Benchmarking Big Data Architectures for Social Networks Data processing using Public Cloud Platforms

Valerio Persico[⋆◇*], Antonio Pescapé[⋆†◇], Antonio Picariello[⋆†], Giancarlo Sperlí[⋆]

[⋆] *University of Naples "Federico II", via Claudio 21, Naples, Italy*
[†] *CINI - ITEM National Lab, Complesso Universitario Monte S.Angelo, Naples, Italy*
[◇] *NM2 s.r.l., Via S. Lucia 20, Naples, Italy*

Abstract

When considering popular On-line Social Networks (OSN) containing heterogeneous multimedia data sources, the complexity of the underlying processing systems becomes challenging, and requires to implement application-specific but still comprehensive benchmarking. The variety of big data architectures (and of their possible realization) for both batch and streaming processing in a huge number of application domains, makes the benchmarking of these systems critical for both academic and industrial communities.

In this work, we evaluate the performance of two state-of-art big data architectures, namely Lambda and Kappa, considering OSN data analysis as reference task. In more details, we have implemented and deployed an influence analysis algorithm on the Microsoft Azure public cloud platform to investigate the impact of a number of factors on the performance obtained by cloud users. These factors comprise the type of the implemented architecture, the volume of the data to analyze, the size of the cluster of nodes realizing the architectures and their characteristics, the deployment costs, as well as the quality of the output when the analysis is subjected to strict temporal deadlines. Experimental campaigns have been carried out on the *Yahoo Flickr Creative Commons 100 Million (YFCC100M)*. Reported results and discussions show that Lambda out-

[*]Corresponding author
Email addresses: valerio.persico@unina.it (Valerio Persico[⋆◇*]), antonio.pescape@unina.it (Antonio Pescapé[⋆†◇]), picus@unina.it (Antonio Picariello[⋆†]), giancarlo.sperli@unina.it (Giancarlo Sperlí[⋆])

performs Kappa architecture for the class of problems investigated. Providing a variety of analyses—e.g., also investigating the impact of dataset size, scaling, cost—this paper provides useful insights on the performance of these state-of-art big data architectures that are helpful to both experts and newcomers interested in deploying big data architectures leveraging cloud platforms.

Keywords: On-Line Social Networks, Lambda and Kappa Architectures, Influence Analysis, Social Networking Benchmarking, Public Cloud Benchmarking

1. Introduction

Today, millions users generate and exchange information by means of Online Social Networks (OSNs). The development of big data technologies has enhanced OSNs features, enabling users to share their own life, generating and interacting with tons of multimedia content (text, audio, video, images) and providing it with feedbacks, comments, or feelings [30, 3]. As a matter of fact, this technological development led to the proliferation of a huge amount of data whose statistics are impressive: on average, everyday Facebook users publish 4.5 billion posts, share more than 4.7 billion status updates, and watch over 1 billion videos; Instagram—which recently reached the 300 million monthly active user marks—sees 2.5 billion likes and 70 million new photo uploads per day; concerning YouTube (where 100 hours of new contents are uploaded each minute) more than 1 billion unique users visit the site each month consuming over 6 billion hours of video. In spite of the social value of OSN data, the multimedia content of such nets may be valuable in a number of public and strategic applications such as marketing, security, but also medicine, epidemiological analyses, counter-terrorism, and so on. Therefore, the analysis of multimedia OSNs represents an extremely interesting real-application domain, characterized by extremely huge and heterogeneous datasets (e.g., type of social network, time variance of data) introducing particularly challenging problems.

The big data paradigm has been designed for achieving the optimal man-

agement and analysis of such large quantities of data (big data analytics). The performance of such analytics—possibly influenced by a number of heterogeneous factors such as the type of data, the class of problem to address, or the underlying processing systems—is of the utmost importance as impacting both the effectiveness and the cost of the overall knowledge extraction process [16]. In this context, big data benchmarks are therefore useful to generate application-specific workloads and tests in order to evaluate the analysis processes for data matching the well known Volume, Velocity, Variety and Veracity (i.e. 4V) properties.

In this paper, we focus on the real-time analysis of massive OSN streams containing both textual and multimedia information coming from multiple data sources, leveraged to feed analytics and advanced applications. We exploited two state-of-art big data streaming architectures, namely *Lambda* and *Kappa*, designing and deploying them on Microsoft Azure public-cloud Platform-as-a-Service. We produced meaningful evaluation results implementing for both architectures a novel *influence maximization and diffusion algorithm* [2] in charge of the automated real-time analysis of multimedia streams. It is worth noting that the problem of stream analysis (although not being a problem novel *per se*) is particularly challenging when applied in presence of several multimedia streams, due to the very nature of multimedia data, which is complex, heterogeneous, and of large size. This makes the analysis of multimedia streams computationally expensive, so that, when deployed on a single centralized system, computing power becomes a bottleneck. Moreover, the size of the knowledge base could be so big to prevent its storage on a single node [28].

The main contributions of the paper are as follows:

- considering the influence analysis problem as an interesting case study for OSNs, we have implemented a state-of-art algorithm [3] for addressing this problem on both Lambda and Kappa architectures and we have selected it as reference task;
- through purposely designed experimental campaigns, we have evaluated

Lambda and Kappa architectures against the reference task; indeed the evaluation has been performed adopting cutting-edge technologies leveraging state-of-the-art open-source analytics frameworks (such as *Apache Storm* and *Apache Spark*), which are more and more adopted in both industry and academia fields, proving to be the *de facto* standard technologies [5, 6];

- the evaluation deployment involved Microsoft Azure cloud PaaS services, thus allowing to obtain easily-reproducible configurations and results as well as estimating the actual costs related to the analyses according to real provider fees;

- the performance of Lambda and Kappa architectures has been evaluated along different dimensions, primarily considering timeliness, deployment costs, and outcome quality; with this goal in mind a number of different factors has been identified and taken into account during the experimental campaigns to investigate the performance of the considered architectures, such as the volume of the input dataset, the size of the deployed cluster,as well as the characteristics of the nodes composing the cluster.

In the light of the considerations above, we believe that this paper provides insightful information to both research and industry practitioners interested in deploying big data analytics systems onto the cloud. In addition, the obtained results are helpful to both newcomers (possibly interested in the qualitative impact of the analyzed factors on the performance) as well as experts (that can leverage provided information to optimize their own deployments).

The paper is organized as follows. In Section 2 a state of the art in big data benchmarking and performance evaluation on cloud infrastructures is reported. Section 3 describes Lambda and Kappa architectural models for OSN applications. In Section 4 we introduce the benchmarking task based on Influence diffusion problems. In Section 5 we describe the dimensions along which the architectures taken into account are evaluated, discussing the choices leading to

the analyses, whose experimental results are reported and discussed in Section 7. Finally, discussions, lessons learned, and conclusions are reported in Section 8.

2. Related work

Due to the tremendous interest in big data by academia, industry, and a larger and larger user base, a variety of solutions and products has been released in recent years. With their gradual maturity, a growing need to evaluate and compare these solutions has been observed. Therefore, benchmarking big data systems has notably attracted the interest of the scientific community. Indeed, benchmarking solutions hugely facilitate performance comparison between equivalent systems, providing useful information for procurement decisions, configuration tuning, features planning, deployment validation, and many other efforts in engineering, marketing, and customer support [9].

2.1. Big data benchmarking frameworks

Only few years ago, the scientific literature advocated cautious approach to developing "big data benchmarks" for fairly and properly evaluating big data systems, as little was known about real life use cases to design enough general solutions [9]. However, in more recent years, several big data benchmarking frameworks have been proposed.

Chen et al. [8] focused on the *MapReduce* framework and provided a characterization of its workloads when partially driven by interactive analyses. They showed how these workloads display diverse behaviors, thus invalidating some common assumptions. Moreover, their results were helpful to create a data-processing benchmark for MapReduce. The joint effort from academia and industry, led Ghazal et al. [12] to the proposal of *BigBench*. It is based on a fictitious retailer selling products to customers via physical and on-line stores, and considers structured, semi-structured, and unstructured data. In more details, the proposal covers a data model, a synthetic data generator, and the related workload description. BigBench aimed at overcoming previous efforts in the

area of big data benchmarks (e.g., *YCSB* [10] and *PigMix* [29]) that were considered as island solutions and not policed by any industry consortia. Similarly, Wang et al. [39] proposed *BigDataBench* proposing a benchmarking methodology focused on web-search engine workloads. Ming et al. [26] focused on data generation and developed *Big Data Generator Suite* to efficiently generate scalable big data while employing data models derived from real data to preserve data veracity.

