Analysing dependability and performance of a real-world Elastic Search application

Malika Bendechache*, Ivanovitch Silva†, Guto Leoni Santos‡, Luiz Affonso Guedes†, Sergej Svorobej*, Manuel Noya Mario§, M. Eduardo Ares§, James Byrne*, Patricia Takako Endo*¶ and Theo Lynn*
*Dublin City University (DCU), Dublin, Ireland
Email: {malika.bendechache, sergej.svorobej, james.byrne, theo.lynn}@dcu.ie
†Universidade Federal do Rio Grande do Norte, Natal, Brazil
Email: ivan@imd.ufrn.br, affonso@dca.ufrn.br
‡Univesidade Federal de Pernambuco, Recife, Brazil
Email: gls4@cin.ufpe.br
§ Linknovate Science SL, Santiago de Compostela, Spain
Email: manuel@linknovate.com
¶Universidade de Pernambuco, Recife, Brazil
Email: patricia.endo@upe.br

***Abstract*—Increased complexity in IT, big data, and advanced analytical techniques are some of the trends driving demand for more sophisticated and scalable search technology. Despite Quality of Service (QoS) being a critical success factor in most enterprise software service offerings, it is often not a generic component of the enterprise search software stack. In this paper, we explore enterprise search engine dependability and performance using a real-world company architecture and associated data sourced from an ElasticSearch implementation on Linknovate.com. We propose a Fault Tree model to assess the availability and reliability of the Linknovate.com architecture. The results of the Fault Tree model are fed into a Stochastic Petri Net (SPN) model to analyze how failures and redundancy impact application performance of the use case system. *Availability* and *MTTF* were used to evaluate the reliability and throughput was used to evaluate the performance of the target system. The best results for all three metrics were returned in scenarios with high levels of redundancy.**

I. Introduction

Search engines are the most common way for users to source information on the Internet. In the UK, search engines are used by 94% of Internet adult users, by far the most popular source for information search [1]. In January 2019, nearly 10 billion search queries were processed by Google in the US alone [2]. While the volume of both searches and content is continually on the increase, delays in search results can lead to user frustration and result in loss of revenue [3]. At the same time, search engine providers are providing increasingly complex search functionality including voice search, predictive search, and object detection-based search among others increasingly based on varying degrees of artificial intelligence. To meet these demands, search engine providers rely on the efficient provision, scaling, and optimization of distributed compute infrastructure at hyper-scale [4].

In this paper, we examine the quality of service (QoS) characteristics of an enterprise data discovery service built on the ElasticSearch search engine. ElasticSearch (ES) [5] is a popular open source search engine designed to be distributed, scalable, and capable of near real-time information retrieval [6]. The analysis of the ElasticSearch performance was carried out using a real-world search service deployment used by Linknovate.com. Linknovate.com generates insight for its clients through the aggregation of large volumes of research and scientific data. Since 2017, they have indexed over 20 million documents, 30 million expert profiles, 2 million entity profiles, and more than 200 million innovation topics. Linknovate.com serves enterprise users located throughout the world who expect relevant information to be identified within a fraction of a second. To meet their end-user demand for real-time query response while processing large volumes of data requires an understanding of QoS dependencies through continuous monitoring, analysis, and remediation.

This paper has two main objectives:

- To propose a Fault Tree model for enterprise search engine reliability and availability analysis, and
- To integrate the Fault Tree model with a Stochastic Petri Net (SPN) model to analyze how failures and redundancy impact application performance.

The result of our work produces a general set of QoS attributes for a given service which can be further used to evaluate and optimize other large distributed search application services.

The remainder of this paper is organized as follows. Section II introduces background concepts and definitions of reliability and availability as dependability measures. Section III describes the modeling and analysis of the dependability of Linknovate.com as a representative ElasticSearch application. Next, the performance model for the Linknovate.com architecture is presented taking into account the dependability model defined in the previous section. Section V briefly summarizes selected related works. The paper concludes with a summary of the paper and a discussion of future work in Section VI.

II. Background Concepts

System dependability can be essentially described by two key measures: reliability and availability [7].

A. Reliability

The reliability $R(t)$ of a system is the probability of a failure not occurring in the interval [0,*t*) [7]. Suppose, without losing generality, that a system has two states [8], *working* and *failed*. The transition from an operating state to a faulty state is triggered by the *failure rate function* $\lambda(t)$. Now, consider T as the *time to failure* of a system component. If T is a random variable defined by a Cumulative Distribution Function (CDF) $F(t)$, it follows that reliability is given by:

$$R(t) = P(T > t) = 1 - F(t) \tag{1}$$

It can be proved that $R(t)$ and $\lambda(t)$, which describes the instantaneous failure rate of a system component, are associated as given by the Equation 2 [9]:

$$R(t) = exp\left(-\int_0^t \lambda(u)du\right) \tag{2}$$

Another metric related to $R(t)$ is the *Mean Time To Failure* (MTTF). This measure is defined as the expected time $E(t)$ elapsed until the occurrence of the (first) system failure [7]. Formally, MTTF is given by [9]:

$$MTTF = E(t) = \int_0^\infty tf(t)dt \tag{3}$$

B. Availability

The availability $A(t)$ of a system is defined as the probability of the system being operational at a given instant t [7]. Like reliability, availability can also be described as a model having two states - working (available) and faulty (unavailable). However, after the occurrence of a system fault, this new model may be repaired (maintenance). The transition from a faulty state to an operating state is induced by the *repair rate function* $\mu(t)$.

