A Microservices Persistence Technique for Cloud-Based Online Social Data Analysis

1 Feras Al-Obeidat, 1 Anoud Bani-Hani, 2 Oluwasegun Adedugbe, 1 Munir Majdalawieh, and 2 Elhadj Benkhelifa

**Abstract**

Social data analysis has become a vital tool for businesses and organisations for mining data from social media and analysing for diverse purposes such as customer opinion mining, pattern recognition and predictive analytics. However, the high level of volatility for social data means application updates due to analytical results requires spontaneous integration. In addition, while cloud computing has been hugely utilised to address computational overhead issues due to the volume of social data, results obtained still fall short of expected levels. Hence, a persistence mechanism for rapid deployment and integration of software updates for the analytical process is proposed. The persistence mechanism constitutes a significant component within a novel methodology which also leverages cloud computing, microservices and orchestration for online social data analysis, one which fully maximises cloud capabilities and fosters optimisation of cloud computing resources. The proposed methodology provides means of delivering real-time, persistent social data analytics as a cloud service, thereby facilitating spontaneous integration of solutions to maximise expectations from targeted social media audience.

**Keywords** Social Data Analysis; Persistent Social Data; Social Networks; Persistent Microservices; Cloud Orchestration; . Cloud Computing

**1 Introduction**

Social data analysis aims at employing diverse methods for analysing data on social media through several processes such as data discovery, collection, preparation and analysis. Businesses and organisations utilise social media for various purposes including detecting new communication trends, as a communication channel for client relationship management, product placement on the social web and to support decision-making processes through reports and pre-defined key performance indicators [1]. It is important to understand challenges of executing social data analysis in order to manage complexities in its implementation. Data volume on social media constitutes a huge challenge, especially in the discovery phase.

Feras Al-Obeidat

feras.al-obeidat@zu.ac.ae

Anoud Bani Hani

anoud.bani-hani@zu.ac.ae

Oluwasegun Adedugbe

Oluwasegun.Adedugbe1@staffs.ac.uk

Munir Majdalawieh

munir.majdalawieh@zu.ac.ae

Elhadj Benkhelifa

 e.benkhelifa@staffs.ac.uk

1 College of Technological Innovation, Zayed University, United Arab Emirates.

2 School of Digital Technologies & Arts, Staffordshire University, Stoke-on-Trent, United Kingdom.

It is important to filter out irrelevant data using appropriate techniques such as advanced topic detection algorithms. Data variety is another major challenge faced in the later stages [2]. The highly dynamic nature of social media adds a significant level of complexity to data collection and preparation for analysis. Technologies for analysing these data deployed in the cloud provide a significant leap in performance levels. However, these can be observed to still be hampered by application deployment patterns. The amount and speed of data on social media requires the most ideal software architecture for data collection, processing, storage and analysis. With traditional environments where data was minimal, a single, monolithic application would process all phases of the data analysis task, resulting in a time-consuming activity as well as constituting a single point of failure. Such solutions are inadequate for the enormous social data available today. The microservices software architecture is an evolving one which provides capabilities for addressing such challenges. The concept of microservices requires splitting applications into smaller services such that each can be tested, scaled, implemented, monitored and deployed independently [3]. Such a deconstruction allows all functionality components to be deployed or updated without affecting other components of the application [4]. It presents an approach for software service design, delivery and development that focuses software application’s development processes on well-established modularisation concepts and emphasises on technical boundaries [5].

With the need for timely social data analysis in order to facilitate responses to sentiments and opinions mined from such data, microservices provide faster ‘time to market’ for applications and software products by speeding up processing time as separate development teams can work on different microservices of the same application in parallel [6]. In addition, Dragoni et al. [7] opines that most organisations today use microservices to become more agile in responding to market changes. With the ability to leverage both hypervisor-level and operating system-level virtualisation for applications and deploy them on a cloud platform, supported by a software integration mechanism that is fully automated, the time-consuming and high computational overhead with social data analysis can be addressed. Table 1 provides a comparison between microservices architecture and the more prevalent monolithic architectures commonly deployed for social data analysis.