Differently from the contributions cited above, rather than proposing a general approach for developing general benchmarking strategies, our work focuses on the performance of state-of-art big data architectures when dealing with specific workloads related to popular influence analysis tasks in OSNs.

2.2. Performance evaluation of big data systems and cloud infrastructures

Some proposals in the literature aimed at benchmarking specific implementations of big data systems.

Möller and Özcep [27] evaluated the accuracy and the response time of a stream-data management system based on *Apache Storm* leveraging a workload the simulates toll charging in a fictional urban area. Wei et al. [40] benchmarked *Apache Spark* with a popular parallel machine learning training application. Singh [35] focused on monitoring data and proposed an evaluation for a big data system (leveraging *Apache Hadoop* and Apace Spark among others) investigating latency and fault tolerance. Satra [34] carried out repeated runs of a movie review sentiment analysis task in Apache Spark on Google Compute Engine clusters of various sizes. In a similar way, we focus on evaluating analytics performance on a public-cloud infrastructure.

In some cases, the performance of big data architectures was also compared to dedicated serial stand-alone implementations (i.e., not leveraging frameworks such as MapReduce, Spark, or Storm [40, 34]. However, as the implementation of these specialized softwares is proved to require much more effort (also possibly impacting scalability, maintainability, etc. in a negative way) we do not consider them in our experimental evaluation, as they cannot be considered a

valid alternative in general terms.

2.3. Architectures for the analysis of social-network Big Data

A number of contributions in the literature specifically focused on social-network data and architectures designed for processing it.

Tan et al. [36] focused on the interplay among connected people, business entities, and connected computers (such as cloud infrastructures) and discuss how it has opened new solutions with regard to how humans can interpret connected data. Erling et al. [11] described a social network benchmark and presented database benchmarking innovation in terms of graph query functionality tested, correlated graph generation techniques, as well as a scalable benchmark driver on a workload with complex graph dependencies. Gribaudo et al. [15] presented a modeling approach providing an evaluation tool to support design choices about parameters and eventually lead to better designs of applications based on lambda architectures. Basanta-Val et al. [6] propose an architecture extending Spark to support time critical map-reduce and Storm with a backward time-critical stack for performing time-critical analytics.

In this paper, we focus on a real application designed for performing influence analysis on OSNs and implemented leveraging different architectural philosophies as well as state-of-art analytics frameworks.

3. Big Data Architectures

In this section, we briefly summarize the characteristics of the two big data architectures we consider in this work—namely, Lambda and Kappa—that are adopted for executing analytics.

3.1. Lambda architecture

Lambda architecture was introduced by Marz and Warren [25] and provides a set of architectural principles to ingest and process both stream and batch data, in a single big data architecture. In Figure 1 the main components of the architecture are depicted. This infrastructure defines layers both for batch

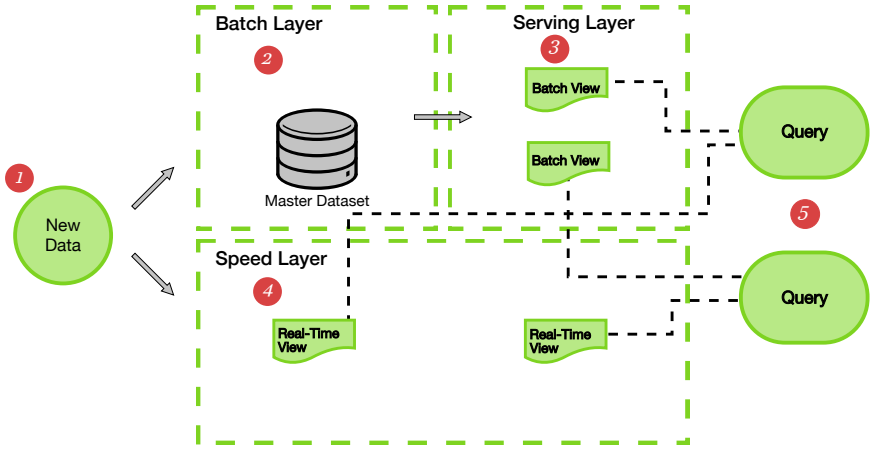


Figure 1: Lambda Architecture.

processing (that can be seen as a *data lake* system) and streaming data analysis. Lambda architecture balances latency and throughput using batch processing to provide accurate view of historical data and on-line data obtained by the ingestion phase (number ① in Figure 1) from heterogeneous sources. The *Batch Layer* (②), also called *cold path* allows to process pre-computed large amount of data; in addition, a *Speed Layer* (④), also called *hot path*, processes incoming data for rapid consumption using an incremental approach in order to reduce the computational costs. In particular, the Batch Layer combines incoming data with historical ones, processing them by means of appropriate analytical models. However, it also introduces high latency, due to its high computation time. The Serving Layer (③) indexes the batch views so that they can be queried in low-latency, ad-hoc way. Any incoming query (⑤) can be answered by merging results from batch views and real-time views. The views can be stored in distributed databased to handle reads.

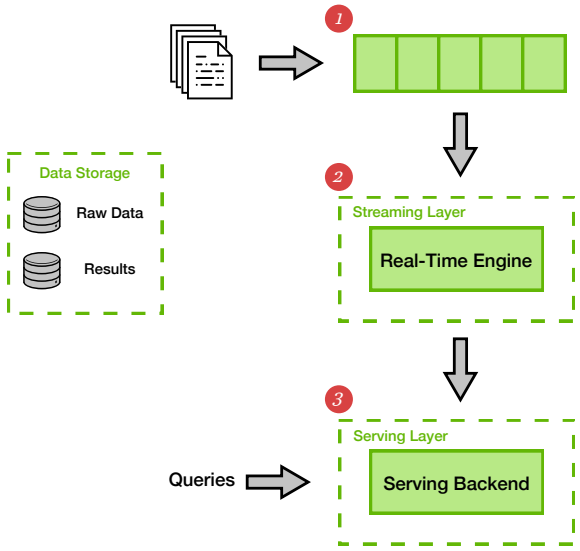


Figure 2: Kappa Architecture.

3.2. Kappa architecture

Kappa architecture was first described by Kreps [22] for data streams. The key idea is to use a single real-time layer to process stream and batch data (Data storage box in Figure 2), avoiding the problems due to the use of two different frameworks. For this reason, Kappa architecture focuses on data processing more than storage and it is considered an *event-streaming platform*. Kappa architecture is natively designed for handling data streams, such as actions of OSN users, events generated by devices connected in the Internet of Things, or transaction processing systems.

As shown in Figure 2, this infrastructure is composed of: a Stream Processing module (②), that ingests ordered data events (①) and a Serving Layer (③), that manages the query results. Kappa architecture is based on the idea to compute the incoming data through a stream processor able to process data at high rate and fed them into an auxiliary stores for serving the analytics clients.

4. Task definition

The estimation of influence exerts among users is an important task in OSNs, contributing to to properly defining the social communities and improving the performance of recommender systems. In this section we first introduce the OSN model we refer to (Section 4.1); then, we provide the details of the algorithm we have implemented for analyzing interactions among OSN users (Section 4.2). This algorithm represents the reference task executed by the big data architectures we benchmark. The workload imposed to the big data architectures under investigation depends upon both the task executed and the amount of data to process.

4.1. OSN model

Several approaches have been proposed to model social networks through different kinds of graphs that are usually incrementally built over the time. Kempe et al. [19] discussed about *temporal network* in which labels on edges are time dependent. In particular, some approaches are based on the analysis of interactions made by users. Goyal et al. [14] proposed a model that learns how influence flows across the network based on propagation traces for estimating expected influence spread. More recently, Hines et al. [17] proposed a Markovian influence-graph model, analyzing the data from many cascading failure simulations in which the outage probabilities depend on the analysis of previous outages. Amato et al. [2] proposed an OSN data model based on sliding windows leveraging user-to-content actions to estimate the likelihood that one users exerts influence on another one for influence analysis applications.

