Fault Tolerant Internet Computing: Benchmarking and Modelling Trade-offs Between Availability, Latency and Consistency

Anatoliy Gorbenko[1,3*], Alexander Romanovsky[2], Olga Tarasyuk[3]

[1]Leeds Beckett University, Leeds, UK; [2]Newcastle University, Newcastle-upon-Tyne, UK;
[3]National Aerospace University, Kharkiv, Ukraine

*Corresponding author. E-mail address: A.Gorbenko@leedsbeckett.ac.uk

Abstract—The paper discusses our practical experience and theoretical results of investigating the impact of consistency on latency in distributed fault tolerant systems built over the Internet and clouds. We introduce a time-probabilistic failure model of distributed systems that employ the service-oriented paradigm for defining cooperation with clients over the Internet and clouds. The trade-offs between consistency, availability and latency are examined, as well as the role of the application timeout as the main determinant in the interplay between system availability and responsiveness. The model introduced heavily relies on collecting and analysing a large amount of data representing the probabilistic behaviour of such systems. The paper presents experimental results of measuring the response time in a distributed service-oriented system whose replicas are deployed at different Amazon EC2 location domains. These results clearly show that improvements in system consistency increase system latency, which is in line with the qualitative implication of the well-known CAP theorem. The paper proposes a set of novel mathematical models that are based on statistical analysis of collected data and enable quantified response time prediction depending on the timeout setup and on the level of consistency provided by the replicated system.

Keywords—service-oriented systems, internet computing, cloud computing, distributed applications, fault tolerance, modelling techniques, trade-off, availability, latency, consistency

1. Introduction

Internet and cloud computing has become an industrial trend, indispensable in dealing with enormous data growth. It is now widely used in different market niches, including critical infrastructures and business-critical systems. Failures of such applications can affect people's lives and businesses. For example, Amazon's S3 cloud storage widespread outage on February 28, 2017 knocked numerous web services offline and costed S&P 500 companies at least $150 million, and U.S. financial-service companies $160 million in lost revenue [1]. A spate of recent service outages of the Amazon, Google, MS Azure, Dropbox and other cloud platforms[1,2,3] highlights the risks involved when companies rely on Internet computing and cloud resources in their mission-critical applications. Thus, ensuring dependability of Internet computing and of the whole spectrum of related technologies (web services, SOA, clouds, Big Data, etc.) is a must, as well as a challenge. The recent microservice architectural style [2] offers greater interoperability and reduces the overall cost of system design and composition but introduces additional operational complexity, increases system latency and its variation due to inter-service

[1] https://www.analyticsindiamag.com/cloud-outages-that-shook-the-tech-world-2018/
[2] https://www.crn.com/slide-shows/cloud/the-10-biggest-cloud-outages-of-2018
[3] https://www.theregister.co.uk/2017/03/01/aws_s3_outage/

rather than in-process calls and also presents reliability issues similar to SOA and web services but on a larger scale.

Although the Internet and cloud computing technologies have been significantly improved recently, we believe that they have not yet revealed their full potential. In particular, it is still in its infancy when it comes to ensuring dependability of large-scale dynamically composed service-oriented systems involving multiple independent services and providers. Dependability enhancing technologies will thus be essential in supporting mission- and business-critical applications intended for personal use or to be used by enterprises, governments or defence.

There is significant research devoted to dependability and performance of Internet computing, clouds and SOA (e.g. [3, 4, 5]). Recent related works, such as [6, 7, 8, 9], have introduced several approaches to incorporating fault tolerance techniques (including *N*-modular redundancy, voting, backward and forward error recovery and replication techniques) into clouds and web service architectures. Important research has been done in fault analysis, evaluation and experimental measurements of dependability and performance of service-oriented systems, e.g. [10, 11, 12]. However, coming from dispersed areas, this work addresses individual issues but has not so far advanced them in combination or offered general solutions. Often, researchers use simple and hence unrealistic failure models or fail to take into account the interdependency between availability and performance that is in the very nature of such distributed interacting systems. For instance, basic fault tolerance solutions such as *N*-modular, hot- and cold-spare redundancy usually assume a synchronous communication between replicas, which means that every message is delivered within a known fixed amount of time [13]. This is a reasonable simplification for the local area systems whose components are compactly located, for instance, within a single data centre. This assumption does not appear to be relevant, however, for the wide area systems, in which replicas are deployed over the Internet and their updates cannot be propagated immediately, making it difficult to guarantee consistency.

To be more efficient, fault tolerance techniques incorporated into Internet and cloud computing applications should distinguish between evident failures of different types, such as application exceptions, communication errors and timeouts, and should be capable of minimizing the probability of non-evident application errors. In addition, the Internet and, more generally, the wide area networked systems are characterized by a high level of *uncertainty*, which makes it hard to guarantee that a client will receive a response from the service within a finite time. It has been previously shown that there is a significant uncertainty of response time and other timeliness parameters in service-oriented systems invoked over the Internet [14, 15]. This uncertainty significantly affects QoS capability of distributed applications. Experimental studies [15, 16] show that the response time of web services can very often be as high as 10 or even 20 times the average value. Moreover, sometimes client applications wait for a response from a service for hours instead of reporting an exception or resending a request. Therefore, the right timeout setting is key to improving performance of many distributed systems, including web services.

Besides, other research [17, 18] and our previous studies show that failures are a regular occurrence on the Internet, in clouds and scale-out data centre networks. When developers apply replication and other fault tolerance techniques in the Internet- and cloud-based systems, they need to understand the time overheads and be concerned about delays and their uncertainty.

In this paper we put forward an advanced failure model for distributed systems and Internet computing applications, taking into account the time-probabilistic relation between different failure modes, and propose analytical models that assess the average servicing and waiting times under certain timeout settings.

Secondly, we examine, both in experimental and theoretical terms, how different fault tolerance solutions [19] implemented over the Internet affect system latency depending on the replication factor and the level of consistency provided.

The paper applies the time-probabilistic failure model, proposed in our earlier work [20] to the CAP conjecture. It discusses trade-offs between consistency, availability and latency taking into account timeout settings. Although these relations have been identified by the CAP theorem in qualitative terms [21, 22], it is still necessary to quantify how different fault tolerance techniques affect system latency depending on the consistency level. Even when the response times of replicas are known, it is not possible to accurately predict latency of the whole replicated system. Hence, the *ultimate goal* of the paper is to provide developers of distributed fault-tolerant systems with the mathematical models and practical guidance allowing them to predict latency of such systems taking into account timeout settings and the required consistency level. The proposed models will help them to trade-off between consistency, availability and latency during system design and operation. In our work we combine experimental measurement of replicas response time with the probabilistic theory and analytical modelling of system latency which makes it possible to predict its dependability and performance depending on the chosen consistency level and timeout setup.