Availability can be obtained in the same way as reliability, by replacing each event for the failure and repair rate functions ($\lambda(t)$ and $\mu(t)$ respectively). Assuming that $\lambda(t)$ and $\mu(t)$ have constant values, then it can be proved that $A(t)$ is calculated as following [9]:

$$A(t) = \frac{\mu}{\mu + \lambda} + \frac{\lambda}{\mu + \lambda} e^{-(\lambda+\mu)t} \tag{4}$$

Assuming that *n* periods occurred (with *working* and *failed* states), the steady-state availability A_∞ can be outlined by Equation 5 [9]. In this case, MTTF $= 1/\lambda$ and MTTR (*Mean Time to Repair*) $= 1/\mu$.

$$A_\infty = \frac{MTTF}{MTTF + MTTR} \tag{5}$$

C. Fault Tree Analysis

A fault tree is a graphical paradigm applied to dependability analysis. It is a tree structure model representing a sequence of components failures that may cause a system shutdown. The model uses a tree structure consisting of events and logic gates. The events represent the normal conditions as well as system failures (components failures, environmental conditions, human errors, and many others) and use boolean logic i.e. they occur or do not occur. Cause-effect relationships between events are represented by logic gates. The most important event is the output of a fault tree, the TOP event, which represents a system failure. It is important to note that a gate input can be a single event or a combination of events originated from the output of another logic gate [10]. Furthermore, several types of gates available in graphic representation e.g. *and*, *or* and *k-out-of-n* gates.

Fault Tree Analysis (FTA) is an effective technique to evaluate system reliability and availability, both qualitatively and quantitatively [11]. The utility of FTA for qualitative analysis depends on the system stage. The development stage aims to identify potential problems that can lead to failures. In the commissioning stage, the FTA technique can be used to identify the causes of failure. Conversely, during the quantitative analysis process, the objective is to evaluate the probability of the occurrence of the TOP event. The main advantages of FTA are the execution of an intuitive procedure to describe the events that lead to a system fault, and the minimization of the state space explosion problem, which is very common when modeling large systems [12].

The evaluation of a fault tree is performed by calculating the TOP event (system failure condition) probability based on basic event probabilities. This calculation is conducted differently for each type of logic gate. Assuming n independent inputs/events, the occurrence of the event i is described by its cumulative distribution function $F_i(t)$. The logic gate outputs are indicated in Figure 1. When an *and* gate is used, the failure condition is only enabled when all inputs/events occur. On the other hand, the output of an *or* gate is enabled when at least one of its inputs/events is active. Finally, if a *k-out-of-n* (*KooN*) gate is used, the output is enabled if at least k inputs/events occurred.

The equations presented in Figure 1 illustrate a fault tree without repeated events. When repeated events occur, the equations are invalidated, hence, a different technique such as *Sum of Disjoint Products* (SDP) may be applied [13]. The SDP method can be efficiently applied to fault trees with repeating events and it is easily automated and computationally efficient. The basic idea is to find a boolean function $\phi(x)$ describing the system fault condition (TOP event) and transform into a different function whose individual terms are mutually exclusive. According to [10], this structural boolean function $\phi(x)$ is given by:

$$\phi(x) = \begin{cases} 1 & \text{system failed} \\ 0 & \text{system is working} \end{cases} \tag{6}$$

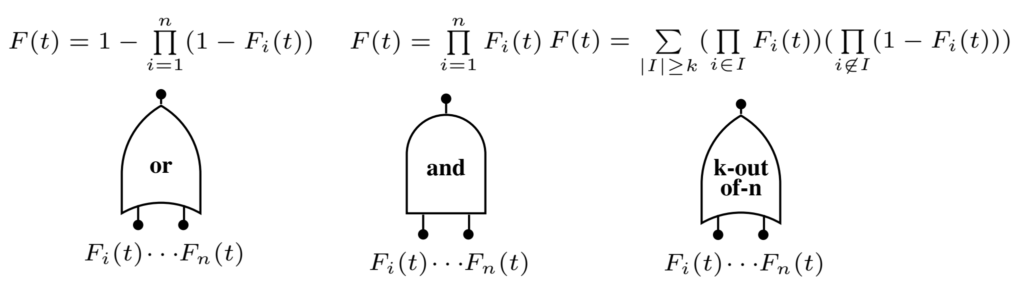
$F(t) = 1 - \prod_{i=1}^{n} (1 - F_i(t))$
$F(t) = \prod_{i=1}^{n} F_i(t)$
$F(t) = \sum_{|I| \geq k} (\prod_{i \in I} F_i(t))(\prod_{i \notin I} (1 - F_i(t)))$
or
and
k-out of-n
$F_i(t) \cdots F_n(t)$
$F_i(t) \cdots F_n(t)$
$F_i(t) \cdots F_n(t)$

Fig. 1. Cumulative distribution function $F(t)$ for the gate output (*and, or, k-out-of-n*).

where x is defined as the *state vector*, $x = (x_1, x_2, ..., x_n)$. Each element x_i is a boolean variable, that represents the component state *i*.

D. Stochastic Petri Nets

Petri Nets are a state-based mathematical formalism to model several kinds of systems with different behaviors such as concurrency, synchronization, and communication mechanisms [14]. A generic Petri Net can be defined as a tuple $PN = (P, T, F, M_0)$ [15], where:

- *P* is a finite set of places;
- *T* is a finite set of transitions;
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of flow relation, and
- $M_0 \in P \to \mathbb{N}_0$ is a set of tokens assigned into places which defines an initial marking.

