Sentiment analysis and time series prediction on Twitter data

Alex Santini

MSc in Data Analytics

Integrated assignment big data processing and Advanced data analytics

CCT

28.10.2023

Sentiment analysis and time series prediction on Twitter data

Introduction

*SQL and NoSQL databases comparison*

*When it comes to storing SQL and NoSQL databases are the two main categories a developer needs to know for choosing the right tool for the right job; especially in the case of managing text data the two architectures have their strengths and weaknesses, In order to test those a testing strategies have been developed through the use of a familiar benchmarking tool called YCSB on a Linux virtual machine.*

*This comparison aims to explore the performances of a SQL database (Mysql) and a Nosql database (MongoDB) when taking a read-heavy and a insert-heavy load.*

*The databases in questions will have to contain the same tabular format to equally compare them, the format chosen is the same present in the excel sheet provided for the rest of this analysis in the csv file (projet\_tweets.csv) with the schema as follow:*

|  |  |  |
| --- | --- | --- |
| **Field** | **type** | **Example** |
| id | INT | 4587 |
| date | DATE | Sat May 16 23:58:44 UTC 2009 |
| flag | STRING | NO\_QUERY |
| user | STRING | bobthebuilder |
| text | STRING | Lyx is cool |

*Fig.1 YCSB comparison schema*

It’s important to notice how the schema must be explicitly mentioned prior to loading the data in the MySQL database whilethe same is not a requirement for MongoDB showing the first noticeable difference between the two.

The testing strategy aim to compare a classic insert heavy load and a more generalist *read heavy load with the following parameters:*

|  |  |
| --- | --- |
| Read Heavy |  |
| Count | 10,000 |
| Read | 80% |
| Update | 5% |
| Insert | 1% |
| Read modify write % | 2% |
| Scan | 3% |
| Distribution | Zipfan |

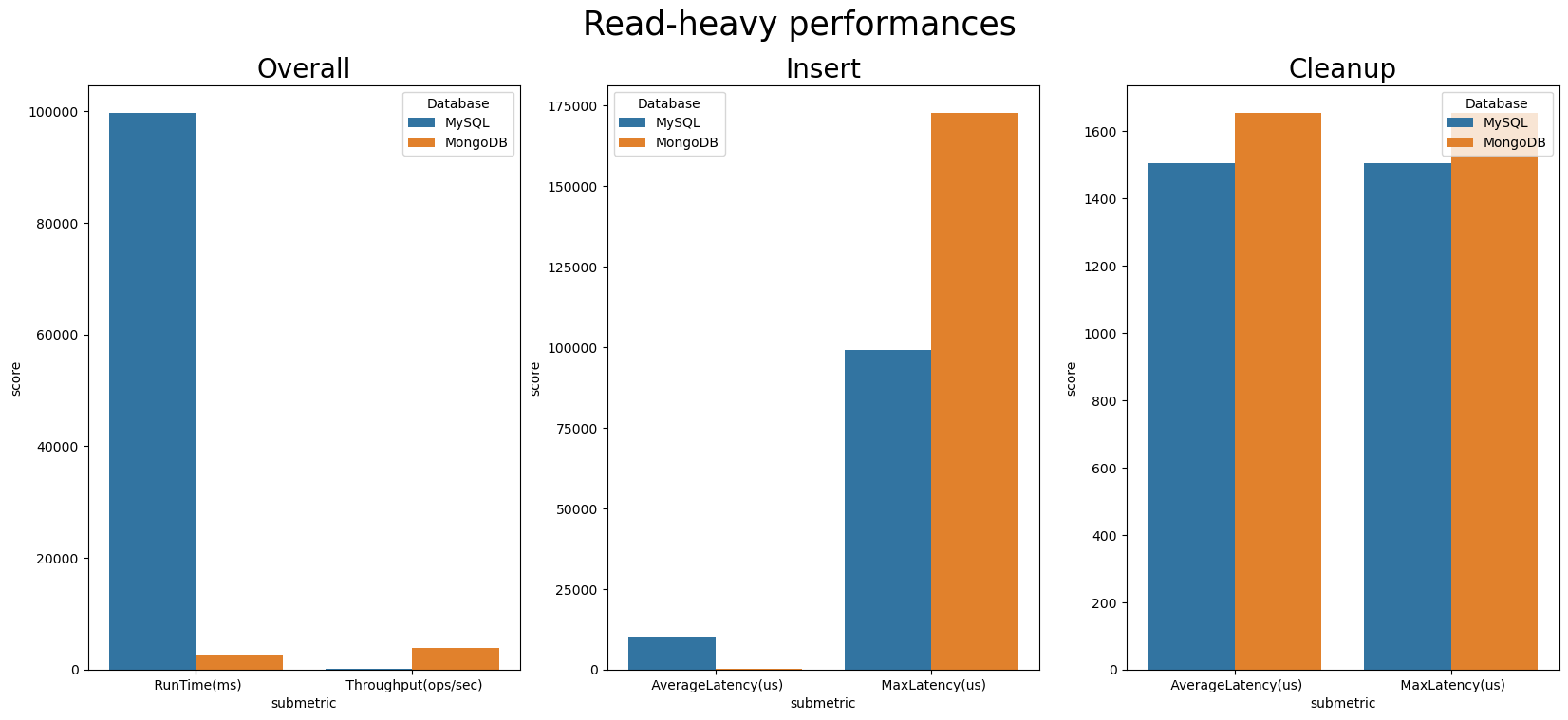
*Fig.2 Read heavy custom load parameters.*