The rest of the paper is organized as following. In Sections 2 and 3 we discuss the uncertainty challenge inherent to distributed service-oriented systems and introduce a time-probabilistic failure model which captures the interplay between system dependability and performance characteristics. Section 4 discusses the impact of the CAP theorem on design principles of modern distributed fault-tolerant systems and examines trade-offs between system consistency, availability and latency. In Section 5 we summarise the results of experimental response time measurements for a testbed fault-tolerant system supporting different consistency levels whose replicas are distributed over clouds. The probabilistic models introduced in Section 6 define the quantitative relation between the system response time and the required consistency level. In Section 7 we evaluate the accuracy of the proposed analytical models by comparing their results with our experimental data. Section 8 investigates how system response time changes with increasing number of generic replicas. Finally, conclusions and practical lessons learnt are summarised in Sections 9 and 10.

2 The Uncertainty Challenge

Internet computing mainly relies on service oriented architectural model where web services (WSs) play a role of major building blocks, often provided by third parties. By their very nature such web services are black boxes, as neither their source code, nor their complete specification, nor information about their deployment environments are available; the only known information about them is their interfaces. Moreover, their dependability is not completely known and they may not provide sufficient quality of service. As a result, it is often safe to treat third party WSs as “dirty” boxes, assuming that they always have bugs, do not fit enough, have poor specification and documentation. WSs are heterogeneous, as they might be developed following different standards, fault assumptions, and different conventions and may use different technologies.

Service-oriented systems are built as overlay networks over the Internet. As a result, their dependable construction and composition are complicated by the fact that, due to a lack of quality and predictability, the Internet is a poor communication medium. Service-oriented systems can be vulnerable to internal faults from various sources and casual external problems such as communication failures, routing errors and network traffic congestion. Therefore, the performance of such systems is characterised by high instability, i.e. it can vary over a wide range in a random and unpredictable manner [14].

The inability of the web services involved to guarantee a certain response time and performance and the instability of the communication medium can cause timing failures, when the response time or the timing of service delivery (i.e. the time during which information is delivered over the network to the service interface) exceeds the time that would be required in order for the system function to be executed. A timing failure may take the form of an early or late response, depending on whether the service is delivered too early or too late [19].

In the case of complex workflows incorporating many different web services, some users may be provided with a correct service, whereas others may have to deal with incorrect services of various types due to timing errors. These errors may occur in any of a number of system components depending on the relative position of a particular user and a particular web service in the Internet, as well as on the instability points which emerge during the execution. Thus, timing errors can become a major cause of inconsistent failures usually referred to, after [23], as the Byzantine failures. Providing remote services, data storage and computing resources is an important element of modern IT and Internet computing. However, significant uncertainty exists regarding service-oriented systems invoked over the Internet [16]. In this work we use the general synthetic term *uncertainty* to refer to the unknown, unstable, unpredictable, changeable characteristics and behaviour of web services and SOA, exacerbated by running these services over the Internet and clouds.

Understanding uncertainty arising in SOA is crucial for choosing the right recovery techniques, setting timeouts, and adopting system architecture and its behaviour to a changing environment such as the Internet and SOA. This uncertainty exhibits itself through unpredictable worst-case response times, unknown service dependability, and the difficulty of diagnosing the root cause of failures. Uncertainty is one of the main challenges to building dependable distributed systems of Internet-scale. This uncertainty is a threat in much the same way that faults, errors, and failures are [19]. Here, we examine that threat and discuss ways to deal with it. We particularly focus on using timeouts as part of fault- and intrusion-tolerance techniques.

Uncertainty has three important consequences. First, it makes it difficult to assess a service’s availability and performance and hence to choose that service over others for its trustworthiness. Second, it complicates the application of fault- and intrusion-tolerance techniques because too much data is missing to make good decisions and exploit dependability mechanisms’ features. Finally, it makes it difficult to predict the performance, cost, and other non-functional characteristics when you apply such techniques over the Internet. Clearly, building fast, dependable Internet applications on a large scale is impossible without addressing these issues.

Our recent studies and a series of experiments [14, 16] showed that the uncertainty in large-scale distributed systems can be effectively mitigated by employing a probabilistic approach. This work defines services response times using probability density functions (*pdf*) instead of using their average values. The *pdf* specifies the relative likelihood for the response time to take on a given value, that is much more informative than response time average or the worst case value. It allows us to estimate a probability that the system response time is less than the specified value or to define confidence intervals. The probability density function can be chosen/estimated by statistical processing of response time measurement results. The corresponding technique is described in [16].

3 Time-Probabilistic Failure Model

Web services and service-oriented systems as any other complex software may contain faults which may manifest themselves in operation. To every request, the web service might return either a correct response – that is, succeed – or an erroneous response or exception—that is, fail. Web services failure behaviour is characterised by the probability of failure on demand (*pfd*). This probability can be statistically measured by a client as a ratio between *r* failures observed in *n* demands [24]. It can vary between the environments and the contexts (operational profiles) in which a web service is used.

The various factors, which affect the *pfd* may be unknown with certainty, thus the value of *pfd* may be uncertain as well. This uncertainty can be captured by a probability density series or probability distribution, built by aggregating usage experience of different clients. A user-collaborative mechanism, aggregating data from multiple clients, was proposed in [25].

Thus, the response returned to the client by a remote service may be of several types:

1. *Correct result*.

2. *Evident error* – an error that needs no special means to be detected. It concerns exception messages of different types reported to the client and notifying about denial of the requested service for some reason.

3. *Non-evident (hidden) error* – an error that can be detected only by using a multiversioning at the application level (e.g. diversity of web services used).

However, the distributed nature of the service-oriented architectural model does not guarantee that the client receives a response from the web service within the finite time. If this happens we face so-called timing failures when the response is received too late or is not received at all. Thus, the known dependability definition [19] should be extended for SOA as the “ability to deliver service *within the expected time* that can justifiably be trusted”.

In the Figure 1 we adopt the failure model introduced by Avizienis, et al. in [19] to the distributed nature of service-oriented systems and, more general, Internet computing. The model distinguishes between the two main failure domains: (i) *timing failures* when the duration of the response delivered to the client exceeds the specified waiting time – the application timeout (i.e. the service is delivered too late), and (ii) *content failures* when the content (value) of the response delivered to the client deviates from implementing the system function.