Stochastic Petri Nets (SPN) are a variation of the traditional Petri Nets that assign a delay to the time transitions, and this delay follows a probability distribution function. With this, it is possible to derive an equivalent Continuous Time Markov Chains (CTMC) in order to solve the SPN model analytically [16]. In other words, the steady-state/transient analysis of the CTMC can be used to get steady-state/transient SPN measures efficiently. However, for the conversion to be possible, the time transitions should have a probability distribution which has a memoryless property e.g. the exponential distribution [17].

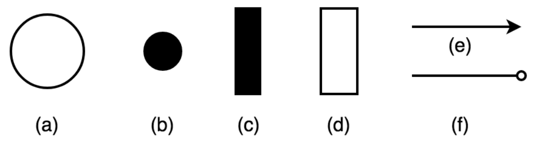

Fig. 2. Main SPN components

Figure 2 illustrates the main Petri Net components [14]. The places (Figure 2(a)) represent the local state variables, while the tokens (Figure 2(b)) may reside at the places; a set of tokens assigned to the places represents a state of the SPN. The transitions represent the events, actions, and activities that may occur in the system. The immediate transition (Figure 2(c)) fires after been enabled, while the timed transition (Figure 2(d)) fires following a delay defined by a distribution function. The arcs connect the places to transitions and vice-versa. There are two types of arcs: directed arcs (Figure 2(e)), which define the normal flow of the tokens; and inhibitor arcs (Figure 2(f)), which disable the transition if there are tokens present at the origin place [16].

Guard functions are expressions that can be used to represent conditions that must be satisfied to enable transitions to fire [18]. The use of these functions can simplify the modeling since arcs do not need to be added. For instance, considering that a transition will be enabled to fire when there are 20 tokens in a specific place, this condition can be represented as guard functions that check the number of tokens in this place and enable the transition when the condition is satisfied.

III. Modeling an ElasticSearch application

As described previously, the target application in this paper is related to an enterprise search engine based on ElasticSearch [19], [20]. As a use case, we consider Linknovate.com, a real-world company with a distributed data discovery engine whose objective is to search among large amounts of heterogeneous data to identify key institutions for a particular technology request. To achieve this objective, the Linknovate.com platform relies on different mechanisms, but the most crucial being distributed inverted-indices optimized for efficient searching, which stores the vast majority of the data.

The basic Linknovate.com platform (Figure 3) is deployed on the Microsoft Azure cloud and consists of *(a)* a web server, *(b)* an ElasticSearch (ES) client node, and *(c)* data nodes. To be able to offer the structured information available through the data nodes, it is necessary to perform extensive offline data processing of the new incoming raw data. This prior work is carried out on servers managed by Linknovate.com and processed results are moved to the cloud (Microsoft Azure) on a daily basis.

The ES client node and the data nodes (that compose the ES Cluster) are responsible for processing the requests, while the web server is where users unleash several internal queries over application indices, retrieving the data to be shown in the User Interface (UI).

A. Shard Distribution and Multi-Zones Environment

Data is distributed in an ES cluster over multiple partition units referred to as shards. Indices are used to link data to shards. The latter is stored in data nodes. If a shard is lost, the information related to that shard is also lost. Thus, at least one replica for each shard is typically configured in an ES cluster.

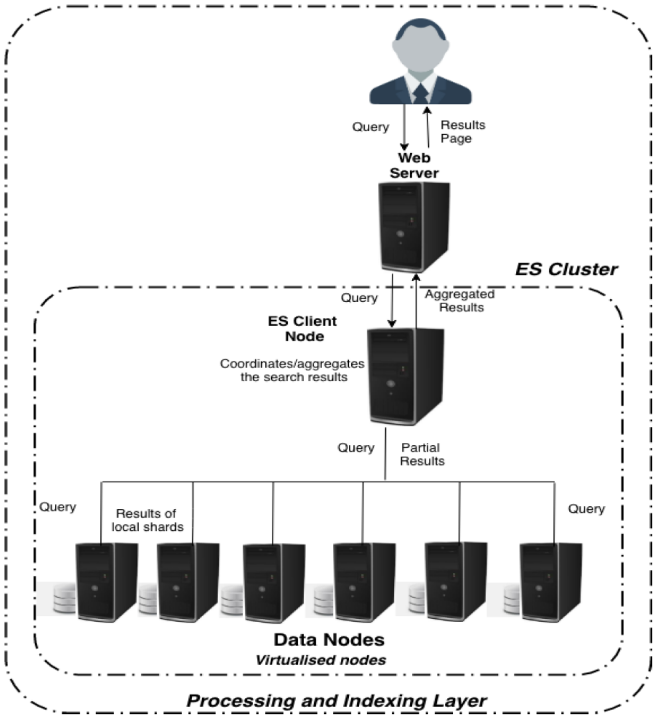

Fig. 3. An overview of the search engine architecture.

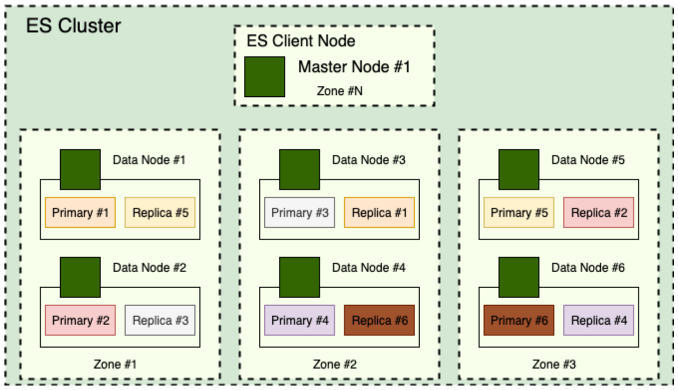

Fig. 4. Shard distribution considering one replica for each indices and three zones.