In this work, we leverage for benchmarking purposes the model and the algorithm introduced in [2]. The novelty of the analyzed method concerns the learning phase based on the analysis of users' behaviors from past logs; more in details this approach is the first attempt to estimate influence probability leveraging user content actions such as users' reactions or comments on user generated content (e.g., post, pictures and so on). The algorithm has widespread

applicability as a wide range of applications potentially benefits from its outcomes. For instance, recommender systems could improve the performance of recommendation combining the interest of each user with the products purchased by its social circle, viral marketing strategy can obtain benefits from the choice of a users subset that can influence the largest number of people, advertising campaigns can be focused on people that share common interests etc. Its basic working principles are summarized in the following.

The algorithm aims at assessing the likelihood that a user u_i can be influenced by a user u_j. The algorithm implements an iterative approach based on *coalescence windows* of different sizes, describing different points of views of the analyzed networks. In more details, the algorithm divides the time axis in different time windows, before analyzing each of them separately. Finally, the results are properly combined in the *influence graph*.

In simple terms, an analyzed OSN stream consists of a set of log tuples (s). Each tuple corresponds to an action ($s.action$) performed by a user ($s.user$) on a given object ($s.object$) at a given time instant ($s.timestamp$) with additional attributes associated to it.

In particular, analyzing an OSN stream, it is possible to define the *reaction operator* between two actions made on the same objects or similar ones. This operator is the main element to define the influence probability that is composed by the following two components: *reactivity* and *shareability*. The reactivity of user u_i with respect to user u_j, is the ratio between number of reactions of u_i with respect to the actions made by u_j ($r_{u_i u_j}$) and total number of reactions of u_i (a_{u_i}):

$$Reactivity_{ij} = \frac{|r_{u_i u_j}|}{|a_{u_i}|} \tag{1}$$

This ratio represents how the user u_i is *reactive* with respect to u_j. Indeed, it is easy to note that this value is equals to one if the user u_i reacts on each content published by u_j.

The shareability of user u_j with respect to user u_i corresponds to the ratio between the number of reactions of u_i with respect to u_j ($r_{u_i u_j}$) against total number of actions of u_j (a_{u_j}):

$$Shareability_{ij} = \frac{|r_{u_i u_j}|}{|a_{u_j}|} \tag{2}$$

This ratio represents how the user u_i reacts to the action of u_j, describing the capability of user u_j to influence directly user u_i. The highest influence from user u_j to user u_i is when every action performed by user u_j is reacted by user u_i, as well as the user u_i is limited only to react on actions made by u_j.

4.2. Algorithm implementation

Details of the procedure for assessing influence are reported in Algorithm 1. It is possible to note that the analyzed algorithm is mainly impacted by the two iterative constructs shown at the lines 20–21 lying in $O(n^2)$ asymptotically. Furthermore, starting from line 18 to line 29 it is shown in details the instruction for graph updating, in which firstly the initial analysis time is modified (line 19), successively the weights of the edges are updated (lines 20–26) and finally data structures are cleared (lines 27–29).

5. Performance Evaluation

Big data systems are required to provide timely, cost-effective, and quality answers to data-driven questions [21]. Therefore, we consider three main dimensions along which the architectures taken into account are evaluated: *timeliness*, *cost*, and *quality of the output*. Here we discuss the choices leading to the analyses whose experimental results are detailed in Section 7.

For what concerns timeliness, works in the scientific literature have proposed and adopted several metrics to investigate the efficiency of big data systems. These metrics are often categorized in (i) *architectural* and (ii) *user-perceivable*

Algorithm 1 τ-algorithm

procedure τ-ALGORITHM(S,Δt,t_s)
- - Input: S (Tuples temporal stream)
- - Input: Δt (Interval analysis)
- - Input: t_s (Start time of analysis)
- - Output: G=(V,E) (Direct graph)
- - Temporary : **P** (Square matrix whose element p_{ij} represents the influence exerts from u_i on u_j)
- - Temporary : s (Analyzed tuple)
- - Temporary : t_i (Timestamp)
- - Temporary : R (Square matrix whose element $r_{u_i u_j}$ represents the number of reactions of u_i with respect to u_j)
- - Temporary : A (Vector whose element a_{u_j} is the total number of actions of u_j)
- - Temporary : $H_{o,u}$ (Hash map stores users that interact with a given object)
$t_i \leftarrow t_s$
$R \leftarrow \emptyset$
$A \leftarrow \emptyset$
$H_{o,u} \leftarrow \emptyset$
while (There are tuples to analyze ($S \neq \emptyset$)) **do**
    s $\leftarrow$ dequeue(S)
    if s.timestamp$\geq t_i + \Delta t$ **then**
        $t_i \leftarrow t_i + \Delta t$
        for i $\in \{1, \cdots, |V|\}$ **do**
            for j $\in \{1, \cdots, |V|\}$ **do**
                $p_{ij} \leftarrow \frac{r_{u_i u_j}}{a_{u_i}} * \frac{r_{u_i u_j}}{a_{u_j}}$
                if ($e_{ij} \in E$) **then** $w(e_{ij}) \leftarrow \frac{w(e_{ij})+p_{ij}}{2}$
                else
                    E $\leftarrow E \cup e_{ij}$
                    $w(e_{ij}) \leftarrow \frac{p_{ij}}{2}$
        $H_{o,u} \leftarrow \emptyset$
        $R \leftarrow \emptyset$
        $A \leftarrow \emptyset$
    if (s.object $\in H_{o,u}$) **then**
        $a_{s.user} \leftarrow a_{s.user}+1$
        for (v $\in H_{o_{s.object,u}}$) **do**
            $r_{vs.user} \leftarrow r_{vs.user} + 1$
        if (s.user $\notin$ V) **then**
            V $\leftarrow V \cup \{s.user\}$

metrics [39]. The former are typically used by architectural research and comprise [39, 4]: million instruction per second (MIPS); L1-cache misses for 1000 instructions (MPKI); CPU Utilization; memory Utilization; disk Utilization; network I/O. The latter may depend upon the specifications of the workload running onto the system and encompass [39, 7]: the number of processed requests per second, used to measure the throughput of on-line service workload; the number of operations per second, used to evaluate Cloud OLTP workloads; the quantity of data processed per second, used for analytic workloads; number of edges per second, that can be leveraged in the case of graph-data input. While architectural metrics can be leveraged even when comparing system performance under different workloads (e.g., database servers vs. MapReduce workloads), user-perceivable metrics can be more conveniently observed and used by users, as they provide a general view on the system modeled as a black box.

In accordance with the specifications of the considered application, since our main goal is to provide a view of the performance of the implemented big data architectures that can be directly understood and leveraged also by the users, hereafter we consider the total processing time as the main metric for evaluating the timeliness.

It is worth noting that this choice is also encouraged by the PaaS cloud deployment we leverage (additional details are reported in Section 6.2), that partially or totally hides architectural information, as preventing direct access to the virtual machines (VMs) composing the cluster. For instance, because of the PaaS abstraction layer, we are not allowed to directly measure neither the CPU and memory utilization of each VM, nor to characterize the network interconnecting them within the provider datacenter, as done in recent literature [31, 32].

Moreover, as the size of the input dataset is expected to impact the timeliness of the system, we run the considered task varying the number of tuples composing the dataset across different orders of magnitude. This allows us to evaluate and compare the scalability of the two architectures as function of the volume of the data provided as input, as well as to assess the potential impact

of bootstrap phases onto timeliness.

Focusing on the real-time resource scalability enabled by the cloud paradigm, for what concerns costs, our analysis aims at investigating (similarly as done in [4]) how the performance of the implemented architectures varies when different amounts of resources are available—or equivalently, when the deployment is subjected to different budget constraints. In more details, we consider how the timeliness of the architectures varies both when resizing the cluster (horizontal scaling) and under different configurations for the nodes in the cluster (vertical scaling) [33].

Finally, we also focus on the quality of the outcome of the architectures investigating how it degrades when it is subjected to different temporal deadlines. More specifically, for this analysis, we evaluate how the information processed by the algorithm until a given time point are valuable to identify the main influencer in the social network.

The dimensions taken into account are briefly summarized in Table 1, with the related metrics and research questions. The main results of the analyses derived from the Table are discussed in Section 7.

Table 1: Summary of the performance dimensions investigated with related metrics and research questions.

