Expediting assessments of database performance for streams of respiratory parameters

Charles J Gillan[1], Aleksandar Novakovic[2], Murali Shyamsundar[3], Adele H Marshall[2] and Dimitrios S Nikolopoulos[4]

[1]The Institute for Electronics Communications and Information Technology,
School of Electrical and Electronic Engineering and Computer Science
Queen's University Belfast
Queen's Road, Queen's Island, Belfast,
Northern Ireland BT9 3DT, United Kingdom

[2]School of Mathematics and Physics
Queen's University Belfast, University Road, Belfast
Northern Ireland BT7 1NN, United Kingdom

[3]The Centre for Experimental Medicine School of Medicine, Dentistry and Biological Sciences
Queen's University Belfast 97 Lisburn Road, Belfast
Northern Ireland BT9 7BL, United Kingdom

[4]Department of Computer Science,
School of Electrical and Electronic Engineering and Computer Science Queen's University Belfast
18 Malone Road Belfast, Northern Ireland BT9 5BN, United Kingdom

***Abstract*—A new methodology is proposed to compare database performance for streams of patient respiratory data from patients in an intensive care unit. New metrics are proposed through which databases may be compared both for this and similar streaming applications in the domain of the Internet of Things. Studies are reported using simulated patient data for four freely available databases. The statistical technique of non-parametric bootstrapping is used to minimise the total running time of the tests. We report mean values and bias corrected and accelerated confidence intervals for each metric and use these to compare the databases. We find that, among the four databases tested, ScaleDB is an optimum database technology when handling between 200 and 800 patients in this application, while PostgreSQL performs best outside of this range. Comparing the non-parametric bootstrapping method to a complete set of tests shows that the two approaches give results differing by a few percent.**

I. Introduction

In previous work in the FP7 NanoStreams project [1], [2] we studied the performance and energy efficiency of servers consuming a financial data feed and computing option contract prices in real-time from the stock prices contained in the data feed. In this present paper we apply and extend the methodology to the case of streams of patient respiratory data. We have developed a novel real-time computer-based screening system that we deploy in an intensive care unit (ICU) in a hospital in Belfast. Our system monitors parameters associated with lung physiology in order to alert clinicians about possible ventilator induced lung injury (VILI) [3]. For this reason we have named our system VILI Alert. Ventilator induced lung injury is a common phenomenon in critical care units. This is a preventable condition where protective lung ventilation not only prevents further lung injury but also prevents the de novo development of lung injury in healthy lungs [4]. Inappropriate use of excessive tidal volumes and other settings of the mechanical ventilator such as the positive end expiratory pressure could result in initiating and propagating VILI. Various phenomena such as volutrauma, barotrauma, atelectotrauma and biotrauma lead to VILI. Despite this robust evidence, the translation of protective lung ventilation is still inadequate as demonstrated in observational clinical studies and highlights the urgent need for a system such as the VILIAlert system to support clinical decision making and improve compliance with best evidence based practice [5]. Development of smart intuitive clinical decision support systems such as VILI Alert therefore have the potential to improve clinical practice and save lives [4]. Our VILI Alert system monitors patients in real-time by continuously computing a set of metrics from the received streams of ventilation data. Mathematical kernels process the data streams to allow patients to be monitored against set thresholds. When a

threshold is violated consistently, an alarm is immediately raised and sent by SMS message to clinical staff.

One difference between our current and previous work lies in the amount of fixed and variable data that has to be handled. For financial options there are only a few fixed parameters such as date of expiry and interest rate, and only one continuously changing value, the stock price. When computing clinical metrics multiple parameters associated with the patient need to be considered, some fixed or very slowly changing such as age or weight, and some changing with every breath, such as tidal volume. In the case of financial options the stock price is the only value that changes regularly. Moreover we are only concerned about the current price at any point in time. In regard to human lung physiology in an ICU, we are interested in averaged values over fifteen minute time bins and also trends over twelve hours. A fundamental component of our ventilator monitoring system is therefore a database whereas in the case of our market data application this is absent, with prices being consumed off the wire. Furthermore the nature of the mathematical kernels is different. There is a considerable amount of logic code that needs to be exercised before time series of tidal volume and airway pressure data can be processed to compute various clinical metrics.

A further distinction with the process of pricing financial options lies in the way that the data is gathered. Human physiology is a continuous process measured by sensors which can be set to take recordings at predefined time intervals before forwarding to a central database. These data points are subject to noise generated, for example, by a patient moving in their bed. Data even may be filtered so that only every third or fourth reading is transmitted to the database. In part, this can be due to network latency and/or the total storage available for the database. By comparison, stock market price changes are discrete events in time, generally taken to follow a Poisson distribution over a trading day. In this paper we repeat experiments with patient data arriving at different fixed intervals. This mirrors the real-life setting where sensor recordings are taken at different intervals. In addition we vary the number of patients and push this to very large numbers, well beyond the actual numbers that one encounters in a hospital setting today. This is in order to stress test the database to the maximum extent possible.

Benchmark testing often involves repeating the same experiment for several iterative rounds (usually five or ten iterations) in order to obtain more reliable estimations of workload metrics. There is no universal rule on how many repetitions of the same experiment that one needs to perform, but obviously more repetitions lead toward better conclusions. By repeating the experiments multiple times, one covers the situation where there is underlying unknown variability.

In some cases completing a single experiment may be very expensive and time consuming. For example, in our use case, completing all simulations (on all databases against all scenarios) would take close to thirty hours elapsed time. This means that if we have had to repeat these simulations at least ten times we would have executed for almost twelve days continuously, which is a prohibitively long time.

In order to capture the unknown variability of the collected measurements with minimal repetitions of the same experiment, we used the non-parametric bootstrap re-sampling technique pioneered by Efron and co-workers [6], [7], [8]. This means that we estimate our performance metrics by sampling from the empirical distribution function of the experimental results. We are also able to extract a correlation between metrics from the data. The question automatically follows as to whether, or not, the non-parametric bootstrap re-sampling technique gives a good approximation to actually repeating the experiment many times. We address this by actually performing the experiments and comparing results. We find that the re-sampling technique generally gives answers within a few percent of those obtained by very many repetitions of the experiments. This is one of the key contributions of our paper, extending beyond the work that we have previously reported for financial options [1], [2].

Our paper is composed of several sections. Section II, which follows next, briefly defines the physiological monitoring that lies at the core of this work and also presents our VILI Alert system. Section III explains the testing strategy that we developed using the non-parametric bootstrapping technique to measure performance of the databases in our study. In section IV we present and analyse the results that we obtained. We discuss related work in section V and provide some further details on how this work fits into the wider context of the FP7 NanoStreams project that explores micro-server technologies for real-time data analytics. Section VI discussed the accuracy of the non-parametric bootstrapping technique as opposed to massive numbers of repetitions of the same experiment. We finish the paper with summarising conclusions drawing from our present work in VII.