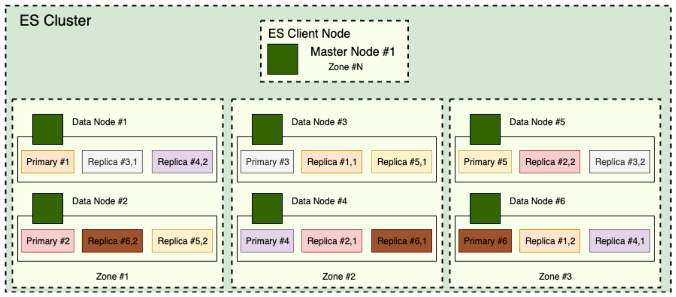

Fig. 5. Shard distribution considering two replicas for each indices and three zones.

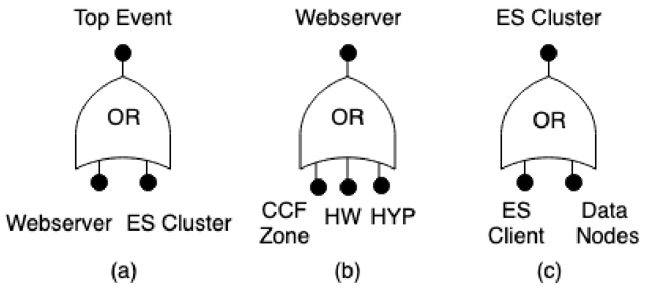

Fig. 6. (a) TOP event of search engine architecture. (b) webserver and (c) ES cluster models.

As the architecture (Figure 3) is deployed in a cloud environment, a multi-zone configuration can be used to meet reliability and availability requirements. Multi-zoning has the potential to isolate zones in case of local failures and thus reduce the consequences of that problem to the boundaries of that zone. Notwithstanding this, multi-zones can suffer from common cause failures (CCF); failures caused by an external event that causes all functioning components of that zone to fail at the same time.

Shard and multi-zones can be used to provide resilient scenarios. The master node in an ES cluster can distribute a primary shard and associated replica shards so that no replica shard is in the same zone of its corresponding primary shard. The ES cluster of Figure 3 was modeled using three zones with one and two replica shards according to Figures 4 and 5 respectively.

In Figure 4, a shard is considered in a faulty state if two data nodes fail; on the other hand, in Figure 5, a shard is regarded in a defective state if three data nodes fail.

B. Reliability and Availability Model

The modeling of a search engine architecture into a fault tree begins with the definition of the TOP event. In this case, we can assume that the system architecture (Figure 3) comprises two main components, the *web server* and the *ES cluster*. Thus, the system fails if the *web server* or the *ES cluster* fails. In this case, a *gate OR* is used to describe this scenario as per Figure 6(a).

Challenges emerge when it is necessary to map the failure events in the *web server* and the *ES cluster*. According to [21], reliability and availability evaluation for virtual and physical machines can be estimated from hardware (*HW*) and hypervisor (*HYP*) failures. In order to model CCF failures for each zone, an event named *CCF Zone* is introduced. The result is that the *web server* is mapped into a *gate OR* of three inputs, meaning that the *web server* fails if its hardware fails or its hypervisor fails, or if there is a common cause failure in the zone where the web server is situated (Figure 6(b)).

The modeling of the *ES cluster* is more complex. It comprises an *ES client* and a set of *data nodes*. This model is described in Figure 6(c). The *ES client* has a major role in the architecture proposed in Figure 3. It serves as the coordinator node of the *ES cluster*. In an optimistic configuration, the *ES client* can be deployed without redundancy (Figure 7(a)). In this case, the model is similar to that used for the *web server*

(Figure 6(b)). As discussed previously, the use of replicas can be an effective approach to improve the overall reliability of the system. In such a scenario, the *ES client* fails if the primary *ES client* fails and all replicas also fail. This model is described in Figure 7(b). Note that each replica is placed in different zones than the primary *ES client* (indices i and j).

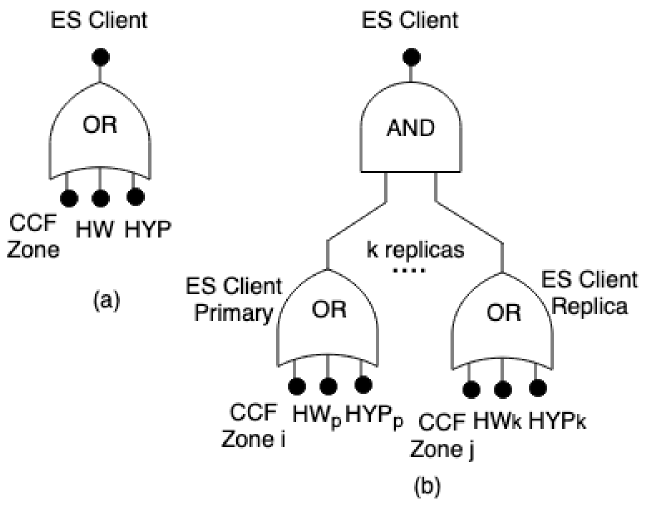

Fig. 7. (a) ES client without redundancy and (b) redundant ES client configuration with k replicas and a one primary (p) device.