Dimension	**Metric adopted**	**Research Questions**
Timeliness	Total processing time	*Which architecture performs better?* *How does performance vary with input size?*
Cost	Deployment cost per analysis	*How does performance vary with cluster size?* *How does VMs characteristics impact performance?*
Outcome quality	Top-k influencer Recall	*How good is the outcome when the architectures are subjected to different time constraints?*

6. Experimental Testbed

In this section, for the sake of repeatability of the analyses, we provide the details of the experimental testbed we set up. We first describe the implementation details of the architectures in Section 6.1; then we detail the cloud

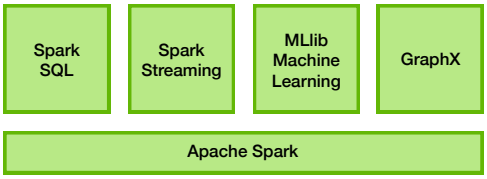


Figure 3: Spark modules.

deployment in Section 6.2; finally, we describe the procedures adopted for obtaining the input datasets in Section 6.3.

6.1. Architecture implementation

Lambda architecture has been implemented using Spark, able to perform fast computation through in-memory processing. Spark is an open-source, general purpose, scalable, high available and reliable computing platform, initially designed to outperform its counterpart Apache Hadoop. Spark analyses leverage a read-only collection of data items—called Resilient Distributed Dataset—partitioned and distributed over a cluster of machines. Spark provides APIs for programming entire clusters, including support for SQL queries, streaming data, machine learning and graph processing. Spark makes available different analyses through a number of modules (Figure 3), such as Spark Core (on which all the other applications are built), Spark Streaming, MLlib, and GraphX.

For what attains Kappa architecture, several distributed platforms have been proposed for processing streaming data from heterogeneous data sources, such as Apache Kafka, Apache Flink or Apache Storm. We leverage Storm, a distributed platform that processes raw stream of data. Storm provides several features, such as cluster balancing (when a new node is added to the cluster), fault tolerance (that guarantees data processing), etc. As shown in figure 4, it is composed of several *Worker Nodes* (also called Supervisors) that execute tasks; and at least one *Master Node* (executing the daemon Nimbus) in charge of assigning tasks to the machines and monitoring their performance. All these

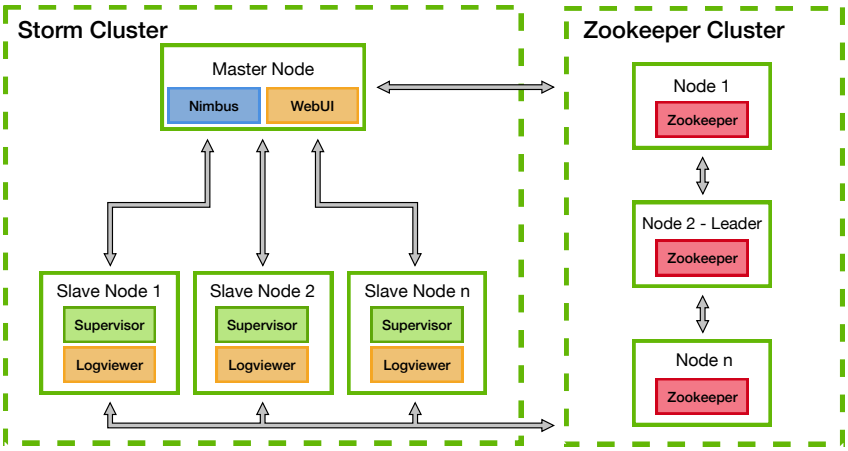


Figure 4: Storm Architecture.

nodes are coordinated by an orchestrator, namely *Zookeeper*.

Among the several implementations available for both architectures, we have chosen Apache Spark and Apache Storm, as they are the two main technologies [18] used in several research fields and commercials firms. For instance, from a research point of view, Spark and Storm technologies have been recently used in several application fields, such as in industrial environments for data processing [5] and in the network field with network monitoring purposes [42]. From a commercial viewpoint, Spark[1] and Storm[2] are adopted by several big companies such as Spotify, Twitter, and Samsung.

It is worth noting that, in spite of the large number of choices available, in this work we aim at comparing Lambda and Kappa architectures leveraging two state-of-art implementations, rather than investigating the performance discrepancies possibly generated by the different implementations of a given architecture.

[1] https://spark.apache.org/powered-by.html
[2] http://storm.apache.org/Powered-By.html

6.2. Cloud deployment

For the experimental evaluation, we have deployed the implemented architectures onto the public-cloud infrastructure [1] made available by *Microsoft Azure* one of the market leaders representing a choice largely adopted by most of the customers [23].

Among all the data centers located world-wide, we chose to deploy our architectures in the *West Europe* region. Although some specific performance indexes may be subjected to variation when changing region [32], this variation is expected not to dramatically affect the observed timeliness. Also, costs imposed by the provider may change across regions. Converting costs from a region to another can be easily done leveraging proper conversion coefficients derived from current fees, however.

In more details, the evaluation has been carried out on *Azure HDInsight*, a cloud distribution of the Hadoop stack on Microsoft Azure. This choice well fits with our needs, as this PaaS service includes the support to the Hadoop technology stack together with related software and utilities, including Apache Spark and Apache Storm, among the others. It is worth noting that similar deployments can be obtained also using other public-cloud platforms. For instance, leveraging Amazon Web Services (AWS), Lambda architecture can be easily deployed onto Elastic Map Reduce. On the other hand, since Kappa architecture is not natively supported on AWS at time of writing, its configuration could be less straightforward.

During the deployment phase, HDInsight allows to customize the cluster acting on a list of parameters, such as (i) the number of VMs composing the clusters (ii) the number of cores, (iii) the type of processor, and (iv) the size of the RAM. Different configurations for VMs are provided for responding to common requirements (e.g., running compute-intensive or memory-intensive tasks). Leveraging these configuration parameters, we are able to evaluate the performance of the architecture both considering *horizontal scaling* (i.e., when additional VMs join the cluster) and *vertical scaling* (i.e., when enhancing the characteristics of each VM).

6.3. Dataset

For our experimental analyses, we took advantage of the *Yahoo Flickr Creative Commons 100 Million (YFCC100M)*, a Flickr multimedia collection dataset used for academic purposes, shared under a *Creative Commons* license [38].

For all our experimentation we leveraged the Flickr API[3] to crawl the following information (*actions*) concerning to each image: *Photo Publishing* (author ID, photo ID, Timestamp); *Comment Publishing* (author ID, comment ID, photo ID, Timestamp); *Favorites* (author ID, photo ID, Timestamp). Table 2 summarizes the information about the dataset.

Table 2: Overall YFCC100m dataset characterization.

Action	#
Photo Pub.	67.812.283
Comment Pub.	19.033.208
Favorites	13.418.945
Avg. actions per user	91

In order to evaluate the performance of the architectures against different volumes of data provided as input, we partitioned the YFCC100M dataset, obtaining datasets of four different sizes. Resulting datasets consist of 1M, 10M, 60M, and 100M tuples, and are labeled as *Small* (S), *Medium* (M), *Large* (L), and *Extra-Large* (XL) respectively. It is worth noting that to obtain the instances of S, M, L and XL datasets, we performed a different random sampling of the initial dataset at each run to avoid introducing bias possibly due to specific samples in the data. Moreover, at each run both architectures were fed with the exactly same dataset. The methodology described, allows us to perform a fair analysis and obtain comparable results.

[3] https://www.flickr.com/services/api

7. Experiments and Results

We report here the results obtained through our experimental campaigns. In Section 7.1 we discuss the timeliness of the architectures when they are fed with different volumes of data in input; in Section 7.2 we evaluate how the performance improves when increasing the number of nodes composing the architectures (*horizontal scaling*) as well as when deploying VMs with enhanced characteristics (*vertical scaling*); in Section 7.3 the trade-off between cost and performance is analyzed; finally in Section 7.4 we experimentally assess how quality degrades when the analysis is subjected to strict time constraints.

7.1. Timeliness with respect to data volume

In this section we analyze the performance of the big data architectures in terms of timeliness, leveraging overall running time for Algorithm 1 as evaluation metric. First, this analysis aims at investigating the performance figures of the architectures when they are fed with different volumes of data. Secondly, it also provides comparative insights.