Table 1. Comparison of monolithic and microservices software architectures [8]

|  |  |  |
| --- | --- | --- |
| **Factor** | **Monolithic Architecture** | **Microservices Architecture** |
| **Code** | Applications are developed as a unit, hence, a single code base for an entire social data analysis application.  | Since there are multiple small services that make up a large application, social data analysis phases can have multiple code bases, each as a microservice. |
| **Understandability**  | As code bases grow over time, it becomes more difficult to understand the entire process and make updates. Maintenance also becomes difficult, due to size and complexity. | This is more understandable as it is micro in nature. It becomes easy to investigate each social data analysis microservice to obtain any information especially while troubleshooting. Maintenance is relatively easier as well. |
| **Reusability**  | Due to size and complexity, it is not easy to reuse the application code base.  | Any part of the entire application can be reused since they are independent in nature.  |
| **Deployment**  | Any update means redeployment of the entire social data analysis application. Continuous deployment becomes difficult. It is complex to deploy within restricted maintenance windows and scheduled downtimes. | It is easier to re-deploy each social data analysis microservice independently, with minimal or zero downtime. This promotes the possibility of continuous deployment for complex applications.  |
| **Reliability**  | When there is an issue with a module, it has the potential to bring down the entire application. Furthermore, troubleshooting could be very cumbersome and time-consuming. | The level of reliability is higher since each unit is on its own. It is easier to troubleshoot a microservice independently without any impact on the others within the application. |
| **Adaptability** | The adoption of emerging technologies for the domain is usually very difficult as transforming from a technology stack to another would probably imply re-developing the entire application from scratch. This will be quite expensive in both time and cost. | Emerging technologies within social data analysis are easier to adopt for specific microservices and with polyglot programming, microservices can run based on different technology stacks, giving developers the opportunity to adopt innovations quicker. |
| **Testing**  | Testing is easier and can be through end-to-end testing by launching the application.  | There is a level of complexity in the process of testing this architecture due to the multiple microservices available. |
| **Technology**  | The entire development is typically based on a specific technology stack. | Each microservice can be based on a separate technology stack from the others. |
| **Scaling** | The entire application needs to be scaled since it’s a single unit. While horizontal scaling is simple when multiple instances run behind a load balancer, this implies a very high computational overhead and cost.  | The entire application does not need to be scaled. Scaling can be automated for each microservice based on pre-defined thresholds, hence optimising computational overhead and cost.  |

Hence, this research is motivated by the potentials for very minimal “time-to-market” of software updates that are based on analysis of data extracted from social media for purposes such as sentiment analysis and opinion mining. The remaining sections of this paper are as follows: section 2 provides critical review of related work, both from the perspective of microservices architecture-based solutions and cloud-based deployments for social data analysis. Section 3 analyses gaps from the investigation and proposes design methodology which includes a novel technique for persistence in integration and deployment of updates for the analytical process. The methodology also includes other features such as application orchestration and asynchronous messaging for fully maximising cloud computing capabilities. Furthermore, it builds on the persistence process flow by developing a framework for microservices-based, persistent social data analysis using cloud computing. The section also includes an analysis of major components within the framework and concludes with its potential impact on the subject matter. A summary for the research effort is provided in section 4, as well as recommendations for further research in the domain.

**2 Background and Related Work**

There is the need for scalable solutions characterised by high performance to process and analyse social data. Cloud computing facilitates implementation of elastic services with scalable performance [9]. The cloud contains data repositories for applications, including social media, thereby enhancing analysis of large datasets with low latency for effective analytical purposes such as reporting and decision-making models. The growth of service-oriented computing is also increasing utilisation of cloud-based systems for scalable social data analysis [10]. Therefore, cloud computing enhances data analysis tasks and applications as services at software, platform and infrastructure levels, based on diverse technological solutions and deployment patterns. Alongside the three levels, other variants exist which are regarded as hybrid; combining two of the default three to deliver cloud solutions. Likewise, all three can be integrated for provisioning to the end-user, in an Everything-as-a-Service (XaaS) model [11]. This stack can deliver data analysis-as-a-service (DAaaS), using the model to support structured development of data analysis systems, tools and applications based on a service-oriented approach [10].

While cloud physical infrastructure differ from one vendor to another, leveraging IaaS for social data analysis is widely available. Likewise, PaaS is leveraged for same purpose. However, two main categories of PaaS can be defined; default and enhanced. A default platform provides the operating system and other software, or packages required to run a solution. Such packages for social data analysis include Hadoop MapReduce and Apache Spark. On the other hand, an enhanced platform requires implementation of a platform-based solution for automating tasks at the application layer of a solution, utilising concepts such as orchestration or choreography. Such enhanced platforms provide a higher level of automation and management capabilities not available with a default platform.

For leveraging default cloud platforms, frameworks such as Google Cloud Dataflow, Apache Spark, Hadoop MapReduce and Hadoop++ are commonly utilised [12]. These frameworks mainly provide storage and processing capabilities for data. Hadoop is among the most renowned frameworks in this context, implementing MapReduce and facilitating distributed processing of large, heterogenous datasets [11]. Furthermore, it can be deployed on both public and private clouds. With public clouds, providers such as Hortonworks, Cloudera and BigInsights provide Hadoop distributions deployable and operated on Rackspace, Microsoft Azure and Amazon Web Services cloud platforms, leveraging IaaS and default cloud PaaS. Cloud providers such as IBM’s SmartCloud offer same on private clouds for deployment of InfoSphere BigInsights and IBM’s PureData System; providing PaaS solutions characterised by a pre-built setup for convenient and easy deployment of Hadoop [11]. Some other frameworks based on different methodologies have also been proposed.