Data nodes are other important components in the proposed architecture. In Figures 4 and 5, data nodes are configured based on the number of replicas used in each shard. A *gate kooN* is used to modeling this behavior in each scenario. When one replica is adopted (Figure 8(a)), failure only occurs when any combination of two nodes (from six nodes available) fails (*2oo6*). However, when two replicas are adopted (Figure 8(b)), at least three nodes need to fail for the data nodes array fails. Thus, a *gate 3oo6* must be configured.

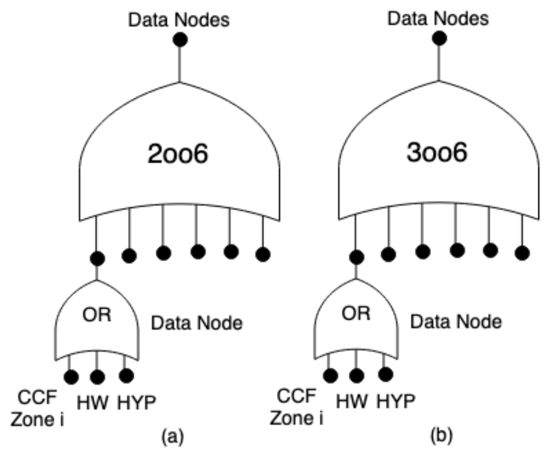

Fig. 8. (a) data nodes using one replica for each shard and (b) data nodes using two replicas for each shard.

C. Assumptions and MTTF Calculation

Our main objective is to calculate the Mean Time to Failure (MTTF) of the system since this metric is fundamental for the reliability and performance models proposed in this work. The main assumptions considered are:

- **Device Resources**: the components of the system have different resources. Specifically, the web server and data nodes are virtual machines with 28GB RAM and 8 CPUs. The ES Client is a more powerful device with 112GB RAM and 16 CPUs. Storage resources were not considered in the model for simplification [20].
- **Failure rate**: we assume that device failures occur at a constant rate (i.e. the exponential distribution). The failure rate of the devices is unknown however we use approximate values that we set based on the methodology described in [20], where the authors described and utilised ElasticSearch in optimised fashion to mine interesting patterns from log files. Zone disruption (CCF) is a rare event and thus it was configured to an MTTF of 20 years ($\lambda \cong$ 5.7077e–6 $hours^{-1}$). Hardware failure rate (HW) was set to an MTTF of 10 years ($\lambda \cong$ 1.1415e–5 $hours^{-1}$) for data nodes and web server, and 5 years ($\lambda \cong$ 2.2831e–5 $hours^{-1}$) for the ES Client. Lastly, the hypervisor failure rate was configured to an MTTF (hours) of 8 years ($\lambda \cong$ 1.4269e–5 $hours^{-1}$) for all devices.
- **Repair rate**: the architecture's devices can be repaired after a failure. After a repair, the device is considered as new. We consider that repair processes are independent and that the number of repairmen (i.e. the number of repair actions) is not limited. The time necessary to repair a device is characterized by a repair distribution. This distribution is defined in an analogous way to the failure distribution discussed previously. Based on [22], the repair rates were approximated for: $\mu_{CCFZone} = 1/72h^{-1}, \mu_{HW} = 1/24h^{-1}, \mu_{HY} = 1/0.5h^{-1}$.
- **Scenarios**: Six evaluation scenarios were considered based on the combination of shard distribution with one and two replicas, and redundancy configurations for the ES Cluster (no spares, one and two replicas). Table I summarizes the six scenarios considered in this study.

TABLE I
DIFFERENT SCENARIOS STUDIED BASED ON SHARD DISTRIBUTIONS AND REDUNDANCY ASPECTS OF THE ES CLIENT.

Scenario	Shard Distribution	ES Client
I	1 replica	No replica
II	2 replicas	No replica
III	1 replica	1 replica
IV	2 replicas	1 replica
V	1 replica	2 replicas
VI	2 replicas	2 replicas

The MTTF evaluation of the defined scenarios is described in Figure 9. As expected, the best results were found in scenarios with a high level of redundancy (IV and VI). Note that scenario VI is only 7.86% better than scenario IV for the MTTF metric. The result of the *availability* metric is approximately equal for scenario IV and VI. Thus, it is not always beneficial to have more replicas.

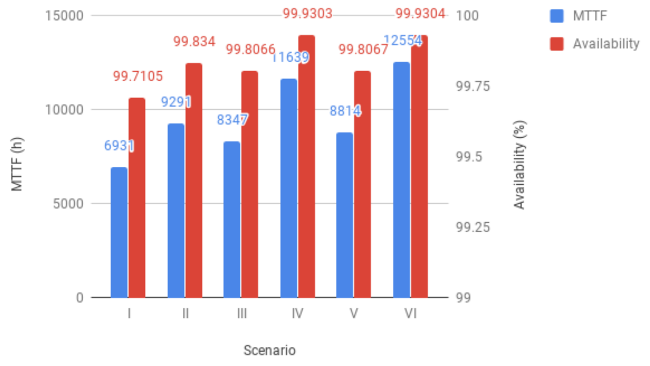

Fig. 9. MTTF and *availability* evaluation across six scenarios.