II. Physiological monitoring of respiratory parameters

Inflation of the alveoli during mechanically assisted lung ventilation generates stress forces which in turn create strain on the cells which may lead to damage. The stress forces created by the inflation process are proportional to the tidal volume (TV), a parameter that is defined at the mechanical ventilator. TV also depends on gender and on ideal body weight [9]. From the perspective of an IT system, the ventilator is a data source which outputs information, meaning that it is in an abstract sense similar to other sensors. Many medical sensors, including ventilators, are engineered to produce their application layer data in the Health Level Seven protocol (HL7) [10]. The HL7 data is typically carried over an IP network, using TCP at the transport layer, to a medical electronic medical record system where it can be processed and stored for inspection.

We developed a system to monitor TV and other airway pressure parameters associated with the respiratory phys-

iology of patients and deployed this at the Royal Victoria Hospital Belfast in the Regional ICU which is located there. Our system intercepts the data feed from the ventilators and runs along side the electronic medical record system deployed there. This design avoids any impact on the record system. All patients receiving ventilation in the ICU were monitored continuously by the computer system. We operated this without interruption for ten calendar months starting in November 2015, tuning the software as we gained operational experience. Data streams from patients can begin, or end, at any time of the day or night, an aspect that requires careful handling of the calculations based on time windows.

In this paper we focus on the performance of the database, a component which is at the heart of the system. We create experiments to examine the performance of the database for many more patients and much more data than we have available in the hospital environment. In practical terms this means that we created an instance of the VILI system in our lab and used only simulated data, an advantageous consequence of which was the fact that we did not need to address ethical or governance issues using this data.

III. Methodology

The primary aim of this study is to push the alert database, into which patient readings are streamed continuously, well beyond its current operational range and so identify appropriate scalability characteristics. While the system deployed in the live hospital environment is only required to cope with twenty patients, and has further constraints imposed on it in order to inter-operate with existing hospital equipment and IT systems, we seek in this work to scale the VILI system along two axes: the number of patients and the rate of data collection per patient. As such this looks forward to the situation in which all patients within a large hospital may have physiological sensors routinely attached during their stay or indeed after they are discharged into the community.

A. Database Selection

We conducted our academic study using the databases described below. All of these were used as freely downloaded versions, without commercial support. We should also note that all selected databases can be installed to all major operating systems (Windows/MacOS/Linux/Unix/BSD). It is expressly not the purpose of this paper to make any comment of any kind whatsoever on fitness for purpose of any commercial offering.

- **PostgreSQL** is an object-relational database, ACID-compliant and transactional by design, and which is very popular in web hosting applications.
- **MariaDB** is a fork from the MySQL database available under GPLv2 license, committed to open development and transparency, and offering improved performance, and better security and availability in comparison to MySQL [11], [12].
- **ScaleDB** is a closed source, but freely available, proprietary storage engine produced by ScaleDB Inc., which is pluggable and compatible with MariaDB and MySQL databases [13]. It is based on a Shared-disk Database Storage Architecture [14] and is designed to offer high performance, scalability and availability for applications with many concurrent users and large datasets.
- **MonetDB** is a column-oriented database management system which aims to deliver high performance on complex queries against large tables. It was designed to be suited to on-line analytical processing, and data mining applications[15]. This database pioneered the technique of incremental partial indexing and sorting of the data, method which shifts the cost of index maintenance from updates to query processing.

Although there is a wide variety of open source database management systems that could be used in our study, we have decided to use those mentioned above because they are all of the relational type with full SQL support, offering a high level of security, scalability and performance. Another significant factor in our selection of these databases is that they are used in many applications. As the measure of popularity of databases, we have used DB-Engines Ranking - a monthly updated comprehensive list, containing more than 300 commercial and open-source DBMS which are ranked by their popularity [16]. The DB-Engines Ranking criteria [17] are based on the robust scores that are calculated using various metrics such as: the number of web citations i.e. the number of results in search engine queries in which the system appeared, the number of available job opportunities which require the knowledge of the particular system, general interest in the system, expressed as the frequency of searches over time obtained from Google Trends, etc.

According to DB-Engines Rating, the PostgreSQL was chosen as the DBMS of the year 2017 as it had the highest popularity score growth measured in the period between January 2017 - January 2018, while the MariaDB, popularity score of which almost tripled in the last three years, came up as the third [18], [19]. It is worth adding that the second place belonged to a database management system (DBMS) which is mainly used as a search engine, and hence it was automatically ruled out of our study.

The version numbers for each downloaded database, as used in our experiments, are shown in table I. We

TABLE I
Details of the databases studied and their corresponding drivers

Database	Version	JDBC driver
MonetDB	1.7.Jun 2016-SP1	monetdb-jdbc-2.17
MariaDB	10.1.13	mariadb-java-client-1.5.0-RC1
ScaleDB	16.04	mariadb-java-client-1.5.0-RC1
PostgreSQL	9.5.3	postgresql-9.4.1208

followed the standard installation documentation supplied with each database making no specific configuration adjustments other than setting unique TCP port numbers for each. In our experimental setup we dedicated one

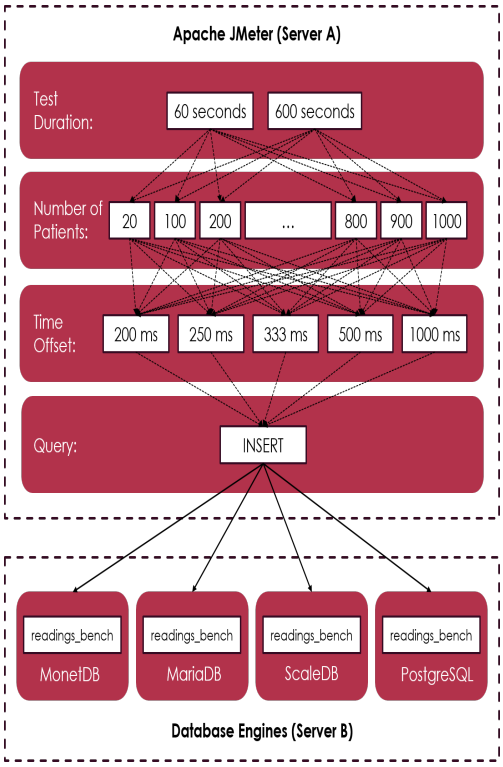

Fig. 1. The layers of variability which multiply to give a large volume of tests. Working from the top down, the selection of test options available at a layer is multiplied by the number of tests at the next lower layer.

server, which we label Server B in figure 1, to execute the database instance. It has the following physical configuration: AMD Opteron(TM) Processor 6272, 2.1 GHz, 8 Cores, 2MB Cache Memory, 63 GB RAM. The operating system executing on the server is CentOS 6.8. During each experiment, one and only one of the databases in table I executed on the server.

