	64C
Brute-force	33598	626	15.34	28.89	53.67
Shallow Learning	22459	444	15.17	28.07	50.58
Deep Learning	16233	345	14.72	26.79	47.05

by introducing multiple learning iterations only a carefully chosen set of medical parameters is investigated instead of the full set, thus the overall computational size is reduced by a large margin. Most notably, even when using a deep-learning rule-set with 3 iterations, the overall speedup of the iterative algorithm remains high, 47.05 on 64 cores.

Figure 5 plots the speedups for all three rule-sets covering all iterations. Although the runtimes vary widely between the rule-sets, as can be seen from the logarithmic scale of the runtime graph on the right hand side, the speedups remain high, above 40 on 64 cores, for all iterations. Each rule-set leads to a different number of MapReduce iterations, with each iteration processing a progressively increasing number of medical parameters. The brute-force approach processes all parameters within a single, compute-intensive iteration and therefore achieves the highest degree of parallelism but also the highest total runtime.

Studying the left graph of Figure 5 in more detail, one could observe that all MapReduce jobs achieve almost ideal speedups up to 16 cores of the Hadoop enabled Beowulf cluster. From that point onwards speedup performance slightly drops for all MapReduce jobs, but still shows good scalability throughout the experiment. One performance critical aspect of the iterative MapReduce implementation is the relative workload inside the individual iterations and their corresponding speedups. Here we note that, while the best speedups are achieved for the largest parameter sets (e.g., brute force and second iteration of shallow learning), even the smaller sets (e.g., first iteration of deep learning) yield comparable speedups and remain scalable. It is also important to mention that the execution times for the iterative learning scenarios are significantly faster, as depicted in the right graph of Figure 5. As an example, the single core runtime for the second iteration of the shallow learning scenario was found faster than the corresponding runtime of the brute-force scenario by a factor of 1.89. Although the difference drops exponentially as the parameter sets become smaller, it is still noticeable: as an example, 4702 seconds were required for the first iteration of the shallow learning scenario, whereas 3547 seconds was the runtime for the second iteration of the deep learning scenario.

Concluding, the aforementioned experimentation in our Hadoop enabled Beowulf cluster demonstrated that algorithmically our learning algorithm gains a significant improvement in end-to-end parallel execution time by dividing the overall computational size into multiple learning iterations, without sacrificing much scalability and speedup efficiency.

6. DISCUSSION

In this paper, we presented a prototype MapReduce [3] iterative learning algorithm for improving the diagnosis of mild HCM, an inherited cardiac disease, which is a major cause of sudden cardiac death in young athletes with "grey zone" cardiac hypertrophy [9]. The MapReduce iterative learning algorithm uses a rule-based approach inspired by machine learning [10, 1, 13] to improve the parameter set contributing to the main diagnostic algorithm, thus realising an iterative MapReduce structure that divides the computational workload into multiple learning iterations and reducing overall complexity. The MapReduce distributed computing framework is used to apply the diagnostic function on large data-sets and to classify the results. This massive data-parallelism, combined with machine learning techniques, brings many new possibilities for non-invasive diagnosis of mild HCM. Our work presents a first step towards this direction.

The core of our MapReduce iterative learning algorithm is a diagnostic scoring function that classifies medical records into either physiological or pathological. As a language comparison of scripting languages for Hadoop, in the context of medical computing, we developed this function in several Hadoop MapReduce languages: Java, Pig and Hive. Since Pig and Hive provide higher-level abstraction mechanisms, the programming effort is lessened: the Pig and Hive programs are 45–49% shorter than the Java version (measured in lines of source code). However, this gain in programmer productivity comes at the price of lower raw performance, achieving speedups of 61.15 for Java, 48.06 for Pig, and 48.08 for Hive, on 80 cores of our Beowulf Cluster. All versions, though, exhibit good scalability up to the maximal number of cores. Recent, better tuned, versions of Pig and Hive though, show the potential for this gap to narrow [17].

In experiments with up to 64 cores, the algorithm succeeds to reduce the end-to-end runtime from several hours on single core Hadoop setups to just a few minutes on our Hadoop enabled Beowulf cluster by just exploiting massive data parallelism. Our benchmarking proves that the multiple MapReduce iterations achieve significant performance gains over a brute force learning. The end-to-end runtime of a deep-learning rule-set, involving three MapReduce iterations, was 1.81 times faster than a brute force approach. In general, even with the most challenging iterative MapReduce job, the learning algorithm achieved speedups between 47.05 and 53.67 on a high configuration of 64 cores of our Hadoop enabled Beowulf cluster.