Probabilities p^{ok}, p^{he} and p^{ex} are conditional probabilities. They are conditioned on the arrival of some response within the timeout. Probabilities p^{ex} and p^{he} refer to failure modes

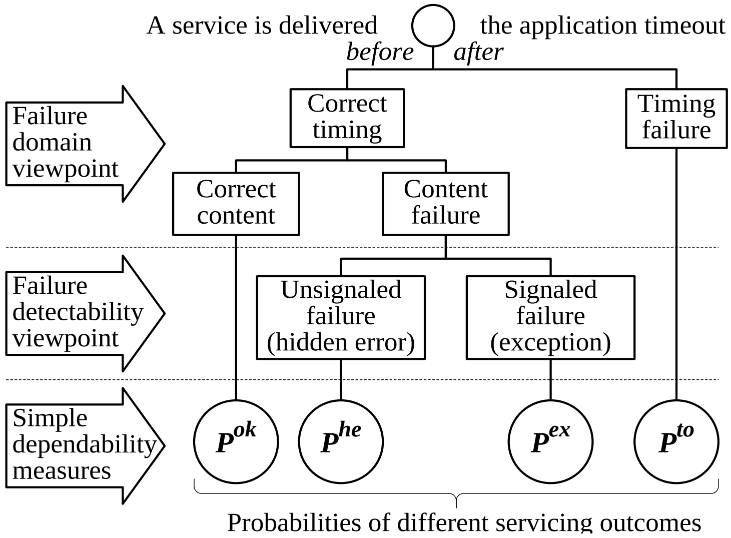


Fig. 1. Service failure modes

that in the Avizienis's classification correspond to the *detectability* viewpoint, where they are classified as: *signaled* and *unsignaled* failures, respectively.

In our failure model we use the following assumptions:

– probabilities of all servicing outcomes (p^{ok}, p^{he}, p^{ex}, p^{to}) form a set of collectively exhaustive events;

– system response time is a random variable with the known probability density function $f_t(t)$ and certain parameters specified based on the result of statistical analysis of measured results;

– time during which a client waits for the response is limited by the *timeout* parameter;

– probabilities of the correct, evident and non-evident incorrect servicing (p^{ok}, p^{ex} and p^{he}) do not depend on the time of the response delivery to the client.

The justification of the assumptions and a detailed discussion of model properties can be found in [20].

The interdependency between probabilities of different servicing outcomes is shown in Fig. 2. Changing of timeout value causes changing the probability of timeout and, hence changing (redistribution) values of p^{ok}, p^{he}, p^{ex} and p^{to} as long as the sum of all probabilities must be equal to one. Hence, they are functions of a *timeout* setting:

$$p^{ok}(timeout) = p^{ok\infty} \cdot \int_0^{timeout} f_t(t)dt \tag{1}$$

$$p^{he}(timeout) = p^{he\infty} \cdot \int_0^{time\text{-}out} f_t(t)dt \tag{2}$$

$$p^{ex}(timeout) = p^{ex\infty} \cdot \int_0^{timeout} f_t(t)dt \tag{3}$$

where $p^{ok\infty}$, $p^{ex\infty}$, $p^{he\infty}$ are the 'eventual' probabilities of getting a correct, evident and non-evident erroneous results assuming the infinite waiting time, i.e. when *timeout* →∞.

The timeliness related unavailability of a system can be estimated as the probability of the client receiving a response after the specified application timeout:

$$p^{to}(timeout) = \int_{timeout}^{\infty} f_t(t)dt \tag{4}$$

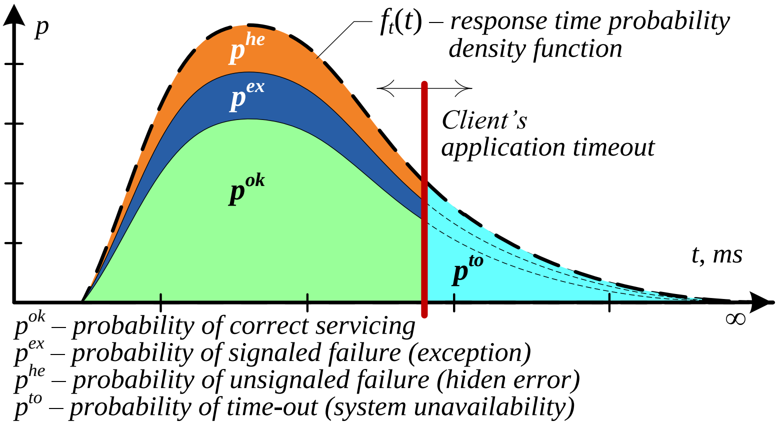

p
$f_t(t)$ – response time probability density function
p^{he}
p^{ex}
p^{ok}
p^{to}
Client's application timeout
t, ms
∞
p^{ok} – probability of correct servicing
p^{ex} – probability of signaled failure (exception)
p^{he} – probability of unsignaled failure (hiden error)
p^{to} – probability of time-out (system unavailability)

Fig. 2. Time-probabilistic failure model: the trade-off between availability and latency depending on timeout setup

Besides, we introduce the following two measures estimating system latency: T^{av_srv} – the *average servicing time* and T^{av_wait} – the *average waiting time*. The expectation of $f_t(t)$ truncated from the right by a timeout is the average response time of those invocations in which the client receives a response of any type before the specified timeout (i.e. the average servicing time):

$$T^{avg\_srv}(timeout) = \frac{\int_0^{timeout} t \cdot f_t(t)dt}{F_t(timeout)} \tag{5}$$

where $F_t(timeout) = \int_0^{timeout} f_t(t)dt$ is the cumulative distribution function of a response time.

The average waiting time T^{avg_wait} estimated for all invocations, including those when a timeout is triggered, is the sum of T^{avg_srv} under the specified timeout and a product of the timeout value and the probability of timeout:

$$T^{avg\_wait}(timeout) = \int_0^{timeout} t \cdot f_t(t)dt + timeout \cdot \left(1 - F_t(timeout)\right) \tag{6}$$

It can be seen that T^{av_srv} and T^{av_wait} are becoming equal when the timeout increases to infinity. But in all practical settings T^{av_srv} is less than T^{av_wait}. This is because the waiting time for those invocations for which a timeout is triggered is equal to the timeout value. So, the weight of a tail of $f_t(t)$ truncated by the timeout is concentrated at the truncation border which increases the average waiting time T^{av_wait}.