B. Definition of Metrics

Our previous work presented three metrics: Seconds per Option (S/Opt), Joules per Option (J/Opt) and Quality of Service (QoS) [1], [2]. Replacing an execution of an option pricing kernel with the execution of an SQL INSERT, these metrics can be extended directly to milliseconds per Insert (MS.INS) and Joules per Insert (J.INS) with the QoS similarly redefined in terms of successful insert operations.

In this work we measure S/Insert in milliseconds. We set up our tests so that they marked an insert request as a failure if the required processing time exceeded a threshold, in the region of seconds. We investigated the QoS metric by performing SQL INSERT operations at the various fixed time intervals, so that the QoS metric is the ratio of successful to total SQL INSERT requests for the full duration of a test.

We found that analysis of the database performance requires the following additional metrics:

- **AVG.INS.S:** Average number of successfully committed SQL INSERT operations per second.
- **AVG.RAM:** Average RAM consumption in gigabytes per test.
- **AVG.INST.P.CONS:** Average instantaneous power consumption in Watts.
- **AVG.CPU:** Average percentage CPU usage.

Several of the charts in our results section refer to these new metrics as well as to the extended metrics above.

C. Performance Test Tools

We performed a series of extensive simulations using the Apache JMeter application. It is an open source, Java-based, benchmarking tool and is designed for performance testing on both static and dynamic resources. We used JMeter to simulate heavy loads on our databases and to collect overall performance metrics including response time and standard system parameters CPU, memory and disk usage, etc. [20]. JMeter uses JDBC drivers to communicate with the database and table I presents the exact version of each driver used in our simulations.

We created multiple JMeter scripts each corresponding to a different type of test and, in order to remove interference effects, executed these scripts on a different server than the one on which the databases executed. The server on which JMeter executed is labeled as Server A as shown in figure 1. On the one hand tests covered different numbers of patients and on the other different rates for gathering patient data. All tests centre on an SQL INSERT operation placing patient readings into one table in the database. The INSERT operation looks typically as follows:

```
INSERT INTO readings_bench
  (
   PatientGender, PatientID,
   PatientHeight, PatientWeight,
   ControlID,     ObservationDateTime,
   Unit,          BedID,
   TidalVolume
  )
VALUES
  {
   'M', 'PatientZA',
   142.0, 95.0,
   'ControlID123', now,
   'RH_RICU,'RICU20',
   123
  }
```

To measure AC power usage at the wall socket of the Server B during execution of the JMeter scripts, we used a Watts Up Pro power meter [21]. This logged measurements at its maximum rate of one per second with an accuracy of 1%.

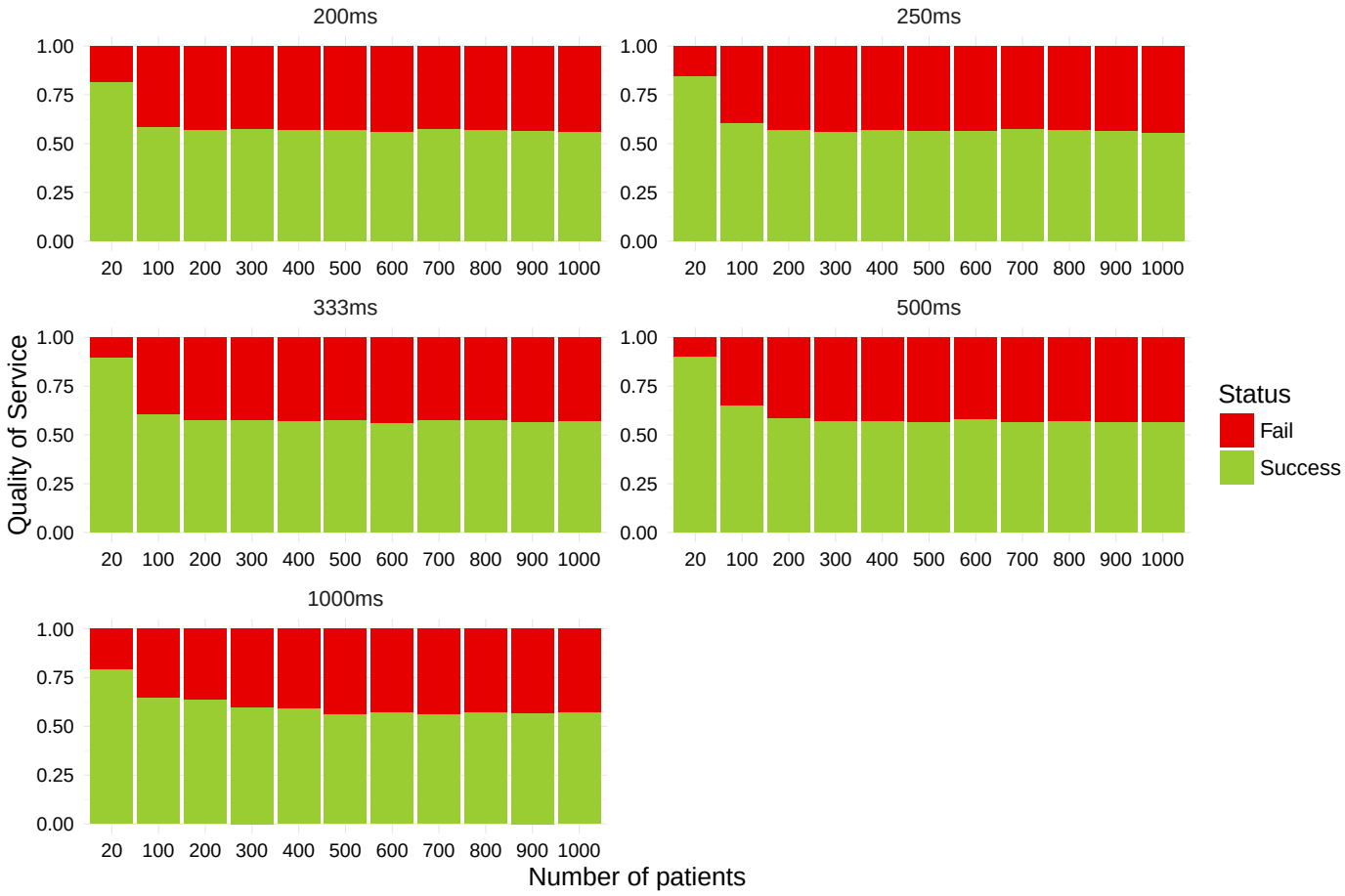

Fig. 2. QoS Metrics for MonetDB as a function of number of patients and data insert rate. The high number of fails shown indicated at an early stage, as discussed in the text, that MonetDb was not going to be suitable for our ICU application.

TABLE II
PERFORMANCE DATA FOR EACH DATABASE.