Marozzo et al. [13] used the DASaaS methodology to develop a cloud-based system called the Data Mining Cloud Framework (DMCF). DMCF caters to three major classes of data analysis and knowledge discovery applications: single-task, parameter-sweeping, and workflow-based applications. Single-task applications allow the performance of a single data mining task such as classification, clustering, or association rules discovery on a dataset. Parameter-sweeping applications on the other hand, foster analysis of a dataset by several instances of the same data mining algorithm using various parameters. Workflow-based applications aid in specifying knowledge discovery applications as graphs that link together data sources, data mining tools, and data mining models. Programming of visual workflows is also possible in DMCF via VL4Cloud (Visual Language for Cloud) [10].

Similarly, other cloud-based systems for data analysis applications include Apache Spark, Sphere, Swift, Mahout, and CloudFlows with most of them as open source software. Apache Spark is for in-memory data analysis and machine learning [14]. It runs both batch processing and dynamic applications such as streaming, interactive queries, and graph analysis. Spark also offers a programming interface based on a data structure known as the resilient distributed dataset (RDD). It is different from Hadoop as well as other systems due to storage of data in memory and repeated querying for improved performance [10]. Apache Swift on the other hand, is a framework based on workflows to implement functional data-driven task parallelism for data-intensive applications.

Furthermore, issues with processing large amounts of social data was studied by Mathur & Purohit [15], suggesting the utilisation of comprehensive parallel processing and new analytics algorithms to effectively process exabytes of data in providing timely and actionable information. Some other research efforts within the domain include architectural frameworks and workflows for processing social media data [16], as well as techniques for large-scale integrated social data access [17] [18]. However, a holistic and multi-dimensional solution for dynamic, continuous and analytics-based software updates is required for a highly volatile and data-intensive domain such as social media.

**2.1 Microservices-Based Social Data Analysis**

The use of microservices for social data analysis can be observed to be having significant impacts. The work by Khaleq & Ra [19] focused on a disaster management system for data analytics, capable of analysing hurricane data on Twitter. It utilised microservices, hosted and managed on Azure cloud platform to pre-process, stream, and implement predictive analysis for disaster management. The developed prototype demonstrated that microservices approach facilitates a distributed, cloud-based system architecture that is also dynamic, reliable and scalable. The flexibility of the system architecture ensures addition, update or removal of independent processes had little or no impact on the entire system. Other frameworks proposed by [20] and [21] also foster application modularity towards large-scale data analytics with robust system architectures for updating independent processes.

In addition, the enhanced analytic approach proposed by Ali et al. [22] focused on a data processing mechanism based on machine learning and semantic web technologies, embedding analytic capabilities in modular microservices for services. It was characterised by efficiency and scalability in supporting adaptive IoT applications. Each microservice focused on encoding a special analytic functionality and display interface through an API. The modification and independent replacement of these microservices are possible in supporting robust and stable IoT applications with enhanced scalability. The work by Hsu & Lin [23] was also similar; implementing a social data analytics platform built on microservices architecture over Data Center Operating System (DC/OS). It involved the use of several frameworks and technologies such as Apache Spark, Apache Kafka, Apache Mesos, Docker, MongoDB, and Node.js. Its streaming processing provided a visual interface to display the hottest hashtags from an online forum while batch processing displayed various statistics such as most liked or commented posts. The use of docker containers over Apache Mesos implied the application could achieve dynamic scalability while Kafka and Spark streaming were utilised in analysing popular hashtags from user posts in real-time.

In addition, the study of Khoonsari et al. [24] utilised microservices for scientific workflows executed in parallel using the Kubernetes container orchestrator. The study focused on integrating key software suites for turn-key workflow, encompassing pre-processing, multivariate statistics, and metabolite identification in mass-spectrometry-based metabolomics. Fernández García et al. [25] as well developed an architecture based on microservices for generating datasets which contained user behaviour for further analysis, with each microservice fetching its own data, resulting in consistent performance for optimal transformation of datasets using feature engineering strategies. Communication channel for the microservices was based on REST API web service.