To provide a fair comparison, we have implemented both architectures with two executor nodes (namely, Supervisor and Worker nodes for Storm and Spark frameworks, respectively). As a result, according to the nodes and topologies these architectures require to be deployed (see Section 3), the overall VMs deployed for implementing Lambda and Kappa architectures is different for this analysis. Regarding the characteristics of each node in terms of memory and computing capabilities, we have referred to those suggested by the provider in the quick setup procedure to obtain representative configurations. It is worth to note that, when strictly following these suggestions would have led to slight different setups, we opted for the more costly choice, in order to satisfy the minimum requirements for both architectures. In more details, we deployed A7 VMs for executing tasks. According to best practices, this choice is slightly overkill in terms of memory for Kappa architecture, but allowed us to design a fair comparison in terms of VM resources.

Table 3: Cluster configuration for timeliness analysis. In order to reproduce configurations usually deployed by cloud users, we have referred provider's setup suggestions when selecting VMs type.

	Node	**Role**	**Type**	**CPU (cores)**	**RAM (GB)**	**Cost (€/h)**
Storm	Zookeper	Coordinating cluster	A3	4	7	3.73
	Nimbus	Assigning tasks to Supervisors	A3	4	7	
	Supervisor	Executing task	A7	8	56	
Spark	Head	Assigning tasks to workers	A3	4	7	2.92
	Worker	Executing task	A7	8	56	

The general setup for the two architectures is as follows: Kappa Architecture consists of 2 nodes for assigning tasks and 2 nodes for carrying out tasks (executors); Lambda Architecture is made up of 3 nodes for coordinating the cluster, 2 nodes for assigning tasks, and 2 nodes for executing tasks. Additional details about the clusters implementing the two architectures can be found in Table 3. The hourly cost for running Kappa and Lambda architectures according to the above configurations was 3.73 and 2.92 €/h, respectively.

To investigate performance variance, for each architecture we run the analysis three times for each dataset size. The same choice applies also for remaining analyses, if not stated otherwise. The choice of running three experiments for each investigated scenario has been guided by the results of a preliminary campaign, where a limited number of scenarios was tested up to ten times. Results statistics obtained with three runs provided a good fit with those obtained through longer campaigns in both average and inferred variability. Taking advantage of low variability, we were encouraged in limiting the number of iterations, thus achieving a good trade-off between results accuracy and costs. Figure 5 reports the results in terms of running time (mean and standard deviation) for both architectures when they are fed with S-, M-, L-, and XL-sized

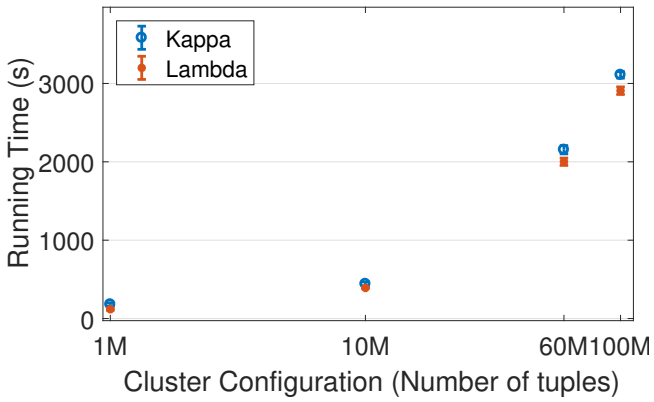


Figure 5: Performance in terms of running time for Lambda and Kappa architectures (mean and standard deviation). Results reports that Lambda architectures perform up to 18% better, on average.

datasets.

First, the performance in terms of timeliness is highly impacted by the volume of the input dataset for both architectures. Kappa architecture suffers more from the increasing input volume. In more details, with respect to the response time observed with S dataset, we observed a +303% (+293%), a +1486% (+1377%), and a +2209% (+1940%) for Kappa (Lambda) architecture when M, L, and XL datasets are provided as input, respectively. In particular, the in-memory computation of Lambda architecture allows to better handle different amount of data having enough computational resources. However, considering that M, L, and XL datasets orders of magnitude larger, both architectures exhibit sublinear scaling.

For what concerns performance variability, results report that its absolute value (in terms of standard deviation) grows with dataset size. For both architectures we observed the largest coefficient of variation (CoV)[4] when providing the L dataset as input. However, also in this case, CoV value is small, being equal to around 2.4% for both architectures, thus highlighting predictable

[4]The coefficient of variation is the ratio of the standard deviation over the mean ($CoV = \frac{\sigma}{\mu}$).

timeliness when analyzing datasets of the same size.

Comparing the results obtained by the two architectures, Figure 5 shows how Lambda architecture reports better performance in terms of timeliness, on average. Indeed, the bigger the input volume, the larger the performance discrepancy is. Lambda architecture performs from 12% to 18% better, on average. An explanation for this results can be found in the in-memory computations performed by Spark (which also requires to equip executor VMs with more memory, and therefore impacts deployment cost).

Since deployment cost is a major issue, we have investigated related aspects through purposely designed campaigns and discuss the results in the following sections.

7.2. Scalability with respect to cluster configuration

The aim of the analysis provided in this section is to evaluate the performance of Algorithm 1 on varying cluster configurations, (i) acting on the number of VMs composing the cluster and thus reflecting different budget constraints, and (ii) deploying VMs with different characteristics.

To provide this experimental evaluation, we fed the architectures with L-sized datasets (see Section 6.3) while varying either the number of executor nodes or their type.

Horizontal scaling. In the context of the horizontal scaling, we provide the results of two distinct campaigns. In both campaigns we have considered deployments with 2, 4, and 6 executor nodes. The upper bound has been defined according to the limitations imposed by the specific agreement stipulated with the provider. While in the first experimental campaign we blindly chose the type of the executors among those suggested in the online quick setup provided by HDInsight, in the second one we implemented the advices we obtain when contacting the support asking for suggestions about our specific use cases. Therefore, in the first campaign we leveraged A7 executors, while for the second one we selected Dv2 VMs (that are memory-optimized). In accordance with

Table 4: Cluster configuration for horizontal-scalability analysis.

(a) *Standard* deployment.

	Storm	Spark
HOR-S_STD	2× A3 nimbus 3× A3 zookeper **2× A7** supervisor	2× A3 head **2× A7** worker
HOR-M_STD	2× A3 nimbus 3× A3 zookeper **4× A7** supervisor	2× A3 head **4× A7** worker
HOR-L_STD	2× A3 nimbus 3× A3 zookeper **6× A7** supervisor	2× A3 head **6× A7** worker

(b) *Optimized* deployment.

	Storm	Spark
HOR-S_OPT	2× A3 nimbus 3× A3 zookeper **2× D4v2** supervisor	2× D12v2 head **2× D13v2** worker
HOR-M_OPT	2× A3 nimbus 3× A3 zookeper **4× D4v2** supervisor	2× D12v2 head **4× D13v2** worker
HOR-L_OPT	2× A3 nimbus 3× A3 zookeper **6× D4v2** supervisor	2× D12v2 head **6× D13v2** worker

the selected configurations, we will refer to these to as *standard* and *optimized* deployments, respectively. Additional details about cluster configuration are provided in Table 4a and Table 4a for the former and the latter experimental campaign, respectively. Table 5 reports details about the memory-optimized VMs.

For ease of discussion, we dubbed the configurations for each deployment (that differ in the number of executor nodes) as *small*, *medium*, and *large*. In more details, we label these configurations as HOR-S_STD, HOR-M_STD, HOR-L_STD, and HOR-S_OPT, HOR-M_OPT, HOR-L_OPT, for standard and optimized deployments, respectively.

It is worth noting that, according to best practices [24], configuring Lambda

Table 5: Cluster Configuration for horizontal-scaling campaign (VM types).

	Node	Type	CPU (cores)	RAM (GB)
Storm	Nimbus	A3	4	7
	Zookeper	A3	4	7
	Supervisor	D4v2	8	28
Spark	Head	D12v2	4	28
	Worker	D13v2	8	56

and Kappa architectures is subjected to different implementation constraints. Therefore, the deployments above led to slightly differing costs, because of the peculiarities of the architectures. Although this analysis is mainly intended to investigate how performance of each architecture varies when subjected to different budget constraints (rather than directly comparing performance obtained by the two architectures) we have planned these experimental campaigns such that hourly costs for configurations in the same deployment class (i.e. L, M, or, H) are coarsely comparable. The deployment for Kappa architecture resulted more expensive with respect to Lambda with the same number of executors (with a discrepancy of 0.81€/h, at most) due to the cost overhead imposed by nodes other than executors to be deployed.