Database	Test Duration (s)	Number of Successes	Overall QoS (%)
MariaDB	60	1,756,864	100
MonetDB	60	950,169	58
PostgreSQL	60	1,770,740	100
ScaleDB	60	1,775,044	100
MariaDB	600	18,630,832	100
PostgreSQL	600	18,603,867	100
ScaleSQL	600	19,338,968	100

D. Two stage test approach

Our series of extensive simulations required the creation of multiple tests and then the execution of each test multiple times. We developed a strategy that had two distinct steps:

1. In the first step each test script lasted for only sixty seconds, that is for one execution of the test. In order to cover all of the possible combinations of number of patients and database software, we created a total of 220 test scripts. The purpose was to identify if there were any databases performing significantly worse than the others. The basis for comparison was the set of metrics which are described in the following section. Clearly, any databases performing badly could be eliminated at this step saving time in step two.
2. In the second step each test lasted for six hundred seconds for one execution thereby probing how the different databases perform under very high, constant load.

An overview of the work flow for each test script is presented schematically in figure 1. The options presented at each layer multiply together from top to bottom of the figure to define the test. Each time that a test executes we report the values of the metrics observed, therefore creating a set of samples for the metrics. Finally we compute the metrics for that test by analysing the mean and variance of the samples.

E. Non-parametric bootstrapping method

We observed that the second step in our experiments required very long run times, sometimes up to nine hours elapsed time, in order to execute very one of the available test scripts. This was not ultimately feasible and we sought a statistical method instead. We found that the non-parametric bootstrapping procedure [6], [8] was an efficient way to extract results requiring fewer executions of each test script. This statistical method [22], [23], [24] is well

suited to our case because it does not place restrictive assumptions on the statistical properties of the collected data. It is used in analysis of medical data [25] because the method is widely applicable and offers a solution to situations where conventional methods may be difficult or impossible to apply, such as finding the confidence interval of the median.

Let the set of n values for one metric produced by independent tests be a vector $\mathbf{y_{obs}}$. The general algorithm for a non-parametric bootstrap is then defined by the following steps, in which bold face type denotes a vector:

- Sample n values randomly with replacement from $\mathbf{y_{obs}}$ in order to obtain a bootstrap data set, which we denote $\mathbf{Y}^*$.
- Calculate the bootstrap version of the metric of interest, $\hat{\theta}^* = \hat{\theta}(\mathbf{Y}^*)$
- Repeat steps 1 and 2 a large number of times, which we denote by B, to obtain an estimate of the bootstrap distribution.

The distribution of values, $\hat{\theta}^*$, computed above can then be used to compute a mean value and a confidence interval (CI) for the metric. In our work we use the bias corrected and accelerated method (BCa) for derivation of the CI from the distribution [7], [25]. BCa allows for skewness in the distribution $\hat{\theta}^*$.

The main idea behind bootstrapping is that the population and empirical distribution functions converge as B increases to infinity. Following recently reported work [23] we used B = 10000 bootstrap samples.

IV. Results

We begin by discussing the outcome from the first step in our test strategy and continue with a comprehensive analysis of the data collected from the second step.

A. Outcome from step one

Table II reports the observations from the initial step in which we ran short sixty second jobs. The most obvious result is that MonetDB had the worst QoS performance. The tests recorded 698, 807 failures in total leading to an overall QoS of 58%. On further investigation, we noted that the logs contained the following message for the failed SQL INSERT operations with MonetDB

```
COMMIT: transaction is aborted because of
concurrency conflicts, will ROLLBACK instead
```

These failures occurred in all tests using MonetDB along both scalability axes, the number of simulated patients and the data collection rate was set. Figure 2 presents a more detailed analysis of the behaviour of MonetDB over both axes. It shows that if MonetDB were used in the live hospital environment, that is with twenty connected patients, a QoS of in the range $[79\%, 90\%]$ would be obtained depending on the data insert rate. Increasing the number of patients to 100, or more, leads to a decreasing QoS.

The reason for this poor performance of INSERT queries lies in the fact that MonetDB is primarily designed as an analytical (OLAP) database, where most operations are expected to be be SELECT-FROM-WHERE-GROUPBY queries. Its transaction management scheme is optimized for reading large chunks of data, instead of writing small data chunks at the high speed concurrently. Therefore, we concluded that MonetDB would not be a good choice in this study and decided to drop it from stage two tests.

B. Outcome from Step Two

Step two testing proceeded using MariaDB, PostgreSQL and ScaleDB, all of which can be seen in table II to have similar performance figures. Removing MonetDB from the test set reduced the number of test scripts to 165 simulations in total with a single run of each test executing for 600 seconds in step two.

1) Correlation coefficients: Figure 3 is a visual presentation of the correlation matrix between our metrics. The figure presents only statistically significant Pearson's correlation coefficients, at the 95% level and 163 degrees of freedom. We believe that this visual style gives a clear overview of linear relationships between workload metrics.

Each metric is a node in the graph and the proximity of the metrics to each other represents the overall magnitude of their correlations. Thus clustering of the metrics is easily seen in this presentation. Each path represents a correlation between the two variables at either end. Blue and red paths represent positive and negative correlations respectively, while the transparency and the width of the path represent the strength of the correlation. Thinner and more transparent paths mean weaker correlation.

In figure 3 we can see that Patients Count, AVG.RAM, AVG.INS.S and J.INS form one cluster, while AVG.INST.P.CONS, AVG.CPU and MS.INS form another distinct cluster. This means that increasing the number of patients has more impact on RAM usage than on CPU usage. This also means that databases that rely more on CPU than on RAM to handle the increased number of patients tend to have higher instantaneous power consumption than the databases that rely more on RAM.

Apart from that, increased CPU usage implies an increment in the MS.INS metric. The best examples for this are ScaleDB and PostgreSQL, both of which had similar performance regarding to AVG.INS.S metric in table II. ScaleDB handles an increased number of patients by using more CPU power and thus having the highest INST.P.CONS metric, while on the other hand PostgreSQL relies more on RAM and therefore have the lowest INST.P.CONS metric. Similarly ScaleDB had the highest, while the PostgreSQL had the lowest MS.INS metric. It is interesting to note that the data insert rate (time offset) does not have much impact on calculated workload metrics. As we can see in the correlation plot, the data insert rate only affects how fast the maximum values of AVG.INS.S metric can be met.

