Cenote: A Big Data Management and Analytics Infrastructure for the Web of Things

Kyriakos C. Chatzidimitriou, Michail D. Papamichail, Napoleon-Christos I. Oikonomou, Dimitrios Lampoudis, and Andreas L. Symeonidis
{kyrcha,mpapamic,noikon}@issel.ee.auth.gr,ldim@olympus.ee.auth.gr,asymeon@eng.auth.gr
Electrical and Computer Engineering Dept., Aristotle University of Thessaloniki
Thessaloniki, Greece

ABSTRACT

In the era of Big Data, Cloud Computing and Internet of Things, most of the existing, integrated solutions that attempt to solve their challenges are either proprietary, limit functionality to a predefined set of requirements, or hide the way data are stored and accessed. In this work we propose Cenote, an open source Big Data management and analytics infrastructure for the Web of Things that overcomes the above limitations. Cenote is built on component-based software engineering principles and provides an all-inclusive solution based on components that work well individually.

CCS CONCEPTS

• **Software and its engineering → Data flow architectures**.

KEYWORDS

web of things, internet of things, analytics, infrastructure, apache kafka, restful api, apache storm, cockroachdb

ACM Reference Format:
Kyriakos C. Chatzidimitriou, Michail D. Papamichail, Napoleon-Christos I. Oikonomou, Dimitrios Lampoudis, and Andreas L. Symeonidis. 2019. Cenote: A Big Data Management and Analytics Infrastructure for the Web of Things. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI '19), October 14–17, 2019, Thessaloniki, Greece.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3350546.3352531

1 INTRODUCTION

One of the most prominent intersections in the modern computing landscape is that of the Big Data, Cloud Computing and Internet of Things (IoT) disciplines (BCI). It is now common practice for sensing, real-world devices to send their data over a networked cloud computing system for information to further be processed. Processing usually implies analyzing the data, a task facilitated by some cloud computing environment and the intelligent - often autonomously made - decisions are propagated to the application at hand through actuating, real-world objects, closing the loop and providing value to the application chain. Besides the Value, the other four V's of Big Data are also present in these types of applications: Volume (scale), Velocity (speed), Veracity (certainty) and Variety (diversity). With the addition of Artificial Intelligence and Machine Learning algorithms, such cyber-physical systems are expected to cover the basic needs of all kinds of modern society organizations: business, finance, manufacturing, health and environment among others. Example applications include e-Health, smart cities, smart home, smart metering, video surveillance, smart mobility, environmental monitoring, smart logistics and more [14].

One of the main challenges between IoT and Cloud Computing is that of interoperability [15], due to competing standards and custom solutions. The Web of Things [16, 17] has been proposed as a way to mitigate this issue by mingling real-world things into the Web and at the same time taking advantage of the Web's merits. Moreover, on the integration of Big Data with Cloud Computing disciplines, typical challenges include: data storage and management, data transmission and curation, data processing and analysis and data privacy and security [20]. At the intersection of all these challenges, we propose Cenote, a Big Data Management System (BDMS) with analytics capabilities for the Web of Things providing (near) real-time analytics capacities to event streams coming in from any kind of web source. Cenote is a system that: (i) is open source, following a component-based development approach, (ii) can be considered as a general-scope, out-of-the-shelf, BDMS, supporting analytics out-of-the-box in many scenarios that involve event stream processing, (iii) combines both (near) real-time analytics and batch processing, and (iv) is deployed in a distributed and scalable manner.

2 CENOTE ARCHITECTURE

Cenote's architecture is depicted in Figure 1. Data transmission is split into two main flows: the write flow and the read flow. The write flow is concerned with writing data from "web-enabled things" to Cenote, while the read flow is concerned with returning answers to analytics-related API calls [5] coming from HTTP-enabled clients. Cenote uses the following components:

NGINX [10] is a popular, high performance load balancer, web server and reverse proxy.

MEN stack API server. MEN stack stands for a technology stack that is composed of MongoDB [9], Express web framework [7] and Node.js [11]. It is a very popular stack for building API servers.