Furthermore, Dinh-Tuan et al [26] proposed microservices-based architecture for industrial data analytics by applying domain-driven design and the bounded context concept for definition of microservices with little inter-services dependency. Asynchronous messaging was used as the major mechanism to facilitate the inter-service interaction, thereby decoupling services. A high level of autonomy and independence between microservices was also enhanced with the database-per-service pattern. Each microservice was deployed as a containerised unit and this enabled addition of another layer to incorporate third-party solutions for visualisation, monitoring, and management. The architecture helps to minimise data dependency among microservices through provision of personal data to each service. Hoque & Miranskyy [27] developed a scalable and maintainable architecture to perform analytics on streaming data, proposing a 7-layered architecture consisting of microservices as well as instances of publish-subscribe pattern (an asynchronous communication hub), spread over multiple layers. The architecture obtains heterogeneous streaming data from a variety of sources as input for identification of a subset of data that suit different use cases, towards analysis of the data to satisfy the use cases.

The study of Innerbichler et al. [28] proposed NIMBLE Collaborative Platform which utilised microservices architecture to process data. The platform architecture helps to monitor, optimise, and negotiate in manufacturing supply chains, based on IoT and in real-time. The microservices within the architecture were implemented using either REST (HTTP) or messaging for service communication. The study of Ciavotta et al. [29] proposed a simulation-based architecture platform which implements a microservice IoT-Big Data architecture to support the publication of multidisciplinary simulation models and to manage streams of data emanating from the shop-floor for real-digital synchronisation. The researchers applied microservices architecture to support their infrastructure for the management of Digital Twins (DTs).

From the diverse related works, the significance of adopting microservices architecture for social data analysis can be observed. Microservice architectures allow for separation of various services that facilitate isolation of updates, deployments and management, thereby allowing for innovation and experimentation of services. Microservice architectures also support continuous upgrading to ensure improved user experience. In addition, encapsulation of modularised application logic functionalities in software containers (otherwise referred to as containerisation) for operating system-level virtualisation using container software such as Dockers fosters microservices’ capabilities. These are configured for hypervisor-level virtualisation as well. Furthermore, a compute method such as orchestration for automating deployment, scaling, monitoring and management of containerised functionalities using software such as Kubernetes, Docker Swarm or Amazon Elastic Container Service provides a deployment platform for the containerised microservices. Application abstraction and inter-process communication are also features noted with microservices-based data analytics. These diverse technological solutions and their roles in social data analysis is summarised in Table 2.

Table 2. Technologies for microservices-based social data analysis

|  |  |
| --- | --- |
| **Feature** | **Role** |
| **Software Architecture**Technology: Microservices | Higher level of decomposability of software architectural components fosters software agility by being lightweight and requiring lesser computing resources to meet user demands. It also greatly enhances automatic scaling. |
| **Application Abstraction**Technology: API Gateway | Abstraction of shared functions for the operation of various applications, acting as a reverse proxy to various microservices, and operating as a single-entry point into the microservice application system. |
| **Containerisation**Technology: Docker Containers | Containerisation greatly enables microservices by encapsulating them with the required files and binaries needed for them to be independently deployable. |
| **Communication Channel**Technology: Apache Kafka | Inter-process communication mechanism based on asynchronous messaging system for data exchange and sharing between the microservices. |
| **Orchestration**Technologies: Apache Mesos, Kubernetes | Provision of a platform for deploying containerised apps; automating their deployment, scaling, monitoring and management which subsequently enhances software agility. |

While the suite of technologies defined in Table 2 enhance social data analysis, the integration of a persistent mechanism for deploying updates to application logic and executing tasks for social data analysis phases would further enhance productivity. This is based on the high level of volatility for social data, requiring very quick and multiple software updates in response to user sentiments and opinions mined from social data [30]. Hence, a dedicated pipeline for deploying and integrating such updates increases agility for data analysis solutions. Generally, some of the tasks involved with the persistent mechanism include rapid initiation of code builds, testing procedures, and deployments, which can be as often as required based on feedback from previous social data analysis instances that triggered software updates.

**3 Persistent Mechanism for Social Data Analysis**

The concept of persistent mechanism for social data analysis defines an application automation lifecycle for continuous integration and delivery of software updates and transactions processing. While the focus is on persistence for software, it applies to hardware as well. This is by means of automation for rapidly provisioning cloud infrastructure using document templates containing machine-readable configuration data. This is as opposed to physically configuring hardware or using graphical configuration tools that require human intervention and subsequently, prone to errors. The template files can be developed using data portable formats such as YAML (Yet Another Markup Language) or JSON (JavaScript Object Notation) and defines diverse rules for deploying updates. These include rules for establishing communication between containerised microservices, mounting storage volumes, locating containerised images, and storing container logs [31]. Based on these rules, updated containers are deployed onto host machines typically in groups that are replicated. Container Orchestration Engines facilitate this by an integration with third-party solutions.