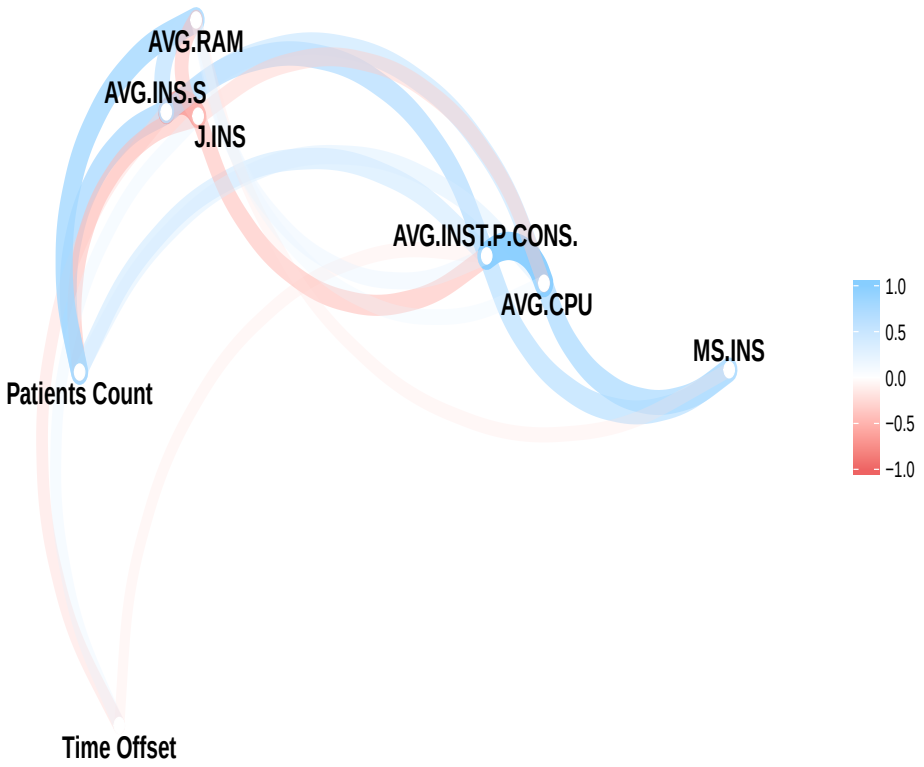

Fig. 3. Visual presentation of the Pearson correlation coefficients from step two tests. Each metric is a vertex in a graph, while each edge represents the statistically significant correlation coefficient between the two metrics at either vertex. The transparency and the width of each edge depends on the strength, while their colour depends on the direction of the correlation coefficient (A thinner and more transparent edge - weaker correlation; blue colour - A positive correlation; red colour - A negative correlation)

2) Workload metrics: Using the bootstrap procedure we computed values for all metrics as a function of eleven different values for the number of patients and five different data insert rate scenarios, presenting these in figure 4 using a data shading approach. The grey colour denotes the calculated workload metrics for all five data rates, the shape of each grey box distinguishes the database used. This means that working upwards on a vertical line from each number of patients on the x-axis, we can see one instance of each shape on that line.

The restricted data rate of 1 insert per second is the one that is used in the hospital setting. For this reason we present coloured shapes for this data rate to distinguish it from all other data rates. These are as follows: MariaDB - red circles, ScaleDB - blue squares, PostgreSQL - green triangles Using this data shading technique we managed to identify patterns for our workload metrics presented in the figure.

The AVG.INS.S metric shown in figure 4(a) increases as the number of patients grows, until it reaches a maximum value and thereafter remains almost constant. The rate of increase depends on the data insert rate and the database technology in use. For instance, in the case of MariaDB and ScaleDB, maximum values were reached between 200 and 800 patients, depending on the data insert rate, with a faster data insert rate implying faster convergence toward the maximum value obtained. On the other hand, in the case of PostgreSQL this maximum value was reached between 700 and 800 patients again depending on the data insert rate. The fact that PostgreSQL reaches its maximum value much slower than the other databases explains why it had the lowest total number of successful inserts in table II.

Similar conclusions can be applied in the case of the AVG.INST.P.CONS and AVG.CPU metrics in figures 4 (c) and 4 (d). For different data insert rates, the maximum values of these metrics occur at the same number of patients, as in the case of AVG.INS.S metric. ScaleDB seems to be the most resource hungry database, consuming the most CPU and power resources, and it is followed by MariaDB and PostgreSQL in that order.

As the J.INS metric depends on AVG.INS.S and AVG.INST.P.CONS, the same rules can be applied as for previous metrics. However, instead of increasing, in the case of J.INS metrics we have a decreasing trend as the number of patients grow, until it reaches its minimum and remains almost constant afterwards. Again, a faster data insert rate implies a faster convergence towards a minimum value (Fig. 4 (e)).

Speaking of the AVG.RAM metric, the average RAM usage linearly increases with the increased number of patients. PostgreSQL consumes the most RAM resources, and it is followed by the ScaleDB and MariaDB. As seen by the shaded data points, there does not seem to be much variation in RAM consumption between data insert rate scenarios (Fig. 4 (b)).