Results reported in Figure 6 show how performance dramatically improves when running the algorithm on larger clusters, both when considering Kappa and Lambda architectures. As shown in Figure 6a, when increasing the number of executors from 2 (low-performing configuration) to 4 (medium-performing configuration) the running time reduces by 18.03% and 18.31% for Kappa and Lambda architectures, respectively. When adding two more executor nodes to the cluster, the running time further decreases by 8.96% and 6.34% more for the former and the latter, respectively.

The comparison of Figure 6a to Figure 6b unveils how a qualitatively similar trend is observed also with the optimized deployment. Moreover, results also show how Lambda architecture benefits from the optimized deployment more. This result is explained by the higher quota of in-memory operation performed

with respect to Kappa architecture.

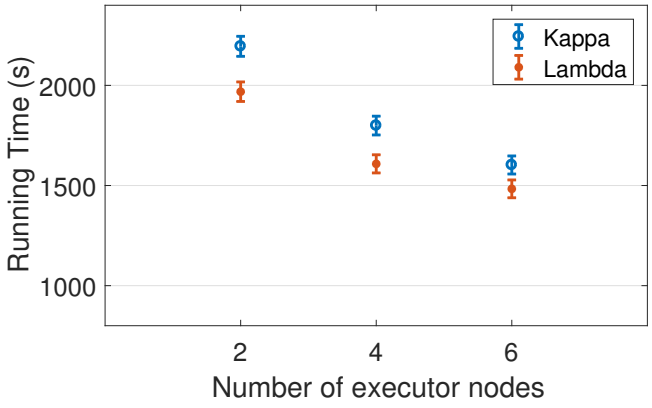
In more general terms, observing the trends of performance on increasing the cluster size, additional benefit is expected to be registered when further adding VMs. However Figure 6a and Figure 6b prompt that the marginal improvement generated by increasing the clusters is progressively reduced for both architectures and both deployments considered, because of the management overhead introduced by larger clusters.

Vertical Scaling. To obtain a more detailed view about the impact of deploying VMs with different characteristics, we purposely designed an experimental campaign to evaluate the impact of vertical scaling. The related results are discussed here.

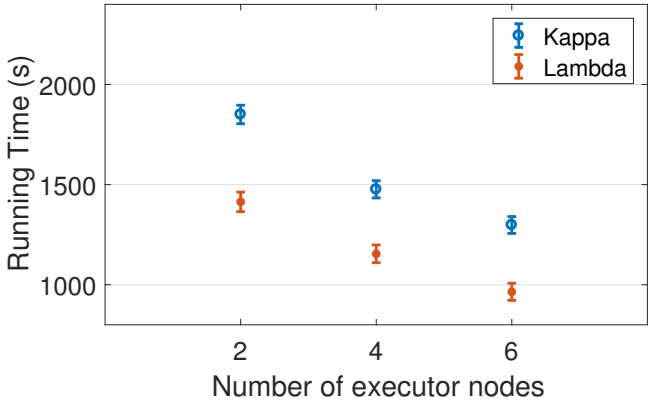
In more details, for this analysis we have deployed clusters with two memory-optimized executors, whose type has varied among D4v2, D5v2, and D14v2. These VM types differ in both the number of cores as well as memory available. Resulting configurations (dubbed VER-S, VER-M, and VER-L) are detailed in Table 6. Also in this case, configuration choices impact hourly deployment costs. Low-, medium-, and high-performing configurations cost 3.45, 3.65, and 5.51 (4.4, 4.6, 6.46) €/h respectively for Kappa (Lambda) architecture.

Results in Figure 7 show how Kappa and Lambda architectures significantly improve performance when enhancing VMs characteristics. Running time varies from around 1800s to 1200 (1000) for Kappa (Lambda) architecture, generating improvements up to 33.19% (42.12%).

Average performance discrepancy between the two architectures varies according the type of the VMs in the cluster. While, for low-performing configurations the performance of Kappa and Lambda architecture vary only by 2%, the highest discrepancy was observed for the medium-performing configuration (18%). Interestingly, Lambda deployed with VER-M configuration outperforms Kappa implemented with VER-L configuration.



(a) *Standard* deployment.



(b) *Optimized* deployment.

Figure 6: Results for horizontal-scalability analyses. Figure 6a reports results related to Hor-S-std, Hor-M-std, and Hor-L-std detailed in Table 4a, while Figure 6b reports results related to Hor-S-opt, Hor-M-opt, and Hor-L-opt detailed in Table 4b.

Table 6: Cluster configuration for vertical-scalability analysis.

	Storm	Spark
VER-S	2× A3 nimbus 3× A3 zookeper 2× **D4v2** supervisor	2× D12v2 head 2× **D4v2** worker
VER-M	2× A3 nimbus 3× A3 zookeper 2× **D5v2** supervisor	2× D12v2 head 2× **D5v2** worker
VER-L	2× A3 nimbus 3× A3 zookeper 2× **D14v2** supervisor	2× D12v2 head 2× **D14v2** worker

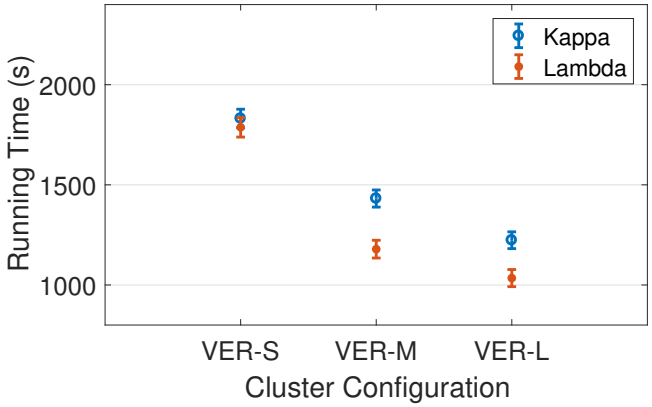
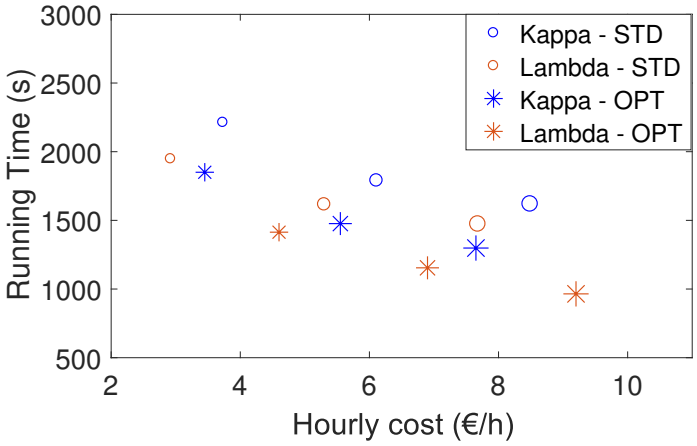


Figure 7: Results for vertical-scalability analysis. Results are related to VER-S, VER-M, and VER-L detailed in Table 6.

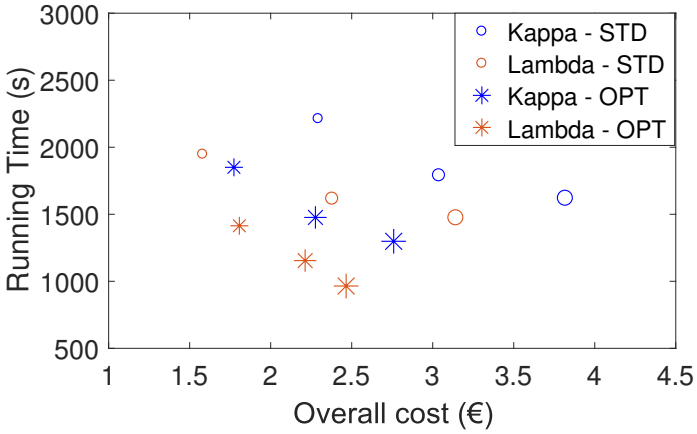
7.3. Cost analysis

The analyses above leveraged different configurations that led to different hourly deployment costs. However, we found that lower hourly costs are also associated to higher running time, which impacts the overall cost of the analysis in accordance with the pay-as-you-go billing model. In this section we deepen the impact of the different configurations tested on the cost for analyzing L-sized datasets.

Figure 8a compares the hourly cost associated to each deployment to the results in terms of running time. As shown in the figure, we have considered deployments ranging from 2.92 to 9.2 €/h. A general trend can be observed,



(a) Hourly cost vs. running time.



(b) Overall cost vs. running time.