The persistent mechanism process flow also involves merging several sets of local changes to a shared code repository, allowing multiple developers define respective services that will be used during application development. The process flow also permits different developers to develop and update different services concurrently. These are then merged to the source code repository. The source code repository is a storage location for hosting code developed when updating the microservices. Predefined configuration files in the source code repository are used to create new microservice images after updates have been pushed to the repository. For example, with docker containers, the YAML “docker-compose.yml” configuration file instructs tools present within the orchestration environment to create or update an image of the containerised microservice, establish networks for communication and a storage location for storing log files generated during orchestration [32]. The new microservice image is then tested to identify any errors. Once the microservice image passes the test, source codes are combined with their respective dependencies to create a new instance. The new instance is a runnable file of the containerised application being updated for re-deployment. Figure 1 presents process flow for the persistence mechanism.



**Fig. 1** Persistent mechanism process flow for social data analysis

Like the microservice image, the health of the new instance must be tested. Testing identifies problems or errors with the application functionality or with the integration process. These tests are automated, based on a test plan to validate the software’s behaviour and functionality. The time used for testing the health of an instance depends on the its size, complexity and scope of the testing process. A healthy instance is one whose code is bug-free, and the developed functions work as intended. In some cases, the test results may show an unhealthy instance in which case, the new instance would be discarded, and the previous instance is left to continue running while identified bugs and errors in the new instance are fixed. Once they are fixed, the new instance is restarted, and it is subjected to the same testing process. The need for testing a new instance until it is 100% healthy is to ensure that reproducible errors do not reach targeted end users. With the large scale for social data analysis, tests are required to be run using several stages. These can start with a smoke test, which is designed to perform sanity tests for end to end integration and from a user’s perspective [33]. Testing the health of new instances expose potential issues not identified during development phase. It is also vital for the process to provide quick feedback in order to maintain the stream. Considering integration of the persistent mechanism with other technologies defined for social data analysis in Table 1, an overall framework based on these is defined in Figure 2.



**Fig. 2** Persistent microservices and cloud-based social data analysis framework

**3.1 API Gateway**

The API gateway stands as an interface between users and the social data analysis application, fostering system security, throttling, caching, and monitoring. It processes heterogeneous needs of various end users, routing client requests and discovering when a microservice is not running. Without the use of this layer, communication through microservices would face various complex challenges [34]. Microservice applications run on various user programs and communicate with architecture layers through various devices. Hence, with the API gateway, service requests can be channelled according based on the analysis task, such as data collection, storage or processing. Most microservice-based architectures rely on API gateways to facilitate problem solution from different microservices. The API gateway also provides adequate management of the complex, and numerous APIs within the application logic layer. Therefore, for social data analysis microservices architecture, it is crucial to include API gateway to act as the only entrance to all the microservices in the entire application architecture, preventing direct access to the microservices, hence fostering security and enhancing structure within the distributed system. Figure 3 illustrates interaction of an API gateway with different microservices for social data analysis.



**Fig. 3** API Gateway for social data analysis microservices

**3.2 Service Discovery**

Service discovery is based on a registry that ensures requests via the API gateway is channelled to the appropriate microservice. This is implemented by keeping a database of the different microservices running within the application and their addresses. Hence, once the API gateway transfers a request to the service discovery, the microservice required for processing it is identified and the request is forwarded to its address. Such requests can be for any of the phases of a social data analysis task. This ensures that the entire application layer is not running when only a single service is required, thereby optimising computing resources. Also, the service discovery ascertains that microservices applications process requests effectively such that it can handle workload changes. Hence, there is a direct communication channel between the API gateway and the service discovery, which in turn, transfers requests to microservices. Its implementation can either be from the client-side, whereby a client determines the location of networks hosting available analytical services and their load balancing requests. The client queries the service registry, a load balancer is used to select the appropriate service instance and client makes a request of the service instance. On the other hand, server-side discovery requires utilisation of load balancers by the client to query for availability of service instances in the service registry, with each request routed wherever a service instance is available. For both solutions, the service registry ‘registers’ and ‘deregisters’ service instances.

**3.3 Containerisation**

Virtualisation at the operating system level, also known as containerisation or container-based virtualisation is lightweight compared to hypervisor-based virtualisation. In this case, host operating system level virtualisation is performed [35]. The host OS (Operating System) kernel is shared by all virtualised instances since there are no hypervisors involved. This largely reduces the runtime overhead and sharing the same operating system reduces storage overhead [36]. The virtualisation layer is positioned between the operating system and application programs running on it. Sets of applications or software written for an OS being virtualised is run by the virtual machine [37]. For social data analysis, each phase of the task is developed as a microservice and runs in what is known as a container, alongside required libraries and binaries for its execution. Containers provide a level of abstraction on top of host operating system kernel, allowing each instance of a container to behave as an independent system in isolation. Figure 4 illustrates the use of container-based virtualisation, as applied for the framework in Figure 2.