IV. Performance model

The reliability results from the previous section are used to feed a performance model and analyze how failures and redundancy impact on the ElasticSearch system performance. We assume that the system follows an M/M/1/K queue, where the arrival time rate, λ, is a Poisson process, and the service time, μ, follows an exponential distribution. To find λ and μ, we analyzed a real trace provided by Linknovate.com composed of 3,577,146 requests, from August 2017 to November 2017. Based on this data set, we found $\lambda = 0.4689$ requests/second, and we applied the *k-means* clustering to partition our data into four clusters (classes of requests) based on the request time, as shown in Table II.

TABLE II
Class of requests, service time and probability of having each class of request.

Request class	Service time (in seconds)	Probability (%)
A	0.2391	54.17
B	0.5833	30.50
C	1.0269	10.03
D	1.7414	5.29

Given the λ, μ, and the probability of each service time class, we implemented the SPN model as shown in Figure 10. When there is one token at place *P1*, the system can receive requests. The transition *ET1* represents the arrival of requests in the system, and when it fires, one token is consumed from *P1* and produced at place *P2*. The transition *ET1* receives the mean time between requests, which is $1/\lambda$. In this state, the immediate transitions *IT1* and *IT2* are enable to fire. The transition *IT2* fires according to the probability of the service time class (P_c); while the transition *IT1* fires with probability equals to $1-P_c$, representing the other classes of service time.

Therefore, if we are simulating the service with class A (e.g. with probability equals to 54.17%), the transition *IT2* will fire according to this probability, representing the requests for service class A. On the other hand, the transition for other services do not need to be considered as we are only evaluating class A. Considering the example, the probability of requests arrival for other services is 45.83%, in this case, the transition *IT1* will fire following this probability.

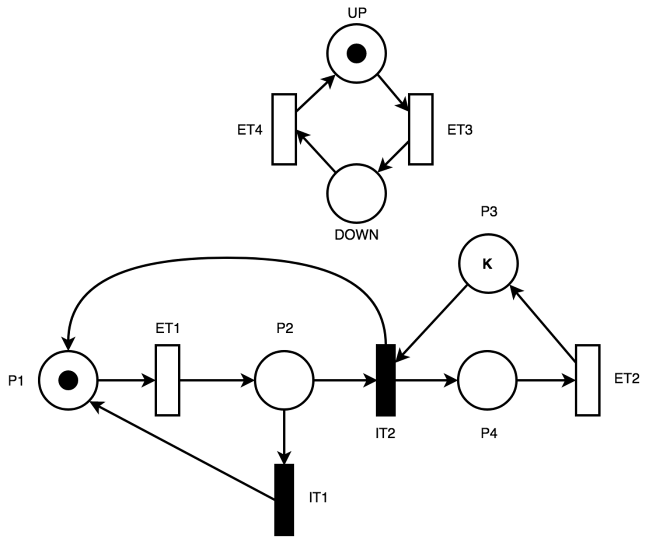

Fig. 10. Performance model based on the *availability* results presented in Figure9

If the transition *IT1* fires, one token is consumed from the place *P2* and one token is produced at *P1* again, enabling the arrival of new requests. On the other hand, if the transition *IT2* fires, the request will be processed by the system. Thus, one token is consumed from the *P2* and *P3* (decreasing a token from the total system capacity, K), and produced at *P4* and *P1* (enabling new requests to arrive). At this point, the transition *ET2* is enabled to fire following the class of service time specified ($1/\mu$), and when it fires, one token is consumed from the *P4* and the other is produced at the *P3*, regenerating the capacity of the system.

As we are interested in evaluating the impact of the availability on system performance, we integrate the results provided by the Fault Tree model with our SPN model. To represent when the system is available, we have an additional SPN building block composed of two places and two stochastic transitions on the top of Figure 10.

One token at place *UP* means that the system is working properly. The transition *ET3* represents the failure of the system, and when it fires, one token is consumed from *UP* and produced at *DOWN*, making the system unavailable. This transition receives the MTTF of the system, as shown in Figure 9. The transition *ET4* represents the repair of the system, and when it fires, one token is consumed from *DOWN* and produced at *UP*, making the system available again. This transition receives the MTTR values which can be calculated through Equation 5. The MTTF and *availability* values are calculated using the Fault Tree model (see results in Figure 9).

The transition *ET2* fires only when there is one token at *UP*, i.e. when the system is operational. Otherwise, tokens are accumulated at *P2*, forming a queue, and the throughput of the system is impacted. To ensure this interdependence, the transition *ET2* has the following guard function: $\#UP > 0$ i.e. if there is at least one token at *UP*, the transition can be enabled to fire.

Table III shows the performance results per request class (as defined in Table II) and scenarios (as defined in sub-section III-C).

TABLE III
THROUGHPUT (IN REQUESTS/DAY) TAKING INTO ACCOUNT THE *availability* OF THE APPLICATION PRESENTED IN FIGURE 9

Scenarios/Classes	**A**	**B**	**C**	**D**
I	21884.5649	12321.9567	4052.2402	2138.8977
II	21911.8204	12337.2670	4057.2643	2141.5482
III	21905.7834	12333.8735	4056.1499	2140.9603
IV	21933.1914	12349.2439	4061.1859	2143.6162
V	21905.7766	12333.8765	4056.1531	2140.9621
VI	21933.1933	12349.2497	4061.1893	2143.6181