Figure 8: Results for cost analysis. The dimension of the points in the figures reflects different cluster configurations (as detailed in Table 4a), i.e. larger points are related to larger configurations and vice versa). Higher hourly deployment costs generates dramatically lower running time, thus positively impacting overall cost.

as the highest the hourly cost associated to the deployment, the shorter the running time is.

When considering the overall cost associated to each analysis (i.e., the product of the running time and the hourly cost) additional conclusions can be drawn. Indeed, Figure 8a shows how the overall best performance in terms of running time is achieved by Lambda architecture (around 1000 s), obtained leveraging a cluster with 6 memory-optimized executors leased at 9.2€/h (the overall highest hourly cost). However, this setup generates a total cost equal to 2.46€, that is lower than 4 out of 11 other configurations tested. Every other deployment with lower hourly costs generates higher overall costs, because of the longer running time. The deployment associated to the highest cost for running the overall analysis is Kappa architecture with 6 A7 executors. In this case, the performance advancement generated by the higher hourly cost does not disclose benefits in terms of running time, also impacting overall costs.

More in general, experimental results witness how leveraging memory-optimized VMs guarantees cost saving, in spite of the higher hourly cost, to both architectures. Moreover, although the proper trade-off has to be evaluated according to specific of both user and application, both overall costs and running time benefits from leveraging larger clusters and its impact is more evident on Lambda architecture.

7.4. Quality vs. timeliness

In this section, we assess the quality of the outcome of Algorithm 1 when it is executed on Lambda and Kappa architectures under differing time constraints. To provide this qualitative comparison, we refer to the partial outcomes of the algorithm for addressing the influence analysis problem obtained at different points in time.

Two steps are usually required to deal with this influence analysis problem: first, a diffusion model has to be defined, to describe how the influence spreads between users over the networks; secondly, top-k influential users have to be chosen. In this evaluation we consider (i) the Independent Cascade (IC) diffusion

model [13] (a stochastic model in which each active node has one attempt to activate its neighborhood), and (ii) IMM [37] algorithm (a two phase algorithm exploiting a set of estimation probabilistic techniques [41] to reduce the number of required sample).

To provide qualitative evaluation we use a recall measure to compare the obtained results with respect to a given ground truth, computed by using Monte Carlo method [20]. Formally, the recall measure is defined as:

$$R = \frac{|\hat{U} \cap \tilde{U}|}{|\hat{U}|}$$

where $\hat{U}$ is the set of influentials in the ground-truth and $\tilde{U}$ corresponding to the set of influentials obtained with respect to time constraints.

To provide this analysis we consider L-sized dataset and the same cluster configuration shown in section 7.1. Figure 9 shows the recall values on varying time constraints (sampled each 5 minutes). Since the proposed algorithm iteratively updates the interaction probability between two users, after an initial startup due to an absence of prior knowledge it provides good results with respect to the ground truth. In more details, with only 5-minutes processing the recall is higher than 60%. After 15 minutes the recall is always higher than 75%.

This analysis shows that both architectures properly support the OSN application taken into account.

8. Discussion and Conclusion

In this paper we have analyzed the performance of two state-of-art big data analytics architectures (Kappa and Lambda) when deployed onto a public-cloud PaaS. To achieve this goal we have considered (i) Apache Spark and Storm (providing the *de-facto* standard implementation for Kappa and Lambda, respectively); (ii) an implementation of the popular influence analysis task to generate the workload; (iii) Flickr YFCC100M big-data dataset as input; (iv) Microsoft

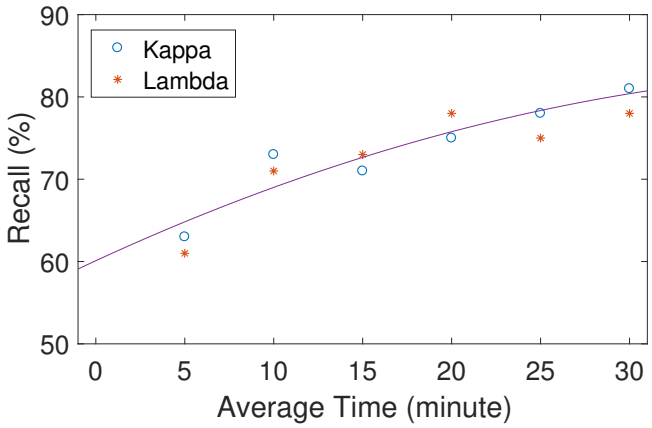


Figure 9: Recall evaluation.

Azure HDInsight PaaS as deployment environment (as it is provided by one of the market leaders and thus being a popular choice among customers).

Beyond the specific outcomes of this work, in this paper we proposed a detailed and reproducible methodology that can be adopted to evaluate big data architectures along different and complementary dimensions. Guided by the scientific literature and by known advantages carried by the cloud paradigm as well as potential criticalities related to the application of the reference task, we have identified three main evaluation dimensions: timeliness, cost, and outcome quality. According to the PaaS abstraction, we have set running time as the reference metric for evaluating timeliness, and have identified a number of factors under the direct control of cloud PaaS customers (i.e. number of deployed nodes and their characteristics in terms of computing resources).

Experimental campaigns have been designed to evaluate the impact of input-data volume, cluster size, VMs characteristics (also considering optimized deployments proposed by the provider), costs for deploying the architectures and carrying out the analyses, and the impact of time constraints on the quality of the outcomes. We believe that the methodology proposed and detailed in the paper can be a useful reference for further works aiming at investigating the performance of big data architectures also deployed on public-cloud Paas

other that Azure. Moreover, differently than other approaches proposed in the literature, the one adopted in this paper does not require accessing to provider-restricted information and thus can be adopted even by cloud customers (e.g., to evaluate the performance of the available services against the related costs). For instance, adopting the proposed methodology researchers and practitioners can compare PaaS services made available by different providers to support different big data distributed architectures, compare them, and evaluate the trade-off between performance and cost.

Beside the methodological contribution discussed above, focusing on OSNs and on the popular influence analysis task and leveraging state-of-art implementations, this paper analyzes in depth the performance of two state of the art big data architectures. Research outcomes provide useful insights to both practitioners involved in architecture design as well as to cloud customers willing to conveniently lease cloud resources. While results have been obtained leveraging a single cloud provider, most of the design factors taken into account (e.g., number of nodes per cluster, memory and CPU resources) are general enough to be meaningful for other providers or even for in-house deployments. This could not strictly apply for what concerns optimized configurations that may change across different providers. However, it is worth noting that most of the optimizations (such as those related to memory, computation, and disk) are made available by a growing number of market players. Moreover, obtained results can be easily generalized to other analysis tasks characterized by the same computational complexity. Therefore, notwithstanding some minor limitations, the results discussed below are general enough to be valid in scenarios wider that those specifically investigated (i.e., other tasks, other cloud providers, other architecture implementations). In the following, the main take-home messages and lesson learned from our analysis are discussed.

Both architectures disclose predictable performance. Experimental campaigns reported that in each specific scenario (identified by size of the cluster, type of VMs, and size of the input) performance variability e extremely low, with standard deviation (CoV) always lower than 52.93 s (0.13), in spite of the

random sampling done.

Both architectures provide good scalability with respect to the size of the dataset. Although the size of the dataset is the factor with the major impact on performance—being able to generate huge variation in terms of processing time—both architectures show sublinear scaling. Indeed, when providing in input a 100× larger dataset (100M instead 1M tuples) the average increase in terms of running time observed for Kappa (Lambda) is +2209% (+1940%), at most. Kappa suffers worst performance degradation on increasing input size than Lambda.

On the other hand, Kappa scales better when implementing the architectures with resource-richer nodes (vertical scaling). In other words, although both architectures significantly improve performance when enhancing VMs characteristics, the relative performance improvement observed for Kappa when enhancing architecture implementation is larger than that observed for Lambda (−30% for Lambda architecture versus −32% for Kappa). Analyzing horizontal-scaling we found that benefits are achieved when adding executor nodes, although the marginal improvement trend significantly decreases already with 6 executors.

More in general, Lambda outperforms Kappa. For what concerns the class of problem analyzed and considering results in the analogous scenarios, Lambda always provided better performance than Kappa deployments. The performance discrepancy between the two architectures varies according to the specific scenarios, and is as high as 26% in the case of we choose the VER-L configuration.