**Fig. 4** Container-based virtualisation for social data analysis

It is crucial to note that the infrastructure referred to in Figure 1 is still a virtual one, as it leverages hypervisor-based virtualisation to run, such as is the case with EC2 instances on Amazon AWS cloud. Hence, the social data analysis microservices leverage both hypervisor-based and operating system-based virtualisation technologies as part of a technology stack to provide more efficient results and optimisation of computing resources utilisation.

**3.4 Messaging Channel**

With each microservice of the social data analysis framework having its own data store, a messaging channel is required to facilitate data exchange between them. With diverse protocols available for data communication among microservices of an application, the publish-subscribe pattern is well suited for social data analysis based on its design for handling big data using asynchronous message passing. Its architecture facilitates a high level of throughput and performance for applications. Data from individual microservices available for sharing is defined as a topic by the pub/sub messaging pattern. These are published to the messaging channel and available to consume by other microservices subscribed to the data topic. Hence, this implies that microservices publish (as a publisher) their data to the messaging channel and other microservices subscribe to consuming such data (as a consumer). Figure 5 illustrates the basic architecture of the messaging pattern.



**Fig. 5** Apache Kafka pub/sub messaging architecture for social data analysis

While several messaging systems exist, Apache Kafka has very high performance levels to meet the computational cost for social data analytics. A single Apache Kafka server (broker) can handle many operations (writes and reads per second) from many customers. It is also preferred because its message delivery system is durable, and messages are ordered. Kafka can effectively handle many benchmarks compared to RabbitMQ [38]. Besides, it is highly scalable, a major requirement for social data analysis. Messages related to a topic are dispensed among partitions. The topics are scalable because they can be divided into several partitions. This ability to divide topics into several partitions enables Kafka to offer both load balancing and ordering guarantees over clients’ processes. Kafka also scales better due to consumer groups’ concept. Furthermore, it is a distributed messaging system making it a preferred option to other messaging systems. It operates as a collection consisting of one or many servers commonly referred to as a broker. The modern design of Kafka provides strong fault tolerance and durability guarantees.

**3.5 Orchestration**

Orchestration as a compute method provides automation for configuring, co-ordinating and managing a collection of computing systems and software harmoniously. In this context, computing systems and software refer to the application logic layer of the framework in Figure 2, comprising of the API gateway, service discovery and the microservices. As such, the containerised microservices can be managed as an entity: aiding their availability, networking and scaling while simplifying overall management. Some key functions of an orchestration platform include cluster state management and scheduling, high availability and fault tolerance, security, service discovery and monitoring [11]. Furthermore, orchestration utilises configuration files, usually in JSON or YAML format to execute processes for applications. These files are machine-readable and defines rules for the successful running of each containerised microservice. These includes rules for establishing communication between microservices, mounting storage volumes, locating container images and storing container logs [31]. Figure 6 presents an illustration for cloud orchestration applicable to social data analysis, using a software such as Kubernetes.



**Fig. 6** Cloud orchestration architecture for social data analysis, using Kubernetes

Based on the rules, containers are deployed onto worker nodes typically in groups that are replicated. With Kubernetes software, each worker node comprises of a kubelet, which monitors and manages activities within the node. It also comprises of a kube proxy for keeping a record of all running pods and their locations. When a service request has multiple pods running, the kube proxy performs load balancing to distribute requests among running pods. The pods define instances of each containerised microservice running within the worker node. On further request to deploy a new container into the cluster, the container orchestration platform schedules deployment by an assessment of the most suitable worker node based on computing resources availability and needs of the container [27]. Upon successful deployment, containers lifecycles are managed based on configuration data. This management usually includes a self-healing mechanism in the event of application or data corruption. The self-healing mechanism is facilitated by an automatic re-start or re-scheduling of affected containers. This also applies within the persistence process flow during deployment and integration of software updates, when required. With social data size and complexity, an automated platform for the different phases addresses some of the challenges relating to computational overhead, with computing resources effectively managed and elastically provisioned in a fully automated manner.

**4 Conclusion**

A persistent mechanism for facilitating microservices-based social data analytics in the cloud was the focus for this research. Beyond default application deployment in the cloud, a methodology leveraging a suite of technologies was defined, with significant considerations for the software architecture, virtualisation, messaging and compute patterns. This is alongside the persistent mechanism for ensuring an automated stream to deploy and integrate software updates for social data analytics based on the highly volatile nature of its data. The persistence mechanism approach fosters several stateless engines for concurrently aggregating, inferring and processing social data, thereby allowing stakeholders such as data scientists utilise dashboards and client applications to access same raw or real-time data derivatives with appropriate data masking and versioning. Hence, the abridged delivery cycles for online social data analysis can be embraced as they automate across delivery pipelines.