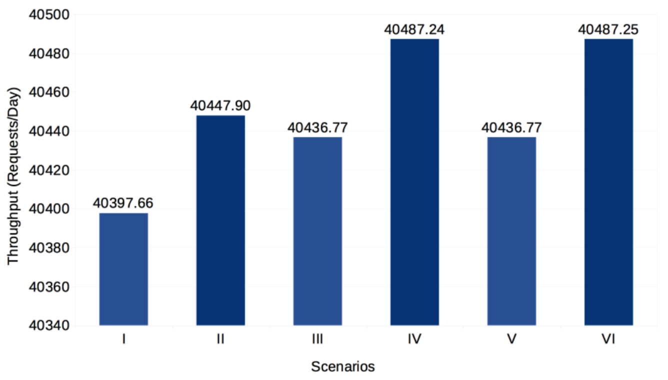

Fig. 11. Total throughput of the four classes of requests across the defined scenarios in Table I

As one can see from Table III, the values of throughput vary dramatically with the four request classes; requests with a high probability of occurrence, as defined in Table II, generate higher throughput traffic. We can also notice from Figure 11 that the total throughput of the four classes of requests does not vary significantly across the defined scenarios. As expected, higher throughput is returned by the scenarios with higher *availability* (level of redundancy). We can also notice that Scenarios IV and VI have the highest throughput due to their having the highest *availability* (as shown in Figure 9). The values of their throughput are approximately the same, also due to their very close *availability* metrics. Thus, taking into account these results and the results of *availability* (discussed in the previous section), we can conclude that the number of replicas does not always affect the availability and throughput results.

V. RELATED WORK

Several prior studies have considered the performance analysis of web and search services. For example, in [23], the authors introduced an approach for bootstrapping to evaluate and monitor the performance- and availability-related QoS attributes. They use a flexible Web service invocation mechanism combined with aspect-oriented programming to weave performance measurement aspects directly into the byte-code of the Web service stubs to gain maximum flexibility. Evaluation results proved the usability of the approach for QoS-based service selection.

Petri net models have been used successfully in the performance analysis of different systems such as cloud and edge computing [24]–[26] and IoT [27]. It has also been used for performance analysis of web and search services. In [28], the authors evaluated the performance of client-server, mobile-agent, and remote-evaluation paradigms in an information retrieval scenario using non-Markovian Petri-net models. The Petri-net models proposed in [28] were relatively simple and assumed execution parameters, such as code and reply size, processing time, and a number of examined servers for queries that were not derived from experimental measurements. In [29], the authors improved their results by defining more accurate Petri-net models fed with experimentally-evaluated parameters through in-depth measurements of the behavior of real users. Thus, whereas [28] was concerned with the characterization of the users behavior in a typical information retrieval scenario, based on thousands of queries generated by hundreds of users search sessions, [29] solves some very accurate Petri-net models thus allowing for complete performance analysis of the main communication paradigms. Petri-net is also used by [30] to model traffic generation patterns of internet-based real-time block-transfer applications (e.g. Web browsing). Model flexibility is achieved through adjustable system parameters.

To the best of our knowledge, no work looked at performance analysis of ElasticSearch as a search engine. In this paper, we evaluated three QoS parameters related to ElasticSearch; we looked at Availability and Reliability metrics. The reliability and availability models were fed into our performance analysis for better monitoring and evaluation of the target system.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we have modelled and analysed the availability and the performance of an enterprise search architecture (ElasticSearch) deployed in a real-world service, Linknovate.com. We proposed a Fault Tree model and analysing different scenarios defined by shard distributions and the level of redundancy of replicas in the ES client node. The results of the analysis suggest that the best results were reached in scenarios with a high level of redundancy.

The results of the proposed Fault tree model analysis were integrated with a Stochastic Petri Net (SPN) model for performance analysis of the Linknovate.com architecture to analyze how failures and redundancy impact on application performance. The results of this analysis suggested that higher throughput is returned by the scenarios with higher availability.

To keep reduced response times when request peaks occur, Linknovate.com currently over-provision nodes in the cloud. This is a clear source of inefficiency which can be optimized to reduce costs and energy consumption. As future work, dynamic provisioning (auto-scaling) together with the co-location of nodes and users can be addressed. Also, we plan to explore the relationships between redundancy and availability from an economic perspective and derive appropriate cost-benefit models. We also plan to validate the proposed model/analysis using a real measurement of the system studied and also study the scalability of the proposed solution.

Acknowledgment

This work is partly funded by the European Unions Horizon 2020 Research and Innovation Programme through RECAP (https://recap-project.eu/) under Grant Agreement Number 732667, by the World Technology Universities Network (WTUN), and by the Irish Institute of Digital Business (dotLAB).

References

[1] "Adults: Media use and attitudes report 2019," https://www.ofcom.org.uk, accessed: 2019-06-07.

[2] "Number of explicit core search queries powered by search engines in the united states as of january 2019 (in billions)," https://www.statista.com/statistics/265796/us-search-engines-ranked-by-number-of-core-searches/, accessed: 2019-06-07.

[3] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, Dec. 2008. [Online]. Available: http://doi.acm.org/10.1145/1496091.1496103

[4] B. B. Cambazoglu and R. Baeza-Yates, "Scalability and efficiency challenges in large-scale web search engines," in *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '16. New York, NY, USA: ACM, 2016, pp. 1223–1226. [Online]. Available: http://doi.acm.org.dcu.idm.oclc.org/10.1145/2911451.2914808

[5] Elasticsearch B.V, "Open Source Search Analytics - ElasticSearch," 2019. [Online]. Available: https://www.elastic.co/

[6] O. Kononenko, O. Baysal, R. Holmes, and M. W. Godfrey, "Mining modern repositories with elasticsearch," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 328–331. [Online]. Available: http://doi.acm.org.dcu.idm.oclc.org/10.1145/2597073.2597091

[7] J. Muppala, R. Fricks, and K. Trivedi, "Techniques for system dependability evaluation," in *Computational Probability*, ser. International Series in Operations Research & Management Science, W. Grassmann, Ed. Springer US, 2000, vol. 24, pp. 445–479.