Using these equations, systems engineers can trade-off between maximizing the service availability and minimizing its latency. Besides, these equations can help to choose appropriate application timeouts, which are the main error detection mechanism here. To be applicable in practice the proposed models have to be concretized using the explicitly defined probability density function. For instance, if system response time is approximated by the exponential distribution $f_t(t) = \mu \cdot e^{-\mu \cdot t}$ (where μ is the rate parameter which is inversely proportional to the mean), the trade-offs between latency, availability and timeout will be identified as following:

$$p^{to}(timeout) = e^{-\mu \cdot timeout} \;\; => \;\; timeout(p^{to}) = -\frac{\ln(p^{to})}{\mu};$$

$$T^{avg_{srv}}(timeout) = -\frac{e^{-\mu \cdot timeout} + \mu \cdot timeout \cdot e^{-\mu \cdot timeout} - 1}{\mu \cdot (1 - e^{-\mu \cdot timeout})},$$

$$T^{avg\_wait}(timeout) = -\frac{e^{-\mu \cdot timeout} - 1}{\mu} =>$$

$$=> T^{avg\_srv}(p^{to}) = -\frac{p^{to} \cdot \ln(p^{to}) - p^{to} + 1}{\mu \cdot (p^{to} - 1)},$$

$$T^{avg\_wait}(p^{to}) = -\frac{p^{to} - 1}{\mu},$$

$$timeout\left(T^{avg\_wait}\right) = -\frac{\ln\left(1 - \mu \cdot T^{avg\_wait}\right)}{\mu},$$

$$p^{to}\left(T^{avg\_wait}\right) = 1 - \mu \cdot T^{avg\_wait}.$$

The numerical example of solving the trade-offs and estimating the probabilities of different types of failures and system latency T^{av_srv} and T^{av_wait} depending on timeout settings can be found in [20]. In our work timeout links system availability and latency. It can also be used as part of failure recovery techniques to trigger the restart or retry in software systems [26].

4 Trade-offs Between Consistency, Availability and Latency in Fault-Tolerant Internet Computing

The CAP conjecture [21], which first appeared in 1998-1999, defines a trade-off between system availability, consistency and partition tolerance, stating that only two of the three properties can be preserved in distributed replicated systems at the same time. Gilbert and Lynch [22] view the CAP theorem as a particular case of a more general trade-off between consistency and availability in unreliable distributed systems which assume that updates are eventually propagated. System partitioning, availability and latency are tightly connected. A replicated fault-tolerant system becomes partitioned when one of its parts does not respond due to arbitrary message loss, delay or replica failure, resulting in a timeout.

System availability can be interpreted as a probability that each client request eventually receives a response. In many real systems, however, a response that is too late (i.e. beyond the application timeout) is treated as a failure. High latency is an undesirable effect for many interactive web applications. In [27] the authors showed that if a response time increases by as little as 100 ms, it dramatically reduces the probability of the customer continuing to use the system. Failure to receive responses from some of the replicas within the specified timeout causes partitioning of the replicated system. Thus, partitioning can be considered as a bound on the replica's response time [28]. A slow network connection, a slow-responding replica or the wrong timeout settings can lead to an erroneous decision that the system has become partitioned. When the system detects a partition, it has to decide whether to return a possibly inconsistent response to a client or to send an exception message in reply, which undermines system availability.

The designers of the distributed fault-tolerant systems cannot prevent partitions which happen due to network failures, message losses, hacker attacks and components crashes and, hence, have to choose between availability and consistency. One of these two properties has to be sacrificed. If system developers decide to forfeit consistency they can also improve the system response time by returning the fastest response to the client without waiting for other replica responses until the timeout, though this would increase the probability of providing inconsistent results. Besides, timeout settings are also important. If the timeout is lower than the typical response time, a system is likely to enter the partition mode more often [20].

It is important to remember that none of these three properties is binary. For example, modern distributed database systems, e.g. Cassandra [29], can provide a discrete set of different consistency levels for each particular read or write request. The response time can theoretically vary between zero and infinity, although in practice it ranges between a minimal affordable time higher than zero and the application timeout. Availability varies between 0% and 100% as usual.

The architects of modern distributed database management systems and large-scale web applications such as Facebook, Twitter, etc. often decide to relax consistency requirements by introducing asynchronous data updates in order to achieve higher system availability and allow a quick response. Yet the most promising approach is to balance these properties. For instance, the Cassandra NoSQL database introduces a tunable replication factor and an adjustable consistency model so that a customer can choose a particular level of consistency to fit with the desired system latency.

The CAP theorem helps the developers to understand the system trade-offs between consistency and availability/latency [30]. Yet even though this theorem strongly suggests that better consistency undermines system availability and latency, developers do not have quantitative models to help them to estimate the system response time for the chosen consistency level and to achieve a precise trade-off between them. Our interpretation of the CAP theorem and the trade-offs resulting from the CAP is depicted in Fig. 3.

The application timeout can be considered as a bound between system availability and performance (in term of latency or response time) [31]. Thus, system designers should be able to set up timeouts according to the desired system response time, also keeping in mind the choice between consistency and availability. We represent the response time as a random variable which possible values and their associated probabilities can be described by a discrete or continuous distribution function.

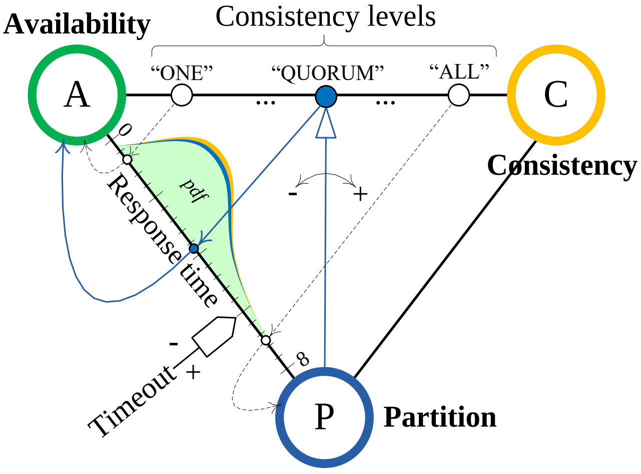


Fig. 3. The CAP trade-offs model

Knowing this function, the system designer can predict the average system latency or estimate a probability of getting response by the specified timeout. In turn, for the distributed replicated system this function is determined by distribution functions of replicas response times and depends on the total number of replicas and the consistency level, provided by a system. In the following sections we discuss our practical experience in measuring latency of fault-tolerant distributed systems depending on the number of replicas and provided consistency level. We also introduce analytical models defining distribution functions of system response time and predicting system latency.