Furthermore, it allows the abstraction of processes such that updates can be pushed to the application layer in a continuous release manner and deployed to sets of clusters across cloud environments. This is also applicable for multi-cloud solutions, thereby enhancing software agility and facilitating minimal ‘time-to-market’ for industry solutions. This is further facilitated with the microservices architecture which implies the updates and release of code changes are to individual components (microservices) of the analytics stream, thereby ensuring minimal statistical and analytical errors. With a monitoring facility, vital data such as logs for system health and performance are also gathered to feed back into the development process for necessary changes and released back into the stream.

The developed framework based on this methodology implies that analysis of social media data can experience a higher level of agility and efficiency, while still significantly minimising computational overhead. These also promises to foster decision-making and implementation for businesses and organisations based on interpretation of analytical results. While the implementation can pose some challenges due to differences across data from multiple social media platforms, a data processing pipeline to aggregate and structure data based on a defined schema would address the challenge. This can be facilitated using ETL (Extract Transform Load) solutions. Recommended further research in the area would include investigation into the use of other compute patterns, such as choreography in comparison with orchestration. This can be facilitated with serverless architectural solutions provided by cloud vendors such as Amazon AWS, Google and Microsoft Azure Cloud Platforms. A comparative analysis between implementation results for both patterns would also be beneficial towards maximising results from analysis of large-scale social media data.