[8] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 11–33, Jan 2004.

[9] M. Rausand and A. Hsyland, *System reliability theory : models, statistical methods, and applications*, 2nd ed. John Wiley & Sons, Inc., Publication, 2004.

[10] K. S. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*, 2nd ed. Wiley-Interscience, 2001.

[11] L. Xing and S. V. Amari, *Handbook of Performability Engineering*. Springer, 2008, ch. Fault Tree Analysis, pp. 595–617.

[12] I. Silva, L. A. Guedes, P. Portugal, and F. Vasques, "Reliability and availability evaluation of wireless sensor networks for industrial applications," *Sensors*, vol. 12, no. 1, pp. 806–838, 2012.

[13] J. S. Choi and N. Z. Cho, "A practical method for accurate quantification of large fault trees," *Reliability Engineering and System Safety*, vol. 92, no. 7, pp. 971 – 982, 2007.

[14] I. Fe, R. Matos, J. Dantas, C. Melo, and P. Maciel, "Stochastic model of performance and cost for auto-scaling planning in public cloud," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2017, pp. 2081–2086.

[15] A. Rogge-Solti and M. Weske, "Prediction of business process durations using non-markovian stochastic petri nets," *Information Systems*, vol. 54, pp. 1–14, 2015.

[16] F. A. Silva, S. Kosta, M. Rodrigues, D. Oliveira, T. Maciel, A. Mei, and P. Maciel, "Mobile cloud performance evaluation using stochastic models," *IEEE Transactions on Mobile Computing*, vol. 17, no. 5, pp. 1134–1147, 2017.

[17] L. Capra, "Stochastic petri nets with changeable layout," in *World Conference on Information Systems and Technologies*. Springer, 2017, pp. 831–840.

[18] N. Yang, H. Yu, H. Sun, and Z. Qian, "Modeling uml sequence diagrams using extended petri nets," *Telecommunication Systems*, vol. 51, no. 2-3, pp. 147–158, 2012.

[19] O. Kononenko, O. Baysal, R. Holmes, and M. W. Godfrey, "Mining modern repositories with elasticsearch," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 328–331.

[20] Y. Li, Y. Jiang, J. Gu, M. Lu, M. Yu, E. M. Armstrong, T. Huang, D. Moroni, L. J. McGibbney, G. Frank, and C. Yang, "A cloud-based framework for large-scale log mining through apache spark and elasticsearch," *Applied Sciences*, vol. 9, no. 6, 2019. [Online]. Available: http://www.mdpi.com/2076-3417/9/6/1114

[21] R. Birke, I. Giurgiu, L. Y. Chen, D. Wiesmann, and T. Engbersen, "Failure analysis of virtual and physical machines: Patterns, causes and characteristics," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2014, pp. 1–12.

[22] D. S. K. Tuan Anh Nguyen and J. S. Park, "A comprehensive availability modeling and analysis of a virtualized servers system using stochastic reward nets," *The Scientific World Journal*, vol. 2014, p. 18, 2014.

[23] F. Rosenberg, C. Platzer, and S. Dustdar, "Bootstrapping performance and dependability attributes ofweb services," in *2006 IEEE International Conference on Web Services (ICWS'06)*. IEEE, 2006, pp. 205–212.

[24] G. L. Santos, P. T. Endo, M. F. F. da Silva Lisboa, L. G. F. da Silva, D. Sadok, J. Kelner, T. Lynn *et al.*, "Analyzing the availability and performance of an e-health system integrated with edge, fog and cloud infrastructures," *Journal of Cloud Computing*, vol. 7, no. 1, p. 16, 2018.

[25] D. Rosendo, P. T. Endo, G. L. Santos, D. M. Gomes, G. Gonçalves, A. Moreira, J. Kelner, D. Sadok, and M. Mahloo, "Modeling and analyzing power system failures on cloud services," in *2017 13th International Conference on Network and Service Management (CNSM)*. IEEE, 2017, pp. 1–7.

[26] G. L. Santos, P. T. Endo, G. Gonçalves, D. Rosendo, D. Gomes, J. Kelner, D. Sadok, and M. Mahloo, "Analyzing the it subsystem failure impact on availability of cloud services," in *2017 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2017, pp. 717–723.

[27] G. Hwang, J. Lee, J. Park, and T.-W. Chang, "Developing performance measurement system for internet of things and smart factory environment," *International journal of production research*, vol. 55, no. 9, pp. 2590–2602, 2017.

[28] A. Puliafito, S. Riccobene, and M. Scarpa, "Which paradigm should i use? an analytical comparison of the client–server, remote evaluation and mobile agent paradigms," *Concurrency and Computation: Practice and Experience*, vol. 13, no. 1, pp. 71–94, 2001.

[29] M. Scarpa, A. Puliafito, M. Villari, and A. Zaia, "A modeling technique for the performance analysis of web searching applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 11, pp. 1339–1356, 2004.

[30] D. Gvozdanovic, D. Simic, U. Vizek, M. Matijasevic, K. P. Valavanis, and D. Huljenic, "Petri net based modeling of application layer traffic characteristics," in *EUROCON'2001. International Conference on Trends in Communications. Technical Program, Proceedings (Cat. No. 01EX439)*, vol. 2. IEEE, 2001, pp. 424–427.