5 Experimental Investigation of the Consistency Impact on Response Time

5.1 Testbed Fault-Tolerant Distributed System

To investigate the CAP impact on fault-tolerant distributed systems we developed a testbed service-oriented system composed of a number of replicated web services. Modern distributed systems and services like Amazon S3, Amazon EMR, Facebook Haystack, DynamoDB, Apache Hadoop, etc. replicate data to at least three servers. The wide-area cooperative storage file system analysed in [32] maintains 6 replicas for each file block. In [33] the authors examined the replication degree customization for high availability when a number of replicas ranges from 1 to 6. Thus, in our experiments we ranged a number of replicas from 1 to 7 to cover the most common replication setups.

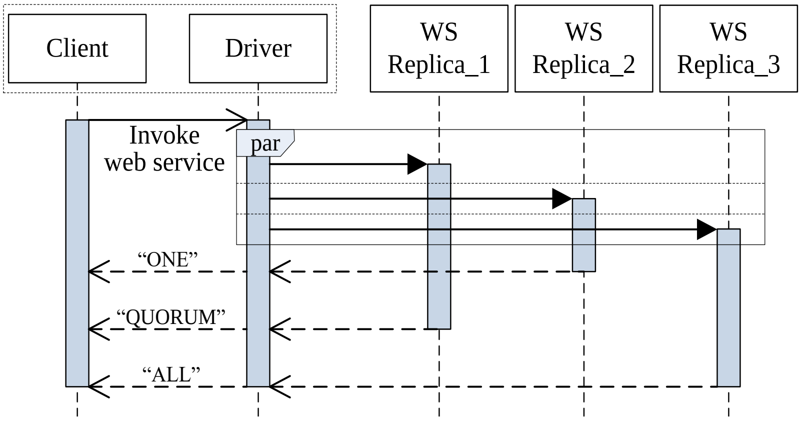


Fig. 4. 3-replicated fault-tolerant service-oriented system

A testbed web service was written in Java and its replicas uploaded to Amazon Elastic Beanstalk and were deployed in the seven different location domains: US West (Oregon), South America (Sao Paulo), Asia Pacific (Tokyo), EU West (Ireland), Asia Pacific (Singapore), US East (Virginia) and US West (N. California). Each WS replica performs a heavy-computational arithmetic algorithm implementing the Gregory–Leibniz series to calculate the mathematical constant Pi and returns the result to the driver. The driver is responsible for invoking each of the replicated web services, waiting for the web services to complete their execution and return response, and, finally, applying a particular fault tolerance scheme with a certain replication factor (see Fig. 4).

In our study we investigated the three basic fault tolerance patterns for web services [34] corresponding to different consistency levels (ONE, ALL, QUORUM). When we refer to consistency here we use the concept of tunable/eventual data consistency [30, 35] that has been recently introduced in the NoSQL and Big Data technologies (e.g. MongoDB, Cassandra, etc.) extending the standard consistency model and quorum-based protocols [36] traditionally adopted in distributed systems. In all cases the driver simultaneously forwards client's request to all replicated web services. The consistency level determines the number of replicas which must return a response to the driver before it sends an adjudicated result to the client application:

- ONE (*hot-spare redundancy*) – when the FASTEST response is received the driver forwards it to the client. This is the weakest consistency level though it guarantees the minimal latency;
- ALL (*N-modular redundancy*) – the driver must wait until ALL replicas return their responses.

In this case the response time is constrained by the slowest replica though the strongest consistency is provided;

- QUORUM – the driver must wait for the responses from a QUORUM of replica web services. It provides a compromise between the ONE and ALL options trading off latency versus consistency. The quorum is calculated as: (*amount_of_replicas* / 2) + 1, rounded down to an integer value.

The driver also implements a timeout mechanism aimed to protect clients from endless waiting in case of network or web-services failures or cloud outages. The driver was implemented as part of the Java client software. The client software was run at a host in the Newcastle University (UK) corporate network. It invoked replica web services several thousand times in a loop using the driver as a proxy.

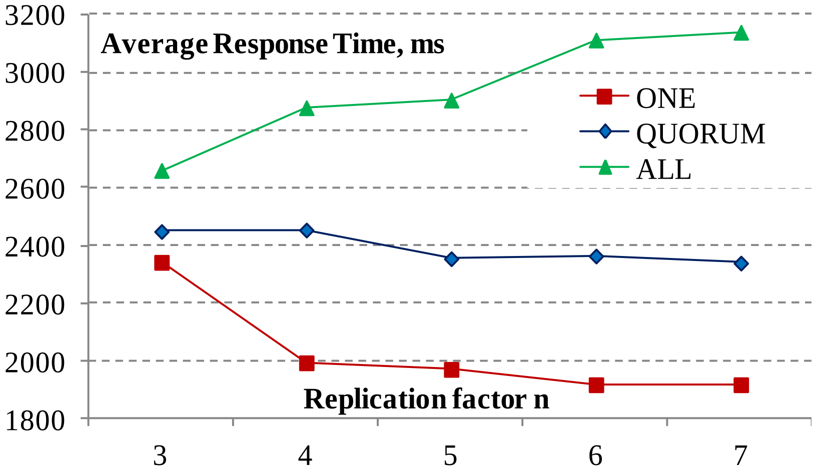


Fig. 5. The average response time of n-replicated fault-tolerant service-oriented system

5.2 Response Time Measurement

For the particular client's request we measured the response time of the each web service replica and also times when the driver produces responses corresponding to different consistency levels. The delay induced by the driver itself was negligible in our experiments.

Tables 1 and 2 summarize basic statistical characteristics of the measured data. Fig. 5 clearly confirms the general CAP implications that increasing consistency worsens system latency and vice versa. In addition, increasing the replication factor decreases the latency of a system providing the weakest consistency level ONE and worsens it if a system provides the strongest consistency level ALL. Though, the particular latency losses or gains are quite irregular and very much depend on response time of system replicas.