**References**

1. Sebei, H., Taieb, M.A.H., Aouicha, M.B.: Review of social media analytics process and big data pipeline. Social Network Analysis and Mining, 8(1), pp.30-31 (2018)
2. Horrocks, I., Giese, M., Kharlamov, E., Waaler, A.: Using semantic technology to tame the data variety challenge. IEEE Internet Computing, 20(6), pp.62-66 (2016)
3. Newman, S.: Building Microservices: Designing fine-grained systems. O'Reilly Media, Inc. (2015)
4. Florio, L., Di Nitto, E.: GRU - An approach to introduce decentralized autonomic behavior in microservices architectures. In: 2016 IEEE International Conference on Autonomic Computing (ICAC) pp. 357-362 IEEE. (2016)
5. Akbulut, A., Perros, H.G.: Software versioning with microservices through the API gateway design pattern. In: 2019 9th International Conference on Advanced Computer Information Technologies (ACIT) pp. 289-292 IEEE. (2019)
6. Taibi, D., Lenarduzzi, V., Pahl, C., Janes, A.: Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages. In: Proceedings of the XP2017 Scientific Workshops pp. 1-5 (2017)
7. Dragoni, N., Lanese, I., Larsen, S.T., Mazzara, M., Mustafin, R., Safina, L.: Microservices: How to make your application scale. In: International Andrei Ershov Memorial Conference on Perspectives of System Informatics pp. 95-104. Springer, Cham. (2017)
8. Adedugbe, O.: Development and Evaluation of a Holistic, Cloud-driven and Microservices-based Architecture for Automated Semantic Annotation of Web Documents, Doctoral dissertation, Staffordshire University (2019)
9. Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C.C., Khandelwal, A., Pu, Q., Shankar, V., Carreira, J., Krauth, K., Yadwadkar, N., Gonzalez, J.E.: Cloud programming simplified: A berkeley view on serverless computing. arXiv preprint arXiv:1902.03383. (2019)
10. Talia, D.: A view of programming scalable data analysis: from clouds to exascale. Journal of Cloud Computing, 8(1), p.4 (2019)
11. Khan, S., Shakil, K.A., Alam, M.: Big Data Computing Using Cloud-Based Technologies, Challenges and Future Perspectives. arXiv preprint arXiv:1712.05233. (2017)
12. Khan, S., Shakil, K.A., Alam, M.: Cloud-based big data analytics—a survey of current research and future directions. In: Big data analytics pp. 595-604. Springer, Singapore. (2018)
13. Marozzo, F., Talia, D., Trunfio, P.: A workflow management system for scalable data mining on clouds. IEEE Transactions on Services Computing, 11(3), pp.480-492 (2016)
14. Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., Ghodsi, A.: Apache spark: a unified engine for big data processing. Communications of the ACM, 59(11), pp.56-65 (2016)
15. Mathur, N., Purohit, R.: Issues and challenges in convergence of big data, cloud and data science. International Journal of Computer Applications, 160(9) (2017)
16. Podhoranyi, M., Vojacek, L.: Social Media Data Processing Infrastructure by Using Apache Spark Big Data Platform: Twitter Data Analysis. In: Proceedings of the 2019 4th International Conference on Cloud Computing and Internet of Things pp. 1-6 (2019)
17. Kumar, A., Bawa, S.: DAIS - Dynamic Access and Integration Services framework for cloud-oriented storage systems. Cluster Computing, pp.1-20 (2020)
18. Sun, S., Gong, J., Zomaya, A.Y., Wu, A.: A distributed incremental information acquisition model for large-scale text data. Cluster Computing, 22(1), pp.2383-2394 (2019)
19. Khaleq, A.A., Ra, I.: Cloud-Based Disaster Management as a Service: A Microservice Approach for Hurricane Twitter Data Analysis. In: 2018 IEEE Global Humanitarian Technology Conference (GHTC) pp. 1-8 IEEE. (2018)
20. Fernando, S., Birch, D., Molina-Solana, M., Mcilwraith, D., Guo, Y.: Compositional Microservices for Immersive Social Visual Analytics. In: 2019 23rd International Conference Information Visualisation (IV) pp. 216-223 IEEE (2019)
21. Houmani, Z., Balouek-Thomert, D., Caron, E., Parashar, M.: Enhancing microservices architectures using data-driven service discovery and QoS guarantees. In: The 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing CCGrid p. 10 (2020)
22. Ali, S., Jarwar, M.A., Chong, I.: Design methodology of microservices to support predictive analytics for IoT applications. Sensors, 18(12), p.4226 (2018)
23. Hsu, M.C., Lin, C.Y.: A Microservices-Based Social Data Analytics Platform Over DC/OS. In: International Conference on Network-Based Information Systems pp. 673-683. Springer, Cham. (2018)
24. Emami Khoonsari, P., Moreno, P., Bergmann, S., Burman, J., Capuccini, M., Carone, M., Cascante, M., de Atauri, P., Foguet, C., Gonzalez-Beltran, A.N., Hankemeier, T.: Interoperable and scalable data analysis with microservices: Applications in metabolomics. Bioinformatics, 35(19), pp.3752-3760 (2019)
25. Fernández-García, A.J., Iribarne, L., Corral, A., Criado, J., Wang, J.Z.: A microservice-based architecture for enhancing the user experience in cross-device distributed mashup UIs with multiple forms of interaction. Universal Access in the Information Society, 18(4), pp.747-770 (2019)
26. Dinh-Tuan, H., Beierle, F., Garzon, S.R.: MAIA: A Microservices-based Architecture for Industrial Data Analytics. In: 2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS) pp. 23-30 IEEE. (2019)
27. Hoque, S., Miranskyy, A.: Architecture for analysis of streaming data. In: 2018 IEEE International Conference on Cloud Engineering (IC2E) pp. 263-269 IEEE. (2018)
28. Innerbichler, J., Gonul, S., Damjanovic-Behrendt, V., Mandler, B., Strohmeier, F.: Nimble collaborative platform: Microservice architectural approach to federated IOT. In: 2017 Global Internet of Things Summit (GIoTS) pp. 1-6 IEEE. (2017)
29. Ciavotta, M., Alge, M., Menato, S., Rovere, D., Pedrazzoli, P.: A microservice-based middleware for the digital factory. Procedia Manufacturing, 11, pp.931-938 (2017)
30. AL-Smadi, M, Qwasmeh, O, Talafha, B, Al-Ayyoub, M, Jararweh, Y, Benkhelifa, E : An enhanced framework for aspect-based sentiment analysis of Hotels' reviews: Arabic reviews case study. 2016 11th IEEE International Conference for Internet Technology and Secured Transactions (ICITST). 98-103.
31. Rodriguez, M.A., Buyya, R: Container‐based cluster orchestration systems: A taxonomy and future directions. Software: Practice and Experience, 49(5), pp.698-719 (2019)
32. Stahl, D., Bosch, J.: Industry application of continuous integration modeling: a multiple-case study. In: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C) pp. 270-279 IEEE. (2016)
33. Sachdeva, R.: Automated Testing in DevOps. In: Pacific Northwest Software Quality Conference (2016)
34. Zhao, J.T., Jing, S.Y., Jiang, L.Z.: Management of API Gateway Based on Micro-service Architecture. In: Journal of Physics: Conference Series Vol. 1087, No. 3 (2018)
35. Eder, M.: Hypervisor-vs. container-based virtualization. Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM), 1. (2016)
36. Taherizadeh, S., Stankovski, V.: Dynamic multi-level auto-scaling rules for containerized applications. The Computer Journal, 62(2), pp.174-197 (2019)
37. Jain, N., Choudhary, S.: Overview of virtualization in cloud computing. In: 2016 Symposium on Colossal Data Analysis and Networking (CDAN) pp. 1-4 IEEE. (2016)
38. Shaheen, J.: Apache Kafka: Real Time Implementation with Kafka Architecture Review. International Journal of Advanced Science and Technology, 109, pp.35-42 (2017)