In order to test the scalability of the two architectures a 80-20 insert-heavy load has been tested for different sizes as shown in Fig.3.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Count | 5,000 | 10,000 | 50,000 |
| Insert | 80% | 80% | 80% |
| Read | 20% | 20% | 20% |

*Fig.3 Insert heavy loads.*

By providing both read and insert comparison the test strategy aims to achieve a good comparison of SQL and NoSQL functions in a text heavy context.

As expected the results shown a clear difference between the performances of the two databases; the main metrics took in consideration for this approach were the overall runtime measured in milliseconds and throughput of the database measured in operations per second: this two metrics were chosen because considered good overall indicators of overall performances when it comes to choosing a database for a production pipeline, other metrics like maximum latency and average latency where taken also in consideration but as secondary point od reference*Fig.4 Read heavy load results*

The result for the read heavy workload are clear as MongoDB shows better performances running the load 20 times faster than the SQL counterpart, this is highly expected as being a document-based database MongoDB can handle large volume of read operations outputting a higher throughput than MySQL as shown in figure; MySQL however seems to perform slightly better in cleanup operations and in minimizing the maximum latency in insert jobs which in this case were only counting as 1% of the total workload.

Second step of the testing strategy was to compare three different workloads by increasing the number of total operations (5k,10k,50k) in order to test the scalability of an insert heavy architectures in SQL and NoSQL models.

The result once again proven how MongoDB is more robust in scalability especially in terms of runtime: MySQL seems to increate its runtime almost exponentially comparing a 5k workload with a 50k workload while MongoDB increasing just slightly its runtime, the same is true for the total throughput of the databases where MongoDB managed to increase it as needed for larger workloads while MySQL showing a performance plateau in fig.5; it’s also important to notice how even is increasing MongoDB performances seems to be affected by diminishing returns in throughput signaling how scalability might become an issue for NoSQL databases after a higher threshold.