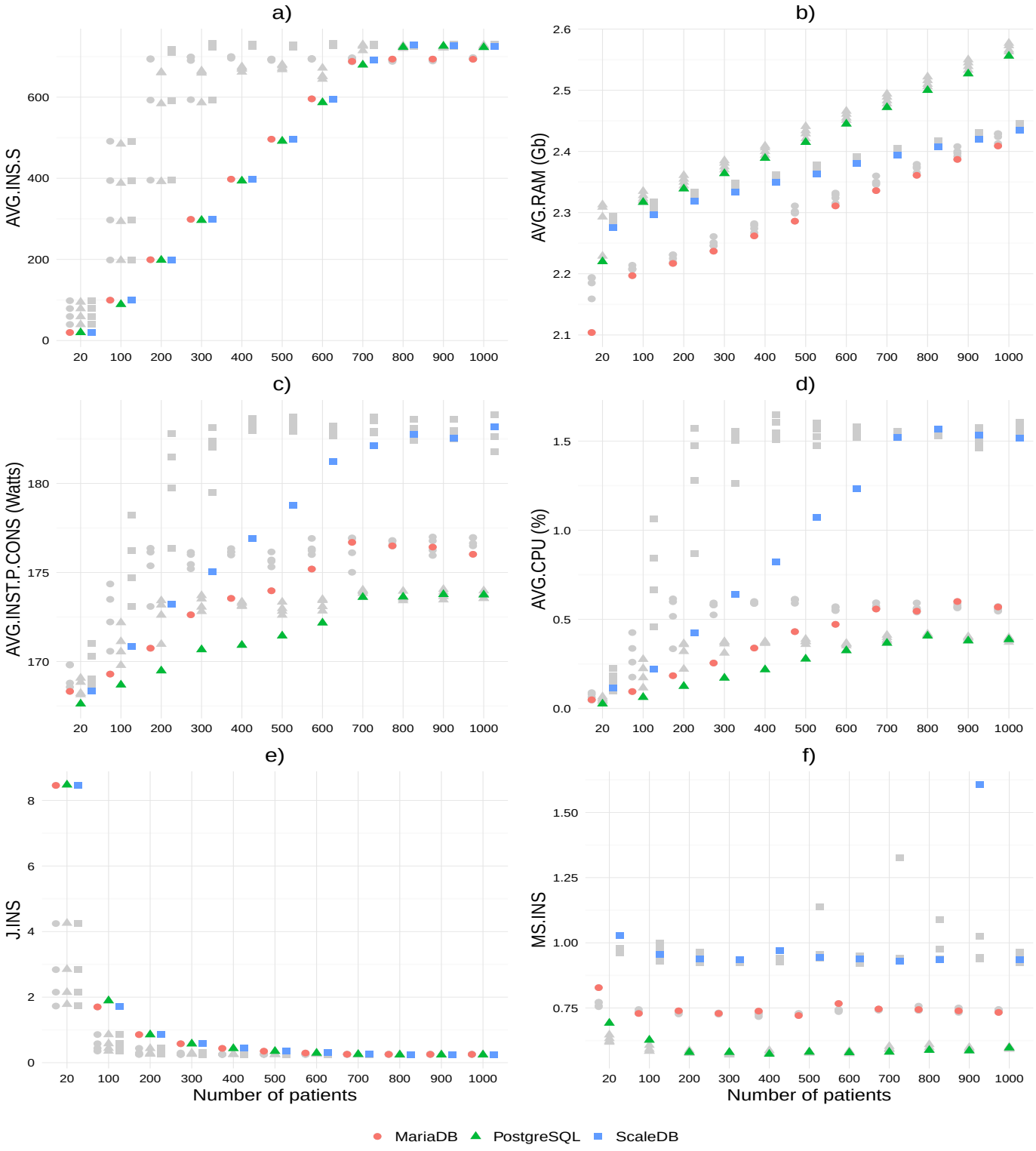

Fig. 4. Metrics obtained using the non-parametric bootstrap procedure in step two for varying numbers of patients. Databases are presented as the following shapes: MariaDB - circles, PostgreSQL - triangles and ScaleDB - squares. The coloured shapes are for a data rate of one insert per second (current hospital settings), while the shapes in grey correspond to other data insert rates (between 2 and 5 inserts per second). This data shading approach allows a cross-comparison of collected workload metrics between mentioned databases by taking into account varying data insert rates and different number of patients, as well as the identification of emerging patterns.

No matter what number of patients and time offset settings were used in our simulations, each database had almost constant average MS.INS metric. Best performance was recorded in the case of PostgreSQL, followed by MariaDB and ScaleDB (Fig. 4 (f)).

3) Achieving maximum performance: Table III lists the minimal number of patients required for each database to reach its maximum value for the AVG.INS.S metric. Achieving this maximum for that metric also implies that the maximum values of metrics AVG.INST.P.CONS and AVG.CPU are met as well. In fact there are very small variations between different numbers of patients and data insert rates. Most importantly, increasing the number of patients above values presented in table III will mean only increased RAM usage without loss of performance.

TABLE III
The number of patients for a database required to meet its maximum performance

Insert time (ms)	MariaDB	ScaleDB	PostgreDB
200	200	200	700
250	200	200	700
444	300	300	700
500	400	400	700
1000	700	800	800

A refinement on table III is to consider the confidence intervals for each metric that we calculated. Table IV reports the mean value for the metrics as well as the upper and lower limits of the calculated 95% BCa confidence intervals for each metric. From table IV, we can propose that if a system were required to support between 700 and 1000 patients, PostgreSQL would be the best choice. It has a slightly lower average number of inserts than ScaleDB and consumes slightly more RAM but has the best energy efficiency (i.e. the lowest AVG.CPU, AVG.INST.P.CONS and J.INS metrics) and the lowest MS.INS metrics of all three candidates.

For the cases between 200 and 700 patients, independent of data insert rate, either ScaleDB or MariaDB are the better performers. Although it consumes more resources and has a higher MS.INS metric, ScaleDB has the highest AVG.INS.S metric and lower J.INS metric making it a more attractive choice. Below 200 patients all databases performed similarly in a terms of the AVG.INS.S metric, making PostgreSQL the better choice as it consumes less resources then the other databases

TABLE IV
Workload metrics when each database operates at maximum performance

Database	Mean value	95% BCa CI limits lower	upper
AVG.INS.S			
MariaDB	693.916	692.884	694.888
PostgreSQL	724.341	722.497	725.566
ScaleDB	727.324	725.446	728.391
MS.INS in milliseconds			
MariaDB	0.737	0.734	0.740
PostgreSQL	0.595	0.592	0.598
ScaleDB	0.982	0.953	1.051
AVG.INST.P.CONS in Watts			
MariaDB	176.294	176.127	176.443
PostgreSQL	173.756	173.666	173.847
ScaleDB	182.969	182.779	183.128
J.INS in Joules			
MariaDB	0.254	0.254	0.254
PostgreSQL	0.240	0.240	0.240
ScaleDB	0.252	0.251	0.252
AVG.CPU in %			
MariaDB	0.577	0.570	0.584
PostgreSQL	0.397	0.390	0.403
ScaleDB	1.548	1.535	1.561
AVG.RAM in GB			
MariaDB	2.338	2.319	2.356
PostgreSQL	2.528	2.514	2.541
ScaleDB	2.393	2.382	2.403

V. Related Work

While there are many investigations of database performance in the literature, in so far as we are aware, no other papers have analysed database performance in an ICU application with either a similar methodology or to the same level of depth as we have presented in this paper. In the following paragraphs we mention related from the literature on database performance.

Schmid and co-workers [26] used JMeter to investigate the performance of a relational-database (PostgreSQL) and a NoSQL-database (MongoDB). They performed geo-calculations on the database level as well as a creating a backend for web mapping services (WMS) provided by GeoServer. They were focused on analysing response times of three geo-queries (geo-function *within* on points, lines and polygons) against three different dataset sizes (small (38.9 Mb), medium (501 MB) and large (2.1 GB)), three different data query time offsets (random, 0.5 an 1 seconds) and three different user categories with varying number of users depending whether they tested WMS performance (10, 25 and 50 users) or performance on database level (100, 250 and 500 users). They conducted all performance tests using Apache JMeter.

Results of the WMS tests showed that MongoDB had a little faster average response time only in the case when geo-function "within" was performed on points, while for all the other cases PostgreSQL had better performance. The response time for PostgreSQL/GeoServer tend to increase just a little with an increasing number of users, while the response time of MongoDB/GeoServer is significantly growing with an increasing number of users. On the other hand, when the performance were measured purely on database level, PostgreSQL's response time rapidly increases with the size of dataset and therefore it performed better for small datasets and complex geometry types like lines and polygons. For MongoDB the size of the dataset does not play a big role, and it had a high performance even on large datasets, which seems to be related to the way how MongoDB handles geohash-index.

Wang and co-workers [27] reported a survey of the performance of in-memory databases related to high frequency trading of financial securities. They report that relying solely on memory to enhance the processing speed

is not sufficient to meet the needs of processing financial transactions online at high speed.