Medical applications are often associated with the processing of massive amounts of high-dimensional clinical data [13]. Patterns extracted out of such datasets, based on established or speculated dependencies, empower the medical community with new knowledge about how to accurately diagnose complex diseases. These results from experimenting with our prototype iterative learning algorithm demonstrate that MapReduce is highly relevant for achieving this purpose. Our implementation achieves a modular design, by separating the expert knowledge on dependencies in an easily tuneable rule-set. Therefore our framework should be applicable to a wider class of medical diagnostic problems that require data-intensive computations. In particular, it meets the medical experts' demands for feasible and accurate diagnosis of mild HCM in athletes.

In the near future we plan to evaluate our iterative learning algorithm on YARN, the next generation of the Hadoop distributed framework that was recently released for alpha testing. The decoupling of Hadoop's JobTracker into multiple modules is very promising towards achieving even greater scalability across hundreds of thousands of computing nodes. For our future benchmarking we plan to increase the size of our experimental medical datasets and use a larger cluster, potentially Amazon's EC2 or Microsoft's Azure (which recently announced support for Hadoop). Access to such resources and further experimentation could provide invaluable insight on the potential of MapReduce for large-scale medical research.

7. ACKNOWLEDGMENTS

We would like to thank our colleagues in the Dependable Systems Group of Heriot-Watt University for their helpful discussions and feedback. Furthermore, we would like to extend our gratitude to the medical staff of the Laboratory of Sports Medicine at Aristotle University of Thessaloniki for their cooperation throughout this multidisciplinary project.

8. REFERENCES

[1] C.-T. Chu, S. K. Kim, Y.-A. Lin, et al. Map-Reduce for Machine Learning on Multicore. NIPS, pages 281–288, 2007.

[2] D. Corrado, J. Drezner, C. Basso, et al. Strategies for the prevention of sudden cardiac death during sports. *Eur. J. Cardiov. Prev. R.*, 18(2):197–208, 2011.

[3] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. OSDI, pages 137–149, 2004.

[4] J. Dean and S. Ghemawat. MapReduce: a flexible data processing tool. *Commun. ACM*, 53(1):72–77, Jan. 2010.

[5] A. Deligiannis, E. Kouidi, N. Koutlianos, et al. Cardiovascular pre-participation screening of 22205 athletes: a Northern Greece seventeen years experience. *Int. J. Sport Med.*, subm. for publ.

[6] A. E. Epstein, J. P. DiMarco, K. A. Ellenbogen, et al. ACC/AHA/HRS 2008 guidelines for device-based therapy of cardiac rhythm abnormalities. *J. Am. Coll. Cardiol.*, 51(21):2085–2105, 2008.

[7] M. Isard, M. Budiu, Y. Yu, et al. Dryad: distributed data-parallel programs from sequential building blocks. EuroSys, pages 59–72, 2007.

[8] J. Lin, D. Ryaboy, and K. Weil. Full-text indexing for optimizing selection operations in large-scale data analytics. MAPREDUCE, pages 59–66, 2011.

[9] B. J. Maron. Distinguishing hypertrophic cardiomyopathy from athlete's heart: a clinical problem of increasing magnitude and significance. *Heart*, 91(11):1380–1382, 2005.

[10] T. M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[11] C. Olston, B. Reed, U. Srivastava, et al. Pig latin: a not-so-foreign language for data processing. SIGMOD, pages 1099–1110, 2008.

[12] E. Pagourelias, G. Efthimiadis, E. Kouidi, et al. Distinguishing the 'grey zone' between athlete's heart and hypertrophic cardiomyopathy: A pilot study. *Int. J. Clin. Pract.*, subm. for publ.

[13] N. Savage. Better medicine through machine learning. *Commun. ACM*, 55(1):17–19, Jan. 2012.

[14] M. Schatz. CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics*, 25(11):1363–1369, 2009.

[15] S. Sharma, B. J. Maron, G. Whyte, et al. Physiologic limits of left ventricular hypertrophy in elite junior athletes: Relevance to differential diagnosis of athlete's heart and hypertrophic cardiomyopathy. *J. Am. Coll. Cardiol.*, 40:1431–1436, 2002.

[16] K. Shvachko, H. Huang, and R. Chansler. The Hadoop Distributed File System. MSST, pages 1–10, 2010.

[17] R. J. Stewart, P. W. Trinder, and H.-W. Loidl. Comparing high level MapReduce query languages. APPT, pages 58–72, 2011.

[18] A. Thusoo, J. Sarma, N. Jain, et al. Hive—a petabyte scale data warehouse using Hadoop. ICDE, pages 996–1005, 2010.

[19] F. Wang, V. Ercegovac, T. Syeda-Mahmood, et al. Large-scale multimodal mining for healthcare with MapReduce. IHI, pages 479–483, 2010.

[20] T. White. *Hadoop: The Definitive Guide, 2nd Edition.* O'Reilly Media, Yahoo! Press, 2010.

[21] M. H. Zweig and G. Campbell. Receiver-operating characteristic (ROC) plots: a fundamental evaluation tool in clinical medicine. *Clin. Chem.*, 39:561–577, 1993.