Apache Kafka [1] is a distributed streaming platform that can publish and subscribe to streams of records (topics). Additionally it stores these streams of records using a fault-tolerant durable way.

Apache Storm [3] is an open source distributed real-time computation system. A Storm topology - a directed graph where each

WI '19, October 14–17, 2019, Thessaloniki, Greece

ACM ISBN 978-1-4503-6934-3/19/10…$15.00
https://doi.org/10.1145/3350546.3352531

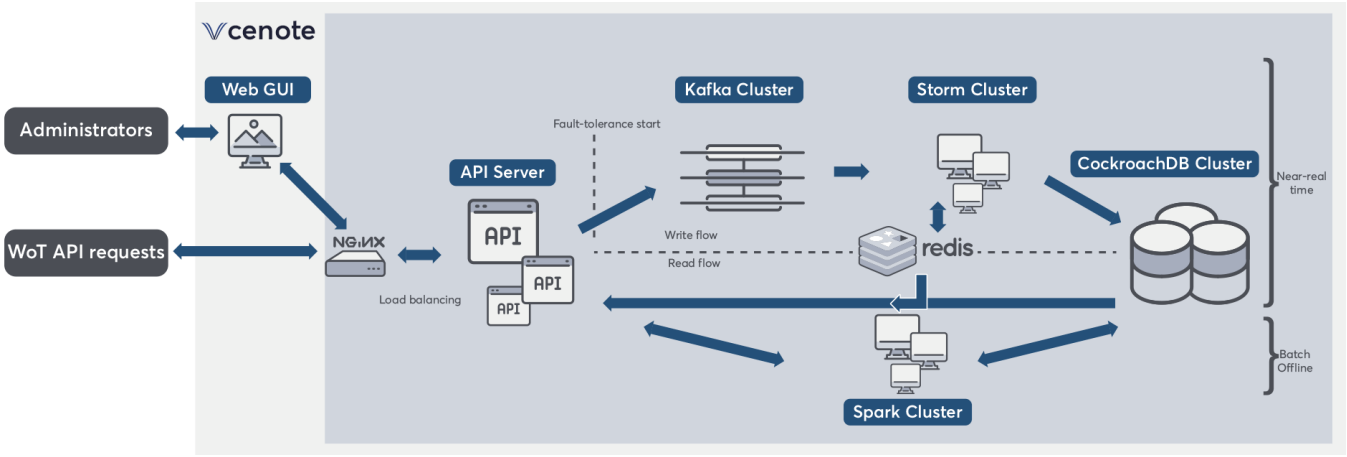

Figure 1: Cenote's architectural diagram depicting the components involved in both write and read flows.

vertex represents a computational element - consumes streams of data and can processes those streams in arbitrarily complex ways.

CockroachDB [6] is distributed CAP-consistent, SQL, ACID compliant database with automatic scaling, rebalancing and repairing mechanisms.

Apache Spark [2] is a unified analytics engine for large-scale data processing. We selected because of its merits as a batch processing engine.

Redis [12] is an open source, in-memory data structure store, used as a database, cache and message broker.

Apache Zookeeper [4] is used to coordinate clusters of components like Storm and Kafka.

2.1 Data modelling

The main data entity of Cenote is that of an event. Events are individual timestamped collections of data points. Each data point contains a set of properties and their values in a specific point in time. Similar events are organized in event collections. Events are modelled as `JSON` documents that do not have an enforced schema. An example is shown in Listing 1. The schemaless nature of the events offers easier extensibility in the expense that one must strive to store similar events in each event collection. From a technical perspective, each event collection is stored in a single CockroachDB table, with each column corresponding to a single JSON property.

Listing 1: JSON load example

```
{"cenote": {
    "created_at": "2012-12-14T20:24:01",
    "timestamp": "2012-12-14T20:24:01",
    "id": "asd9fadifjaqw9asdfasdf939"
  },"device": {
    "id": "1234567890abcdef",
    "model": "H09 Beta",
    "temperature": 29.5}}
```

2.2 Write flow

The write flow is found on the upper half of Figure 1. `HTTP` requests for writing data arrive in Cenote to the load balancer, which routes the requests in a round-robin fashion to the web API nodes. From there the requests are queued in Kafka. From this point on fault-tolerance starts. Storm Spouts consume messages from Kafka and transmit them to the processing Storm Bolts. Each processing Bolt is responsible for updating running statistics for outlier detection and writing the events to the data store.