TABLE 1. REPLICA RESPONSE TIME STATISTICS

Replica ID	Replica Location	Response Time, ms			
		min.	avg.	max.	std.dev.
Replica1:	US West (Oregon)	2324	2428	2821	60
Replica2:	South America (SaoPaulo)	2164	2434	3371	228
Replica3:	Asia Pacific (Tokyo)	2344	2588	5573	522
Replica4:	EU West (Ireland)	1513	2226	10831	1103
Replica5:	Asia Pacific (Singapore)	2010	2189	5078	300
Replica6:	US East (Virginia)	1816	2252	10931	1095
Replica7:	US West (N. California)	2271	2415	5377	306

TABLE 2. SYSTEM RESPONSE TIME STATISTICS

System replication factor *n*	Consistency Level	Response Time, ms			
		min.	avg.	max.	std.dev
3 (Replicas 1-3)	ONE	2164	2342	2509	80
	QUORUM	2324	2449	2830	72
	ALL	2386	2660	5573	529
4 (Replicas 1-4)	ONE	1513	1993	2404	183
	QUORUM	2324	2454	2830	74
	ALL	2386	2878	10831	1079
5 (Replicas 1-5)	ONE	1513	1970	2367	155
	QUORUM	2164	2354	2509	77
	ALL	2386	2904	10831	1100
6 (Replicas 1-6)	ONE	1513	1917	2159	117
	QUORUM	2164	2364	2520	80
	ALL	2386	3113	10931	1422
7 (Replicas 1-7)	ONE	1513	1917	2159	117
	QUORUM	2164	2339	2509	66
	ALL	2386	3140	10931	1438

In the rest of this Section we analyse in details the data related to the 3-replicated system configuration. As we mentioned earlier, replication factor equal to 3 is the most typical setup for many modern distributed computing systems and Internet services. For instance, Amazon S3 by default replicates user data to three data centres, each separated by large distances across an AWS Region [37]. This follows from the well-known 3-2-1 rule adopted for Cloud backup [38].

The measurement results obtained for the first 100 invocations are presented in Figs. 6 and 7. Probability density series (*pds*) of system and replicas response times are depicted in Figs. 8 and 9. In Fig. 9 we also depict theoretically obtained *pds* of system response time as proposed further in Section 6.2.

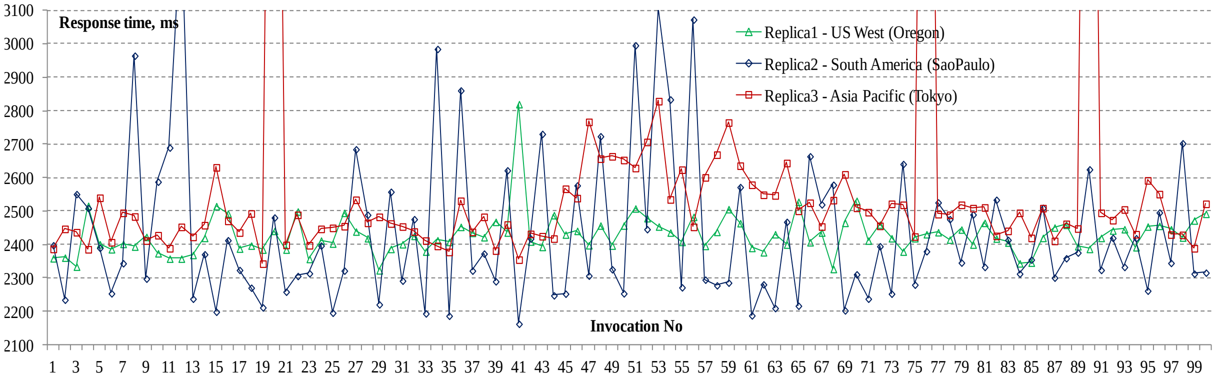


Fig. 6. Response time of different web service replicas for the 3-replicated system setup

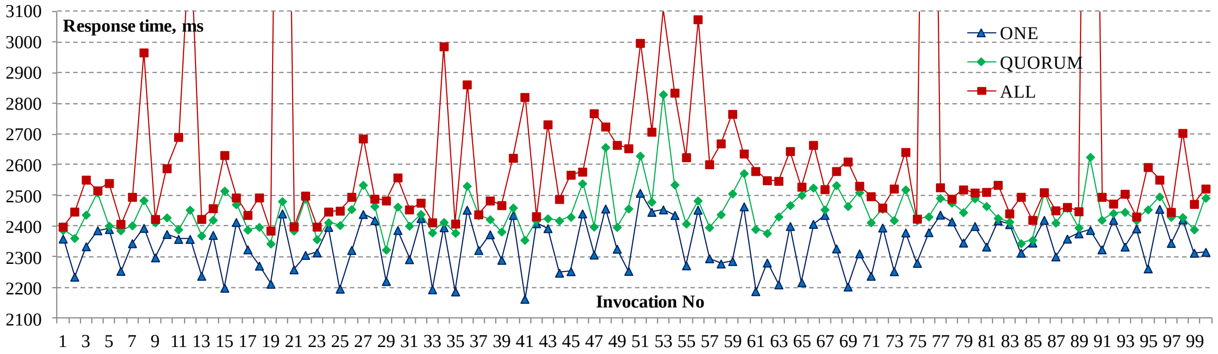


Fig. 7. System response time corresponding to different consistency levels for the 3-replicated system setup

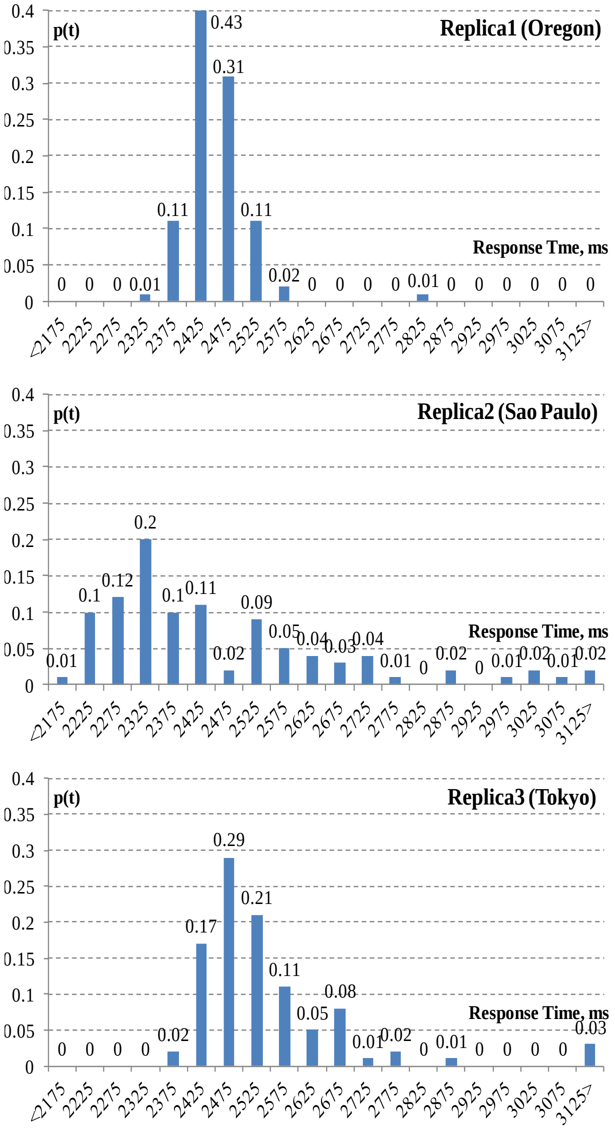


Fig. 8. Probability histograms (*pds*) of replicas response times.