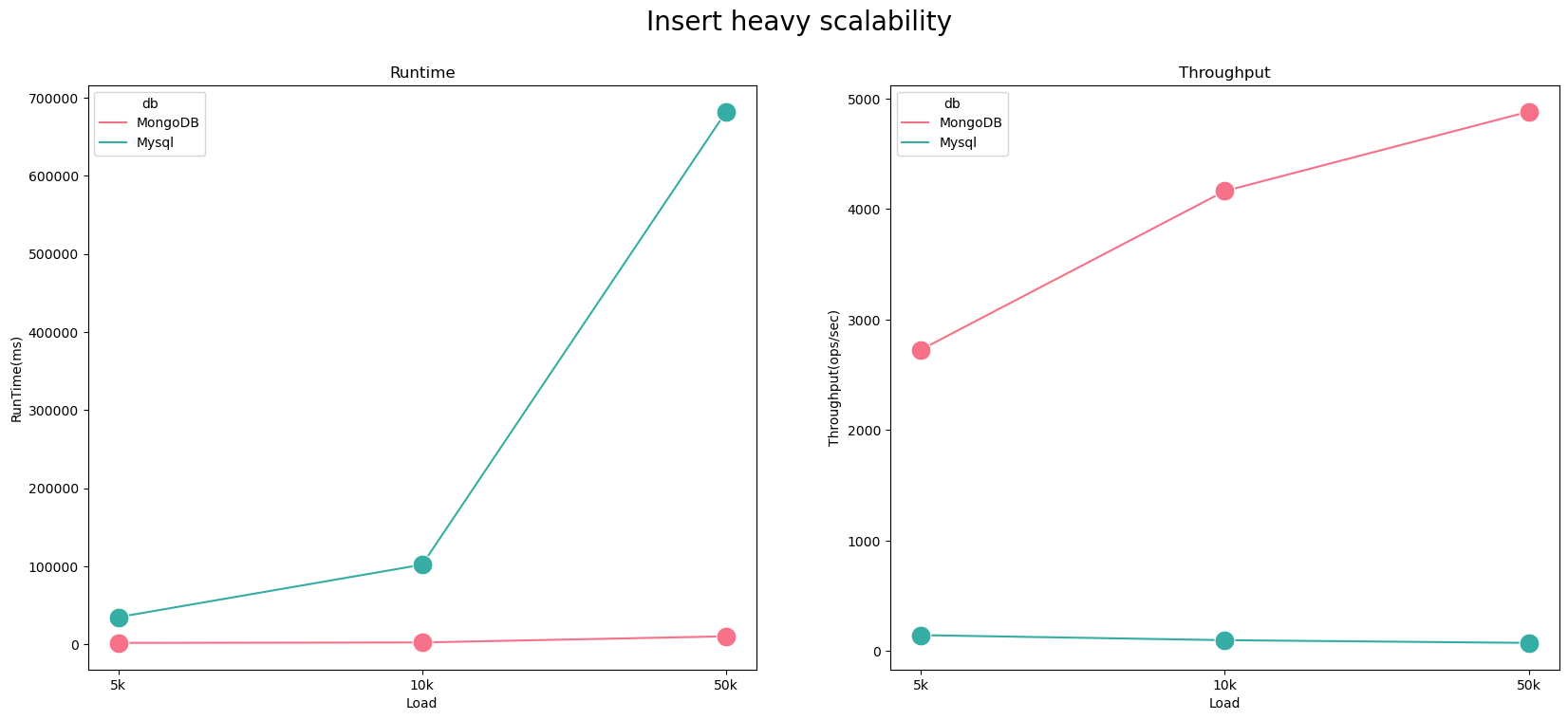


Fig.5 Insert heavy scalability load.

*Processing architecture*

After determining the performances of both SQL and NoSQL databases a conscious decision have been made to process the 1.5 million of data points required for the analysis; multiple databases and processing environments have been researched to come up with the best solution.

As per requirement the analysis needs two databases: one for storing raw data and another which can be used for extracting the final data for the time series analysis.

As result the raw data has been stored in a MySQL database: as shown in the first section MySQL is not the ideal choice for read operations, however raw data needs to be read only once, likewise the slow inserting speed can be surely limiting but again he data needs to be store only once not posing a big issue in terms of scalability and performances.

For storing post map-reduce data a Cassandra database has been deemed the most fitting as there might be a need for reading the data multiple times for the various steps in time series analysis, Cassandra is highly scalable and can produce high performances in heavy read tasks such as this one.

For processing environment Pyspark has been deemed the most idoneous especially because processing such large volume of data as Hadoop reduce job would have meant to not have access to the results of the data processing process in terminal, which for such large and variegated data can be problematic and difficult to handle.

Much focus in this research has been invested in finding the right tools in order to connect MySQL and Cassandra with the Pyspark environment: for connecting pyspark to MySQL it has been necessary to download a driver connector as jar file, while Cassandra connectors can be found online in the MVN repository for correct versions of Scala and Spark.