2.3 Read flow

The read flow is found on the lower half of Figure 1. `HTTP GET` requests are received by Cenote that correspond to analytics queries. Whatever calculations can be handled by CockroachDB are handled by the database, to reduce data transfer volumes, and after some post-processing in the web servers the responses are sent to the clients. Cenote supports the following types of analytics queries for a given property, event collection and timeframe of reference: (i) average (ii) sum (iii) count unique (iv) count (v) maximum (vi) median (vii) minimum (viii) percentile (ix) select unique. In addition the API supports filtering and group-by capabilities for querying and extraction capabilities of large chunks of events for external processing.

2.3.1 Outlier Detection. Anomaly detection enhances the veracity of the data and may prove to be a highly valuable addition, especially in real-time mode of operations. This requirement imposes the need for an online, inside a streaming context, outlier detection algorithm. To this end we have examined a lot of different alternatives [21] and have picked the algorithm found in [13] that is based on Chebysev's inequality. This algorithm was selected due to: a) its simplicity, with the implication that it will not increase the latency between request-response in velocity terms, b) the ability to work without making any assumption on the distributions. One limitation is that we examined only algorithms that perform outlier detection in numerical values. As a threshold we chose the p–value to be 0.05.

In the write flow the algorithm calculates two pairs of running averages and variances of a numeric property based on [23] using

Lua scripting inside Redis. Listing 2 displays the script for the calculations. `KEYS[1]` contains the triplet n (count), m (mean) and $m2$ (squared distance from the mean) and `ARGV[1]` contains the new numerical value.

Listing 2: Running mean and variance calculation.

```
local aggregate = redis.call('get',KEYS[1])
local decode = cjson.decode(aggregate)
local n = decode['n']
n = n + 1
local m = decode['m']
local m2 = decode['m2']
local delta = ARGV[1] - m
m = m + delta/n
m2 = m2 + delta * (ARGV[1] - m)
decode['n'] = n
decode['m'] = m
decode['m2'] = m2
local encoded = cjson.encode(decode)
redis.call('set', KEYS[1], encoded)
```

In the write flow and for a given numeric property:

(1) update running mean and variance based on Listing 2
(2) if the numeric property is more extreme than the ODV (Outlier Detection Value) the property is an outlier
(3) else, the numeric property is not an outlier and update the 2nd pair of running average and variance.
(4) continue with storage as all the data are written to Cenote whether they are outliers or not.

The ODV lower (ODV_L) and upper (ODV_U) thresholds are calculated based on the equations below:

$$ODV_U = \mu + \frac{1}{\sqrt{p}} \times \sigma, ODV_L = \mu - \frac{1}{\sqrt{p}} \times \sigma \quad (1)$$

In the read flow, in all API calls there will be a query parameter named `outliers`. The default value will be `include`, but there will be two other values: `exclude` and `only`. The value `include` will include all outliers in the queries, `exclude` will exclude them and `only` will only use the outliers. In the write flow the ODV values are based on the 2nd pair of running averages and variance. In the read flow a lower p−value should be used, i.e. 0.01.

3 EXPERIMENTATION

For experimentation Cenote was installed on a cluster that consists of one master node and three workers (Table 1). We evaluated Cenote on two axes: (i) measure the scalability of the system in terms of handling the incoming traffic (web part) and serving the incoming requests (write pipeline), (ii) quantify the capabilities of our system while performing certain analytics operations such as outlier detection and summation computations.

3.1 Incoming traffic experiment

The main objective of the first experiment is to measure the performance efficiency of Cenote in handling a big amount of incoming traffic. Table 2 presents the results.