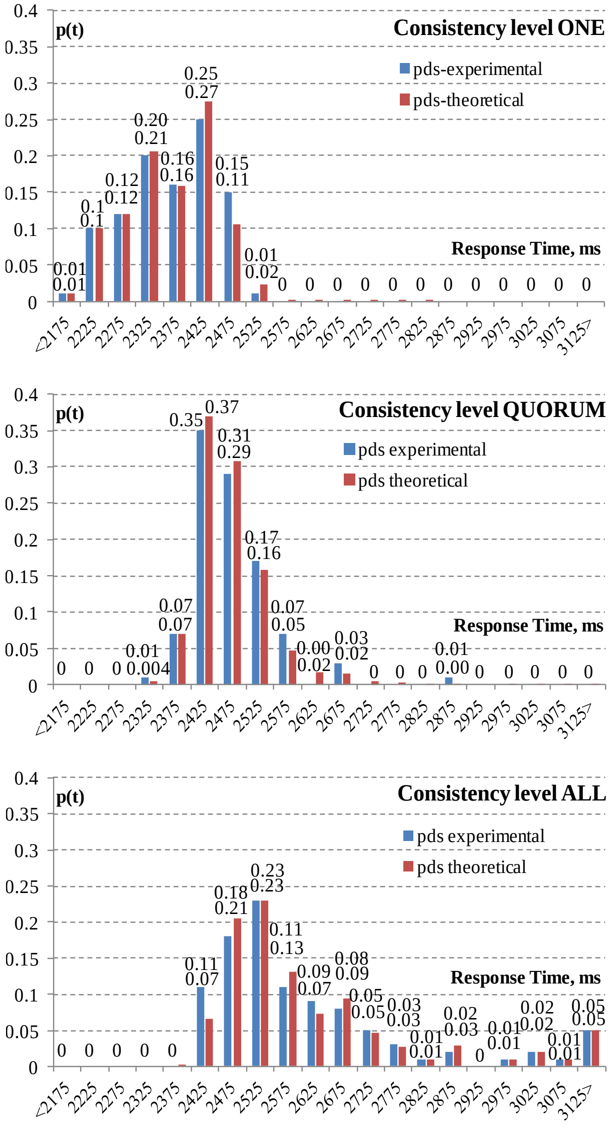


Fig. 9. Probability histograms (*pds*) of system response time for different consistency levels, estimated experimentally and using models (9), (10) and (11).

As expected, when the system is configured to provide consistency level ONE, its latency is on average lower than the average response time of the fastest replica. When the system provides consistency level ALL, the average system latency is larger than the average response time of the slowest replica. System latency associated with consistency level QUORUM is in the middle. However, our main observation is that it is hardly possible to make an accurate prediction of the average system latency corresponding to a certain consistency level when only the common statistical measures of replicas response time (i.e. the minimal, maximal and average estimates and standard deviation) are known.

This finding resulting from the massive statistical data gathered during our current and previous (e.g. [14, 16]) experiments is in line with the work of other researches [15, 39]. It shows that it is extremely difficult to predict the timing characteristics of various types of wide-area distributed systems, including fault-tolerant SOAs, distributed databases and file systems (e.g. Cassandra, GFS, HDFS), parallel processing systems (e.g. Hadoop Map-Reduce).

The dynamic and changing nature of timing characteristics of such systems can be better captured by employing probability density functions. In the next section we propose a probabilistic modelling approach that addresses this problem. It relies on using continuous and discrete probability density functions (*pdf*) of replica response times to predict system latency at different consistency levels.

6 Probabilistic Models of System Response Time for Different Consistency Levels

6.1 Deduction of Probability Density Function of System Response Time

In the section we propose a set of probabilistic models that allow us to build a combined probability density function of system response time by taking into account the required consistency level and incorporating response time probability density functions for each replica. Once we get the combined probability density function of system response time we can predict system latency using (5) and (6).

When the system is configured to provide consistency level ALL, the probability of returning response at time *t* is equal to the probability that one of the replicas returns its response <u>exactly at time *t*</u>, i.e. $g_1(t)$ while two other replicas return their responses <u>not later than *t*</u> (by time *t*), i.e. $\int_0^t g_2(t) = G_2(t)$ and $\int_0^t g_3(t) = G_3(t)$.

So far, as we have three replicas, all three possible combinations have to be accounted. As a result, the probability density function of the system response time for consistency level ALL can be defined as following:

$$f_{ALL}(t) = g_1(t)G_2(t)G_3(t) + g_2(t)G_1(t)G_3(t) + g_3(t)G_1(t)G_2(t) \tag{7}$$

where $g_1(t)$, $g_2(t)$ and $g_3(t)$ – are response time probability density functions of the first, second and third replicas respectively; $G_1(t)$, $G_2(t)$ and $G_3(t)$ – are response time cumulative distribution functions of the first, second and third replicas respectively.

When the system is configured to provide consistency level ONE, the probability of returning a response to the client at time *t* is equal to the probability that if only one of the replicas (e.g.

the first one) returns its response exactly at time *t*, i.e. $g_1(t)$, while two other replicas return their responses at the same time or later on, i.e. $\int_t^\infty g_2(t) = 1 - G_2(t)$ and $\int_t^\infty g_3(t) = 1 - G_3(t)$.

Keeping in mind three possible combinations we can deduce the probability density function of the system response time for consistency level ALL as:

$$\begin{aligned} f_{ONE}(t) = g_1(t)\big(1 - G_2(t)\big)\big(1 - G_3(t)\big) + \\ + g_2(t)\big(1 - G_1(t)\big)\big(1 - G_3(t)\big) + \\ + g_3(t)\big(1 - G_1(t)\big)\big(1 - G_2(t)\big) \end{aligned} \tag{8}$$

Deducing the response time probability density function for the QUORUM consistency level is based on a combination of the previous two cases. The probability of returning response to the client at time *t* is equal to the probability that one of the replicas returns its response exactly at time *t*; one of the two remained replicas returns its response by time *t* and another one responds at time *t* or later on. Taking into account all possible combinations the probability density function of the system response time for consistency level QUORUM can be deduced as:

$$\begin{aligned} f_{QUORUM}(t) = \big(g_1(t)G_2(t) + g_2(t)G_1(t)\big)\big(1 - G_3(t)\big) + \\ + \big(g_1(t)G_3(t) + g_3(t)G_1(t)\big)\big(1 - G_2(t)\big) + \\ + \big(g_2(t)G_3(t) + g_3(t)G_2(t)\big)\big(1 - G_1(t)\big) \end{aligned} \tag{9}$$