Main limitation for this approach can be found in the reliance of external connectors for the PySpark environment to read and to MySQL and Cassandra databases as different versions of spark and Scala might require different connectors.

As shown in figure raw data can be loaded into MySQL either directly through CLI commands or by using PySpark; second step of the architecture is to apply reduce processing and data processing for extracting the sentiment of each tweet and storing a daily average sentiment dataset into Cassandra where it can be extracted to a small csv file for proceeding with the time series analysis.

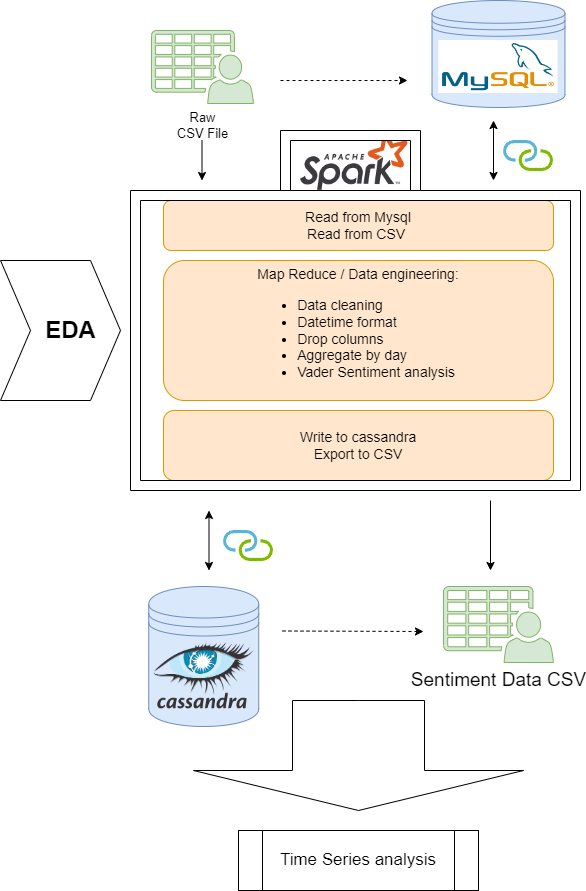


Fig.6 Processing architectures

Main advantages of this architecture are as follows:

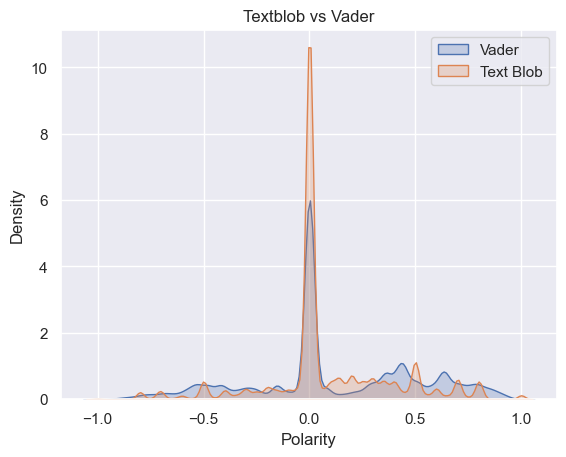
1. Ability to read and write to csv file directly: if needed it’s possible to load the csv file as PySpark dataframe directly and to extract a final dataset for sentiment analysis as csc file bypassing both MySQL and Cassandra.
2. Once uploaded through Pyspark, CSV files (raw and final) can be extracted from the MySQL and Cassandra into csv files.
3. More secure data processing: to get from a raw dataset to a daily sentiment average dataset knowledge from Exploratory data analysis has been applied to data engineering carried out in Pyspark reducing the work needed in terms of Pyspark jobs.
4. Having access to output in the terminal a more explainable output can be produced as Jupiter notebook compared to a reducer python file.

*EDA and Sentiment analysis*

Exploratory data analysis has been a key aspect of the whole project as the knowledge discovered during this phase acted as guidance for carrying out all sorts of transformations which were essentials to converting a raw dataset of tweets into a time series of aggregated sentiment values.