Regarding deployment costs, cheaper implementations in terms of hourly cost cause poor performance and require longer executions times for completing the analysis task, thus leading to higher overall cost. Cost analysis shows that high-performance deployments are associated to higher hourly cost. On the other hand, these same deployments may lead to shorter execution time, thus unveiling a positive impact on overall expenditure.

Finally, the last analysis shows that both architectures properly support the considered OSN application.

We believe that the insights provided in this paper are potentially useful to

both research and industry practitioners, interested in deploying big data analytics systems onto the cloud. Moreover, because of the variety of the analyses proposed results are helpful to both newcomers and experts of the field.

Acknowledgements

This work is partially funded by art. 11 DM 593/2000 for NM2 srl (Italy).

References

[1] Microsoft Azure website. `http://azure.microsoft.com`, December 2017.

[2] F. Amato, V. Moscato, A. Picariello, and G. Sperlí. Diffusion algorithms in multimedia social networks: a preliminary model. In *ASONAM '17: Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, pages 844–851, New York, NY, USA, 2017. ACM.

[3] Flora Amato, Vincenzo Moscato, Antonio Picariello, and Giancarlo Sperlí. Multimedia social network modeling: A proposal. In *Semantic Computing (ICSC), 2016 IEEE Tenth International Conference on*, pages 448–453. IEEE, 2016.

[4] Chaitanya Baru, Milind Bhandarkar, Carlo Curino, Manuel Danisch, Michael Frank, Bhaskar Gowda, Hans-Arno Jacobsen, Huang Jie, Dileep Kumar, Raghunath Nambiar, et al. Discussion of bigbench: a proposed industry standard performance benchmark for big data. In *Technology Conference on Performance Evaluation and Benchmarking*, pages 44–63. Springer, 2014.

[5] Pablo Basanta-Val. An efficient industrial big-data engine. *IEEE Transactions on Industrial Informatics*, 2017.

[6] Pablo Basanta-Val, Neil C Audsley, Andy J Wellings, Ian Gray, and Norberto Fernández-García. Architecting time-critical big-data systems. *IEEE Transactions on Big Data*, 2(4):310–324, 2016.

[7] Mihai Capotă, Tim Hegeman, Alexandru Iosup, Arnau Prat-Pérez, Orri Erling, and Peter Boncz. Graphalytics: A big data benchmark for graph-processing platforms. In *Proceedings of the GRADES'15*, page 7. ACM, 2015.

[8] Yanpei Chen, Sara Alspaugh, and Randy Katz. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *Proceedings of the VLDB Endowment*, 5(12):1802–1813, 2012.

[9] Yanpei Chen et al. We don't know enough to make a big data benchmark suite-an academia-industry view. *Proc. of WBDB*, 2012.

[10] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[11] Orri Erling, Alex Averbuch, Josep Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat, Minh-Duc Pham, and Peter Boncz. The ldbc social network benchmark: Interactive workload. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 619–630. ACM, 2015.

[12] Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. Bigbench: towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*, pages 1197–1208. ACM, 2013.

[13] Jacob Goldenberg, Barak Libai, and Eitan Muller. Talk of the network: A

complex systems look at the underlying process of word-of-mouth. *Marketing letters*, 12(3):211–223, 2001.

[14] Amit Goyal, Francesco Bonchi, and Laks VS Lakshmanan. Learning influence probabilities in social networks. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 241–250. ACM, 2010.

[15] M. Gribaudo, M. Iacono, and M. Kiran. A performance modeling framework for lambda architecture based applications. *Future Generation Computer Systems*, 2017. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2017.07.033. URL `http://www.sciencedirect.com/science/article/pii/S0167739X17315364`.

[16] Rui Han, Lizy Kurian John, and Jianfeng Zhan. Benchmarking big data systems: A review. *IEEE Transactions on Services Computing*, 2017.

[17] Paul DH Hines, Ian Dobson, and Pooya Rezaei. Cascading power outages propagate locally in an influence graph that is not the actual grid topology. *IEEE Transactions on Power Systems*, 32(2):958–967, 2017.

[18] Ziya Karakaya, Ali Yazici, and Mohammed Alayyoub. A comparison of stream processing frameworks. In *Computer and Applications (ICCA), 2017 International Conference on*, pages 1–12. IEEE, 2017.

[19] David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 504–513. ACM, 2000.

[20] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM, 2003.

[21] T. Kraska. Finding the needle in the big data systems haystack. *IEEE Internet Computing*, 17(1):84–86, Jan 2013. ISSN 1089-7801. doi: 10.1109/MIC.2013.10.

[22] Jay Kreps. Questioning the lambda architecture. *Online article, July*, 2014.

[23] Lydia Leong, Douglas Toombs, and Bob Gill. Magic quadrant for cloud infrastructure as a service, worldwide. `http://www.gartner.com/technology/reprints.do?id=1-2G2O5FC&ct=150519&st=sb`, 2015.

[24] Nathan Marz and James Warren. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2015. ISBN 1617290343, 9781617290343.

[25] Nathan Marz and James Warren. *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications Co., 2015.

[26] Zijian Ming, Chunjie Luo, Wanling Gao, Rui Han, Qiang Yang, Lei Wang, and Jianfeng Zhan. *BDGS: A Scalable Big Data Generator Suite in Big Data Benchmarking*, pages 138–154. Springer International Publishing, Cham, 2014. ISBN 978-3-319-10596-3. doi: 10.1007/978-3-319-10596-3_11. URL `https://doi.org/10.1007/978-3-319-10596-3_11`.

[27] Ralf Möller and Özgür Özcep. Implementation of the linear road benchmark on the basis of the real-time stream-processing system storm. *Hamburg University of Technology*, 2014.

[28] Duc T Nguyen and Jai E Jung. Real-time event detection for online behavioral analysis of big social data. *Future Generation Computer Systems*, 66:137–145, 2017.

[29] Keren Ouaknine, Michael Carey, and Scott Kirkpatrick. The pigmix benchmark on pig, mapreduce, and hpcc systems. In *Big Data (BigData congress), 2015 IEEE International Congress on*, pages 643–648. IEEE, 2015.

[30] Alex Pentland. *Social Physics: How social networks can make us smarter*. Penguin, 2015.

[31] Valerio Persico, Pietro Marchetta, Alessio Botta, and Antonio Pescapé. Measuring network throughput in the cloud: the case of amazon ec2. *Computer Networks*, 93:408–422, 2015.

[32] Valerio Persico, Pietro Marchetta, Alessio Botta, and Antonio Pescapé. On network throughput variability in microsoft azure cloud. In *Global Communications Conference (GLOBECOM), 2015 IEEE*, pages 1–6. IEEE, 2015.

[33] Valerio Persico, Domenico Grimaldi, Antonio Pescapé, Alessandro Salvi, and Stefania Santini. A fuzzy approach based on heterogeneous metrics for scaling out public clouds. *IEEE Transactions on Parallel and Distributed Systems*, 28(8):2117–2130, 2017.

[34] N Satra. Is 'Distributed'worth it? Benchmarking Apache Spark with Mesos. `http://www.cl.cam.ac.uk/~ey204/pubs/ACS/Mini_Projects/Neil_Spark.pdf`, 2015.

[35] Samneet Singh. Empirical evaluation and architecture design for big monitoring data analysis. *Concordia University*, 2016.

[36] Wei Tan, M Brian Blake, Iman Saleh, and Schahram Dustdar. Social-network-sourced big data analytics. *IEEE Internet Computing*, 17(5):62–69, 2013.

[37] Youze Tang, Yanchen Shi, and Xiaokui Xiao. Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1539–1554, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-2758-9. doi: 10.1145/2723372.2723734. URL `http://doi.acm.org/10.1145/2723372.2723734`.

[38] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: The

new data in multimedia research. *Communications of the ACM*, 59(2): 64–73, 2016.

[39] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, et al. Bigdatabench: A big data benchmark suite from internet services. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, pages 488–499. IEEE, 2014.

[40] Jinliang Wei, Jin Kyu Kim, and Garth A Gibson. Benchmarking apache spark with machine learning applications, 2016.

[41] David Williams. *Probability with martingales*. Cambridge university press, 1991.

[42] Baojun Zhou, Jie Li, Xiaoyan Wang, Yu Gu, Li Xu, Yongqiang Hu, and Lihua Zhu. Online internet traffic monitoring system using spark streaming. *Big Data Mining and Analytics*, 1(1):47–56, 2018.