Table 1: Cluster Specifications

Node	CPU	RAM	Memory
master	2x vCPUs	9GB	100GB HDD
worker	2x Intel Xeon 4114	64GB	1.6TB SSD
worker	2x Intel Xeon 4114	64GB	1.6TB SSD
worker	2x Intel E5-2630 v3	9GB	1.8TB HDD

3.2 Analytics experiments

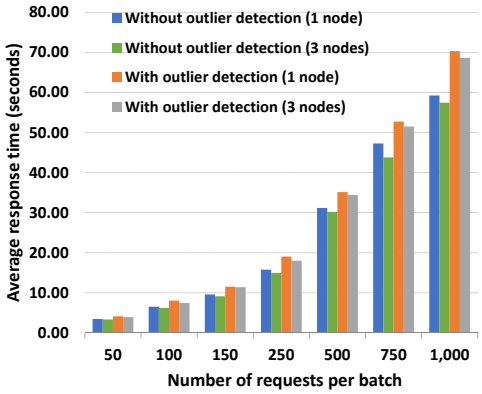

Figure 2: Comparative analysis results for finding maximum operation

We conducted two analytics experiments. Figure 2 illustrates the results for the operation of finding the maximum on a table that contains 450,000 events. As shown in the figure, Cenote is able to serve 250 requests in less than 20 seconds. It is worth noting that the impact of enabling outlier detection remains constant and causes an overhead which lies in the interval from 14% to 16% of the response time. This is expected as the operations are performed inside the database and thus this is what determines the execution time. This fact is demonstrated in Table 3, where we compared the measured response times between having 2 and 3 CockroachDB instances. The results showed that increasing the number of DB instances improved the average response time in all cases.

4 RELATED WORK

When building a BDMS, two of the most influential architectures are known as lambda and kappa architectures. In the lambda architecture [19], a series of layers (batch, serving, speed) is built in order to satisfy Big Data properties and to facilitate the next layer to build upon it. One of the problems of the lambda architecture is that the code needs to be maintained in two complex distributed systems and produce the same results [18]. This has led to the kappa architecture that uses only a stream processing engine to handle the full problem. Cenote has acquired features from both architectures. The data storage is immutable and constantly growing, there is only one codebase and we have targeted more specifically (near) real-time analytics.

Cenote is heavily influenced by the commercial analytics service of keen.io [8]. Keen.io receives events in the form of JSON messages

Table 2: Incoming traffic benchmarks. The scaling factor refers to API servers and Storm supervisors. In the last row with 3 Storm supervisors the system has reached its limit, degrading its performance. At this traffic levels, the cluster should be upgraded.

Incoming throughput (requests/second)	Scaling factor	Av. Response Rate API (req/sec)	Av. Response Rate Storm (req/sec)
25,000	1	154	534
25,000	2	161	1,273
25,000	3	362	754

Table 3: Database scaling results

Req	Response Time (sec)		Diff
	2 DB instances	3 DB instances	
50	4.40	3.34	-31.61%
100	6.36	6.23	-2.06%
150	9.51	9.12	-4.26%
250	16.18	14.97	-8.09%
750	47.09	43.78	-7.57%
1,000	62.64	57.43	-9.07%

that can be analyzed and visualized for taking informative decisions. The main difference between keen.io and Cenote is that Cenote is an open source BDMS. It includes also a batch processing component along with an online outlier detection algorithm.

In [22] the authors use the combination of the components Kafka, Storm and MongoDB to store the sensor data from the manufacturing process. Density-Based Spatial Clustering of Applications with Noise (DBSCAN)-based outlier detection and Random Forest classification were used to remove outlier sensor data and provide fault detection during the manufacturing process, respectively. The main difference is that Cenote is a general-case BDMS that can work in multiple domains, organizations and projects, while its outlier detection algorithm is online.

5 CONCLUSIONS AND FUTURE WORK

In this work we proposed Cenote, an open source BDMS focusing on Big Data management and (near) real-time analytics for the Web of Things. We have extensively tested both the design principles and the cluster deployment, where our production cluster was able to handle $1K$ requests per second end-to-end, meaning from receiving to persisting the request. In addition, it is capable of handling much bigger loads - we have tested it with $25K$ requests/second - in its current state, with increased latency. The querying operations were found to be also scalable, while the selection of the online outlier detection algorithm was proven to have a small impact on the response times. Our future plans include the ad-hoc fusion of batch and real-time querying.