During EDA multiple sentiment extraction tools have been tested with the following metric in mind: which tool can give the least number of neutral scores on the whole dataset?

The tweets seem to be selected randomly in a three-month span, no specific tag or mention seems to appear which seems to link the tweets to a specific topic; having this in mind the main assumption is that a large chunk of them will return a neutral polarity score no matter the tool used, therefore the tool selected for extracting sentiment will be the one white the least neutral scores.

  
Fig.7 Vader vs Text Blob

The result in fig.7 shown as Vader is more efficient than TextBlob in capturing sentiment as not neutral, both algorithms have been applied after removing stop words and stemming; stemming appears to be the correct choice compared to lemmatization as the latter risks to derive the same lemma from two semantics opposite words possibly skewing the results.

The main problem encountered during time series analysis has been the huge amount of missing data, this project required to make predictions for up to three months, however the choice of granularity of the datasets was up to the researcher to establish; after extracting sentiment from each tweet the main question the researcher had to encounter was to group by the data by which time measurement: minutes? Hours? Weeks?

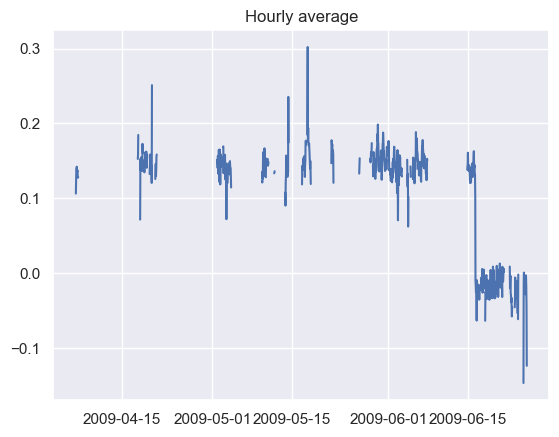


Fig.8 Hourly average raw dataset (before interpolation)

An ideal dataset would have an average dataset by the minute or even by the second, however missing data in the dataset used was massive and most of the choices were made to reduce the impact of missing data as the most requirement for time series is to have all datapoints recorded with regular intervals.

During EDA data has been grouped by seconds, minutes, hours and days in order to check which granularity would be the best in terms of limiting the amount of missing data.

A black screen with white text

Description automatically generated

Fig.9 Different granularities and missing data points

It was observed that, as expected, different granularities were bringing different percentages of null values; the choice made was to proceed with time series analysis by grouping the data by average day: this approach surely had its limitations as it means that at best the maximum datapoints would been not more than 90 (3 months of data) definitely too little to develop a successful model for any production environment but good enough for showcasing best practices applicable to real word scenarios.

Huge amount of work has gone into choosing the correct approach for inputting missing data both through experimentation on the data itself and through research (Petrusevich, 2021), multiple interpolation strategies have been tested and compared, comparison has been made by applying interpolation to the longest uninterrupted sequence of data by simulating a missing series of data with the same percentage of missing data as per the whole dataset (41%) and checking the result against true values; as shown in Fig.10 where the portion of data in red has been simulated as missing values inside of the longest interrupted sequence in yellow: the result of various interpolation strategies are shown in Fig.11.

A graph showing the amount of data

Description automatically generated

Fig.10 Interpolation test set

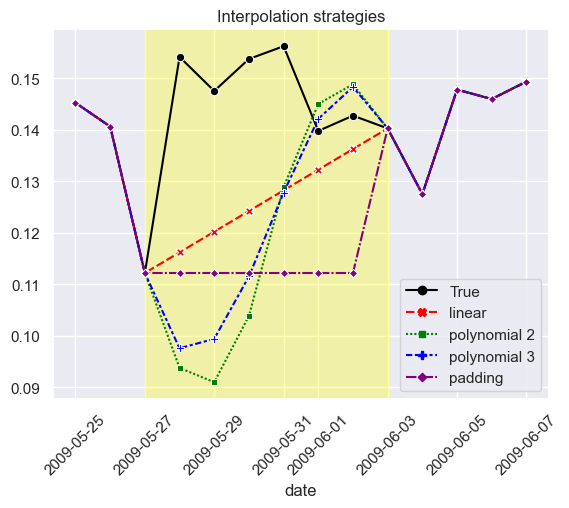


Fig. 11 comparison of multiple interpolation strategies against true values

Both Linear and Polynomial interpolations have been tested against true values: while linear strategies seem to find the closest route between two points, polynomial interpolations follow a sigmoid route for connecting the points.