Difallah and co-workers [28] developed an open source, multi-threaded benchmarking framework called OLTP-Bench, which can be used to test the performances of any JDBC-enabled relational database. The framework supports over fifteen domain specific benchmarks consisting of standard Transactional benchmarks such as TPC-C, as well as the modern Web-Oriented and Feature Testing benchmarks such as LinkBench [29] and YCSB [30].

On-line transaction processing systems (OLTP) were one of the first widespread uses on relational databases when such databases became common in the Eighties. OLTP applications typically have a large number of users but each of which conducts only short transactions. Among the business use cases are: entry of new product orders, conduct of retail sales and holding data on customer relationship management (CRM). The OLTP-Bench is an extensible framework with the capabilities that enable researchers to dynamically control throughput rates, transaction workload mixture and workload skew during the execution of experiments in order to recreate real system loads. By extending it, Gobel [31] created the MuTeBench framework, which combines dynamic and flexible control features of the underlying OLTP-Bench, and allows creation of OLTP benchmarks for multi-tenant databases. Aken and co-workers [32] demonstrated dynamic and flexible control features of the framework through BenchPress, a graphical user interface that allows users to control the OLTP-Bench's behaviour in real time. OLTP applications and applications streaming data from Internet of Things use-cases represent two quite different sets of operational characteristics for the underlying databases that act as a permanent store for the data. With streaming IoT applications the data flows continuously into the system, several times per minute basis in our case. Analysis kernels, on the other hand, run at much less frequent intervals in our system but may extract large amounts of data on which to perform computations.

VI. Assessing the accuracy of the non-parametric bootstrapping method

In order to test the accuracy of our non-parametric bootstrapping approach in comparison to traditional benchmarking approaches, we conducted a series of complex for hunderds of times. The workflow of the experiment is as follows:

- We used our framework built on top of Apache JMeter and ScaleDB database to simulate patients' behaviour in intensive care unit of a hospital in Belfast. We simulated the behaviour of 20 patients with data insertion rate of one insert per second (current hospital settings).
- We were interested in the accuracy of four workload metrics introduced above: AVG.INS.S, MS.INS, AVG.RAM and AVG.CPU.
- To obtain as good as possible estimations of these workload metrics, instead of 5 or 10 repetitions of the same experiment (as it is usually done in traditional benchmark analytics), we performed 500 repetitions which resulted in a total running time of almost 84 hours.
- As it is done in traditional benchmarking analysis, we estimated values of our workload metrics using the following formula:

$$\text{Metric} = \frac{1}{500}\sum_{i=1}^{500}\text{Metric}_i \tag{1}$$

where Metric_i stands for one of the defined metrics above.
- Using the non-parametric bootstrapping procedure based on collected measurements for each simulation (out of 500) we created $B = 500$ *artificial* datasets. The purpose of this step was to compare how the computed values of workload metrics derived from the artificial datasets differ from the values obtained from the traditional approach.

To assess accuracy level we used the mean absolute percentage error (MAPE) described with the following formula [33]:

$$\text{Metric}_{\text{MAPE}} = \frac{1}{5}\sum_{i=1}^{500} \mid \frac{\text{Metric} - \text{Metric}_i^{\text{B}}}{\text{Metric}} \mid \tag{2}$$

An overview of mean absolute percentage errors for each metric, as well as the overview of 95% confidence intervals of these errors is shown in table V. As can be seen, estimations of workload metrics using the non-parametric bootstrapping approach represents a powerful alternative to the traditional approach.

TABLE V
Non-parametric bootstrapping accuracy assessment

Workload	MAPE	95% CI limits	
AVG.INS.S	0.08%	0.076%	0.084%
MS.INS	5.50%	5.179%	5.818%
AVG.RAM	1.67%	1.572%	1.762%
AVG.CPU	12.37%	11.634%	13.111%

VII. Conclusions and Future Directions

In this paper we have presented a methodology for comparing performance of various databases for streams of patient respiratory parameters in an ICU. Our methodology considers the impact of scaling both data insert rates and patients numbers, on the performance, energy-efficiency, system resources usage of four different freely available databases. Our methodology uses the statistical method of non-parametric bootstrapping in order to minimise the time taken to execute experiments.

It is expected that hospitals will increasingly deploy more sensor based monitoring outside the ICU, that is to a wider population of patients[34], something which invariably requires higher performance from the databases used. More generally, sensor based applications within the emerging Internet-of-Things (IoT) will stream data in a

fashion similar to that which we find in the ICU today, making results in this paper of wider interest therefore. Our results show that MonetDB would not be a good database engine for sensor applications as its transaction management scheme is optimised for reading large chunks of data instead of concurrent writing of small data chunks at the high speed [35]. Independently of the data insert rate, if the application needs to support more than 700 sensors, PostgreSQL has the best performance, the lowest energy consumption and the least use of system resources. For all the other cases, the choice of the database depends on the data collection parameters of the use case.

Our paper has focused on analysing database performance for SQL INSERT operations. In future work we will apply our methodology to investigate performance of these databases for SQL SELECT operations. Database technology can be divided into four categories: embedded, RDBMS, NoSQL and NewSQL, of which RDBMS is arguably the most widely used. The architecture of RDBMS offerings is rooted in the IBM System R database [36], which adopted a disk-centric design. NewSQL database systems, such as SAP HANA, have a variety of new internal architectures yet retain support for online transaction processing (OLTP) read-write workloads and maintain the ACID guarantees of a traditional database system. Our methodology can be employed to evaluate any RDBMS or NewSQL database. We aim to do this in future studies.

Acknowledgment

Funding: This work was supported by the European Commission under its Seventh Framework Programme, project Nanostreams [grant number 610509] and its Horizon 2020 Programme, project AllScale [grant number 671603].

References

[1] Georgakoudis, G., Gillan, C. J., Sayad, A., Spence, I., Faloon, R., & Nikolopoulos, D. S. (2016). Methods and metrics for fair server assessment under real-time financial workloads. *Concurrency and Computation Practice and Experience*, *28*(3), 916–928.

[2] Georgakoudis, G., Gillan, C., Sayad, A., Spence, I., Faloon, R., & Nikolopoulos, D. S. (2015). Iso-Quality of Service: Fairly ranking servers for real-time data analytics *Parallel Processing Letters*, *25*(3) doi:10.1142/S0129626415410042.

[3] Determann, R. M., Royakkers, A., Wolthuis, E. K., Vlaar, A. P., Choi, G., Paulus, F., Hofstra, J. J., De Graaff, M. J., Korevaar, J. C., & Schultz, M. J. (2010). Ventilation with lower tidal volumes as compared with conventional tidal volumes for patients without acute lung injury: a preventive randomized controlled trial. *Critical Care*, *14*(1):R1.

[4] Curley, G. F., Laffey, J. G., Zhang, H., & Slutsky, A. S. (2016). Biotrauma and Ventilator-Induced Lung Injury Clinical Implications. *CHEST*; *150*(5), 1109-1117.