ACKNOWLEDGMENTS

This research has been co–financed by the European Regional Development Fund of the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH–CREATE–INNOVATE (project code:T1EDK-04045).

REFERENCES

[1] 2019. Apache Kafka. Retrieved July 22, 2019 from https://kafka.apache.org
[2] 2019. Apache Spark. Retrieved July 22, 2019 from https://spark.apache.org/
[3] 2019. Apache Storm. Retrieved July 22, 2019 from https://storm.apache.org/
[4] 2019. Apache Zookeper. Retrieved July 22, 2019 from https://zookeeper.apache.org/
[5] 2019. Cenote documentation. Retrieved July 22, 2019 from http://issel.ee.auth.gr/cenote
[6] 2019. CockroachDB. Retrieved July 22, 2019 from https://www.cockroachlabs.com/
[7] 2019. Express. Retrieved July 22, 2019 from https://expressjs.com/
[8] 2019. keen.io. Retrieved July 22, 2019 from https://keen.io/
[9] 2019. MongoDB. Retrieved July 22, 2019 from https://www.mongodb.com/
[10] 2019. NGINX. Retrieved July 22, 2019 from https://nginx.org/
[11] 2019. node.js. Retrieved July 22, 2019 from https://nodejs.org
[12] 2019. Redis. Retrieved July 22, 2019 from https://redis.io/
[13] B. G. Amidan, T. A. Ferryman, and S. K. Cooley. 2005. Data outlier detection using the Chebyshev theorem. In *2005 IEEE Aerospace Conference*. 3814–3819.
[14] Alessio Botta, Walter De Donato, Valerio Persico, and Antonio Pescap?? 2016. Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems* 56 (2016), 684–700.
[15] Manuel Diaz, Cristian Martin, and Bartolome Rubio. 2016. State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. *Journal of Network and Computer Applications* 67 (2016), 99–117.
[16] Dominique Guinard and Vlad Trifa. 2009. Towards the Web of Things: Web Mashups for Embedded Devices. In *WWW (International World Wide Web Conferences), Enterprise Mashups and Lightweight Composition on the Web (MEM 2009) Workshop*. Madrid, Spain.
[17] Dominique Guinard and Vlad Trifa. 2015. *Building the Web of Things*. Manning.
[18] Jay Kreps. 2014. Questioning the Lambda Architecture. Retrieved July 22, 2019 from https://www.oreilly.com/ideas/questioning-the-lambda-architecture
[19] Nathan Marz and James Warren. 2013. *Big Data: Principles and best practices of scalable realtime data systems*. Manning.
[20] Georgios Skourletopoulos, Constandinos X Mavromoustakis, George Mastorakis, Jordi Mongay Batalla, Ciprian Dobre, Spyros Panagiotakis, and Evangelos Pallis. 2017. Big Data and Cloud Computing: A Survey of the State-of-the-Art and Research Challenges. 22 (2017).
[21] Imen Souiden, Zaki Brahmi, and Hajer Toumi. 2017. A Survey on Outlier Detection in the Context of Stream Mining: Review of Existing Approaches and Recommadations. In *Intelligent Systems Design and Applications*, Ana Maria Madureira, Ajith Abraham, Dorabela Gamboa, and Paulo Novais (Eds.). Springer International Publishing, Cham, 372–383.
[22] Muhammad Syafrudin, Ganjar Alfian, Norma Latif Fitriyani, and Jongtae Rhee. 2018. Performance Analysis of IoT-Based Sensor, Big Data Processing, and Machine Learning Model for Real-Time Monitoring System in Automotive Manufacturing. *Sensors* 18, 9 (2018).
[23] B. P. Welford. 1962. Note on a Method for Calculating Corrected Sums of Squares and Products. *Technometrics* 4, 3 (1962), 419–420.