All the interpolation strategies tested succeeded only partially, what if the answer to missing data might rely into exogenous features?

A deeper look revealed how different weekdays are correlated to different levels of polarity on average, an ANOVA test has been carried out and the difference in weekday average sentiment appears to be statistically significant.

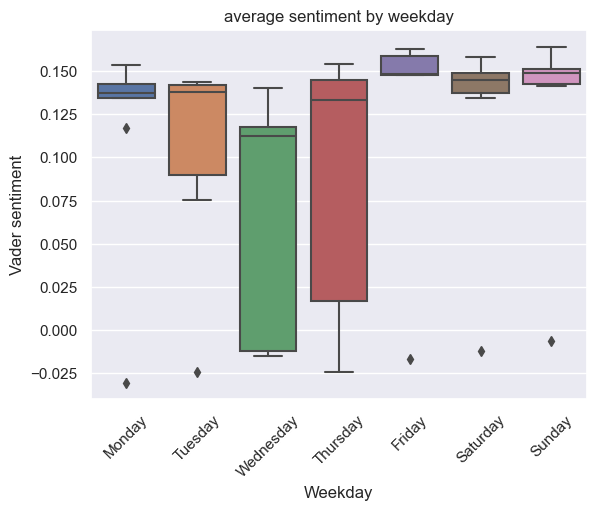


Fig.12 Average sentiment by weekday

Therefore a feature for weekdays has been employed as exogenous feature in predictions, on the interpolation side a 7 day seasonal average has been used to calculating missing values to reflect the difference in sentiment levels across different weekdays; clearly this approach contains both pros and cons as it returns a strong seasonal time series which should reflect the changing sentiment during the weeks, however it can also inputs noise under the assumption that the sentiment follows a 7 day seasonality.

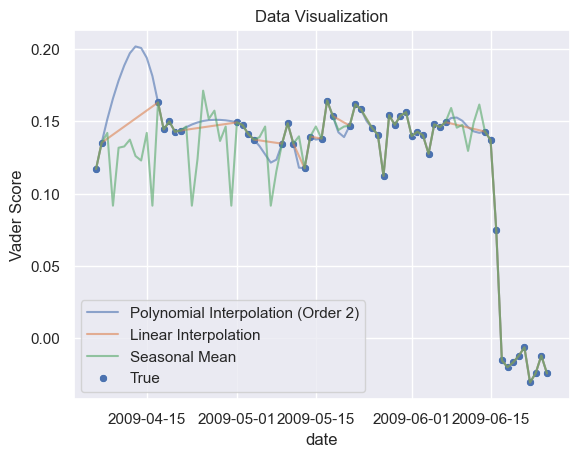


Fig. 13 result after 7 days seasonal imputation

*Time series prediction*

As per any time series first step has been to identify the trendiness of the series and to remove the trend from the data which will help massively in order to predict the sentiment averages three months from the last day recorded.

In terms of selected algorithm, an auto regressing random forest forecasted has been chosen because of multiple reasons; not only the innate ability to handle time series data but also as being an ensemble model it can generate robust models especially in situation with smaller datasets which can be tricky in terms of overfitting; random forest models are also robust against outliers, however the selected dataset did not present any as the sentiment score returns values between -1 and +1 therefore eliminating possible outliers by nature.

Sources

Caruso, F. & Quarta, (1998) ‘Interpolation methods comparison’, Computers & Mathematics with Applications, 35(12), pp. 109-126.

Lepot et al. (2017) ‘Interpolation in Time Series: An Introductive Overview of Existing Methods, Their Performance Criteria and Uncertainty Assessment’ *Water*, *9*, pp. 796.

Petrusevich, D. A. (2021) 'Title of the Article', Journal of Physics: Conference Series, 1889(3), pp. 032009.C.