[5] Bellani, G., Laffey, J. G., Pham, T., Fan, E., Brochard, L., & Esteban, A. (2016). Epidemiology, patterns of care, and mortality for patients with acute respiratory distress syndrome in intensive care units in 50 countries. *JAMA*, *315*(8), 788âĂŞ800.

[6] Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *Annals of Statistics*, *7*(1), 1–26.

[7] Efron, B. (1987). Better bootstrap confidence intervals. *Journal of the American Statistical Association*, *82*(397), 171–185.

[8] Efron, B., & Tibshirani, R. J. (1993). *An introduction to the bootstrap*. New York, NY: Chapman; Hall/CRC.

[9] Barak, N., Wall-Alonso, E., & Sitrin, M. D. (2002). Evaluation of stress factors and body weight adjustments currently used to estimate energy expenditure in hospitalized patients. *Journal of Parenteral and Enteral Nutrition*, *26*(4), 231-238.

[10] Dorj, U. O., Lee, M., Choi, J. Y., Lee, Y. K., & Jeong, G. (2017). The Intelligent Healthcare Data Management System Using Nanosensors. *Journal of Sensors*, vol. 2017, Article ID 7483075, 9 pages. https://doi.org/10.1155/2017/7483075.

[11] MariaDB. MariaDB versus MySQL - Features. Retrieved from https://mariadb.com/kb/en/library/mariadb-vs-mysql-features/.

[12] MariaDB. White Paper - Guide to Open Source Database Selection: MySQL vs. MariaDB. Retrieved from https://bit.ly/2vhYvWK.

[13] Januzaj, Y., Ajdari, J., & Selimi, B. (2015). DBMS as a Cloud Service: Advantages and Disadvantages. *Procedia - Social and Behavioural Sciences*, *195*, 1851-1859.

[14] Muntjir, M., Asadullah, M., Hassan, S., & Hussain, A. (2014). Cloud Database Systems: A model Conversion in Databases. In *2nd International Conference on Information Technology and Electronic Commerce (ICITEC 2014), December 20-21* (pp. 280-285), Sydney, Australia: University of Technology.

[15] Boncz, peter Alexander, Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications. Ph.D. Thesis. Universiteit van Amsterdam. May 2002. Available on-line at: http://oai.cwi.nl/oai/asset/14832/14832A.pdf

[16] DB-Engines. DB-Engines Ranking. Retrieved from https://db-engines.com/en/ranking.

[17] DB-Engines. Method of calculating the scores of the DB-Engines Ranking. Retrieved from https://bit.ly/2J0Rn2W.

[18] Andlinger, P., & Gelbmann, M. (2018, January). PostgreSQL is the DBMS of the Year 2017. Retrieved from https://bit.ly/2H4Jnxr.

[19] Gelbmann, M. (2018, April). MariaDB strengthens its position in the open source RDBMS market. Retrieved from https://bit.ly/2qCDPDs.

[20] Halili, E. H. (2008). *Apache jMeter: A practical beginner's guide to automated testing and performance measurement for your websites*. Birmingham, UK: Packt Publishing Limited.

[21] Devices, E. E. (2010) WattsUp PRO. Retrieved from http://www.wattsupmeters.com/

[22] Mooney, C. Z., & Duval, R. (1993). *Bootstrapping: A nonparametric approach to statistical inference*. London, UK: SAGE.

[23] Streukens, S., & Leroi-Werelds, S. (2016). Bootstrapping and pLS-sEM: A step-by-step guide to get more out of your bootstrap results. *European Management Journal*. doi://doi.org/10.1016/j.emj.2016.06.003,10.1016

[24] Wood, M. (2005). Bootstrapped confidence intervals as an approach to statistical inference. *Organizational Research Methods*, *8*(4), 454–470.

[25] Carpenter, J., & Bithell, J. (2000). Bootstrap confidence intervals: when, which, what? A practical guide for medical statisticians, *Statistics in Medicine*, *19*, 1141-64.

[26] Schmid, S., Galicz, E., & Reinhardt, W. (2015). WMS performance of selected sQL and noSQL databases. In *IEEE - International Conference on Military Technologies (ICMT)*, *May 19-21* (pp. 1–6). Brno, Czech Republic: University of Defense.

[27] Wang, Y., Zhong G., Kun L., Wang L., & Kai, H. (2015). The performance Survey of In Memory Database. In *IEEE 21st International Conference on Parallel and Distributed Systems*, *December 14-17* (pp. 815-821), doi: 10.1109/ICPADS.2015.109.

[28] Difallah, D. E., Pavlo, A., Curino, C., & Cudre-Mauroux, P. (2014). OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *Proceedings of the VLDB Endowment*, *7*(4), 277-288.

[29] Armstrong, T. G., Ponnekanti, V., Borthakur, D., & Callaghan, M. (2013). LinkBench: a database benchmark based on the Facebook social graph. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*, *June 22-27* (pp. 1185-1196), New York, New York, USA.

[30] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., & Sears, R. (2010). Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing (SoCC '10)*, *June 10-11* (pp. 143-154), Indianapolis, Indiana, USA.

[31] Gobel, A. (2014). MuTeBench: Turning OLTP-Bench into a Multi-Tenancy Database Benchmark Framework. In *CLOUD*

COMPUTING 2014 : The Fifth International Conference on Cloud Computing, GRIDs, and Virtualization, *May 25-29* (pp. 84-87), Venice, Italy.

[32] Aken, D. V., Difallah, D. E., Pavlo, A., Curino, C., & Cudre-Mauroux, P. (2015). BenchPress: Dynamic Workload Control in the OLTP-Bench Testbed. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*, *May 31 - June 04* (pp. 1069-1073), Melbourne, Australia.

[33] Bowerman, B. L., O'Connell, R. T., & Koehler, A. B. (2004). *Forecasting, time series and regression: An applied approach.* Belmont, CA: Thomson Brooks/Cole.

[34] Chipara, O., Chenyang, L., Bailey, T. C., & Roman, G. C. (2010). Reliable Clinical Monitoring Using Wireless Sensor Networks: Experiences in a Step-down Hospital Unit. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys '10)*, *November 03-05* (pp. 155-168), Zurich, Switzerland.

[35] Nedev, D. (2014, January). MonetDB sQL transaction management scheme. (MonetDB, Ed.). Retrieved from https://www.monetdb.org/blog/monetdb-sql-transaction-management-scheme

[36] Chamberlin, D. D., Astrahan, M. M., Blasgen, M. W., Gray, J. N., King, W. F., Lindsay, B. G., Lorie, R., Mehl, J. W., Price, T. G., Putzolu, F., Selinger, P. G., Schkolnick, M., Slutz, D. R., Traiger, I. L., Wade, B. W., & Yost, R. A. (1974). A History and Evaluation of System R. *Communications of the ACM*, *24*(10) 632-46.