Using similar reasoning it is possible to deduce response time probability density functions of a system composed of *n* replicas:

$$f_{ALL}^{n}(t) = \sum_{i=1}^{n} \left(\frac{g_i(t)}{G_i(t)} \cdot \prod_{j=1}^{n} G_j(t) \right) \tag{10}$$

$$f_{ONE}^{n}(t) = \sum_{i=1}^{n} \left(\frac{g_i(t)}{1 - G_i(t)} \cdot \prod_{j=1}^{n} \big(1 - G_j(t)\big) \right) \tag{11}$$

It is difficult to build a general form of the probability density function of the system response time for consistency level QUORUM. However, the general reasoning is as following. The composed probability density function should be presented as a sum of *m* items, where *m* is a number of *k*-combinations of *n* (*k* is a number of replicas constituting a quorum). Each of the *m* items is a product of two factors. The first one defines the probability that a particular combination of *k* replicas returns responses by time *t*. Another factor defines the probability that the remaining (*n*–*k*) replicas return their responses after *t*.

6.2 Using Discrete Form of Probability Density Functions

Probability density function is a useful means of probabilistic uncertainty representation. Its continuous form allows calculating the probability of getting response from a system by any given time, as it was demonstrated in the previous work [40].

Though, finding theoretical distributions of replicas and system response times, as described in [16], includes non trivial statistical checks and mathematical transformations. Existing mathematical tools (e.g. Matlab, R, MathCAD, etc.) help to simplify this calculation even though they are costly and too ‘heavy’ to be used for run-time optimization. Besides, sometimes known theoretical distributions cannot approximate measured data with an adequate accuracy.

Replacing continuous probability density function with its discrete form (i.e. the probability density series, *pds*) is important for practical application of the proposed models. The probability distribution series of response time is a list of probabilities associated with each of the defined time intervals. The more time intervals are defined and the narrower they are, the closer approximation is provided.

Using reasoning similar to that in Section 6.1, we can define the discrete probability density functions of the response time for the three-replicated system depending on the chosen consistency level – see (12), (13) and (14).

$$P_{ALL}[i] = p_1[i]\sum_{j=1}^{i} p_2[j]\sum_{j=1}^{i} p_3[j] + p_2[i]\sum_{k=0}^{i-1} p_1[k]\sum_{j=1}^{i} p_3[j] + \\ p_3[i]\sum_{k=0}^{i-1} p_1[k]\sum_{k=0}^{i-1} p_2[k], \tag{12}$$

$$P_{ONE}[i] = p_1[i]\cdot\sum_{j=i}^{n+1} p_2[j]\cdot\sum_{j=i}^{n+1} p_3[j] + p_2[i]\sum_{k=i+1}^{n+1} p_1[k]\sum_{j=i}^{n+1} p_3[j] + \\ p_3[i]\sum_{k=i+1}^{n+1} p_1[k]\sum_{k=i+1}^{n+1} p_2[k], \tag{13}$$

$$\begin{aligned} P_{QUORUM}[i] = {} & \Big(\big(p_1[i]\cdot\sum_{j=0}^{i-1} p_2[j] + p_2[i]\cdot\sum_{j=0}^{i-1} p_1[j]\big)\cdot\sum_{k=i+1}^{n+1} p_3[j] + p_1[i]p_2[i]\Big) + \\ & + \Big(\big(p_1[i]\cdot\sum_{j=0}^{i-1} p_3[j] + p_3[i]\cdot\sum_{j=0}^{i-1} p_1[j]\big)\cdot\sum_{k=i+1}^{n+1} p_2[j] + p_1[i]p_3[i]\Big) + \\ & + \Big(\big(p_2[i]\cdot\sum_{j=0}^{i-1} p_3[j] + p_3[i]\cdot\sum_{j=0}^{i-1} p_2[j]\big)\cdot\sum_{k=i+1}^{n+1} p_1[j] + p_2[i]p_3[i]\Big) - \\ & - 2p_1[i]p_2[i]p_3[i], \;\; where\ i \in 1..n;\ p_x[0] = 0;\ p_x[n+1] = 0 \end{aligned} \tag{14}$$

Fig. 9 shows a significant closeness between experimentally measured *pds* and theoretical *pds* obtained with the help of the proposed models.

Probability density series can be directly estimated from the experimentally measured response time [41]. A possible Java implementation of finding replicas *pds* at run time is shown in Fig. 10. There we define such variables:

rt – is the measured replica response time for the current invocation [ms];

n – is the total number of the defined time intervals;

```
rt = getReplicaResponseTime();
m++;
if (rt < leftbound) {
   // rt is in interval [0..leftbound]
   num[1]++;
} else if (rt>rightbound) {
   // rt is in interval [rightbound..infinity]
   num[n]++;
} else {
   i = rt / delta - leftbound / delta + 2;
   num[i]++;
}
// estimation of discrete pdf of response time
for (i=1; i<=n; i++) {
   p[i] = num[i]/m;
}
```

Fig. 10. Practical estimation of replica response time *pds* using run-time measures

leftbound, *rightbound* – are boundaries, defining the first time interval [0..*leftbound*] and the last one [*rightbound*..∞];

num[*i*] – is the number of *rt* measures fallen in *i*th time interval, $i \in 1..n$;

p[*i*] – is the estimated probability of *rt* being in *i*th time interval, $i \in 1..n$;

delta – is the interval width [ms];

m – is the total number of *rt* measures.

7 MODELS VERIFICATION

In this section we check the validity and accuracy of the proposed models by comparing their prediction with the experimental data presented in Section 5. This check includes the following four steps:

- finding out theoretical distributions that accurately approximate the measured replica response times;
- applying the proposed mathematical models (7), (8) and (9) to deduce probability density functions of the system response time for different consistency levels;
- estimating the average replica and system response times using the theoretical probability density functions;
- comparing the theoretical and experimental values of the average replica and system response times.

7.1 Finding Theoretical Distribution Laws of Replica Response Times

The accuracy of theoretical modelling depends a lot on the adequacy of the distribution functions selected to approximate replicas response time. A guidance of finding theoretical distribution laws approximating replica response times can be found in [16]. It is based on performing a series of hypotheses checks [42]. The techniques of hypothesis testing consist of the two basic procedures. First, the values of distribution parameters are estimated by analysing an experimental sample. Second, the null hypothesis that experimental data has a particular distribution with certain parameters should be tested.

To perform hypothesis testing itself we used the `kstest` function: `[h,p]=kstest(t,cdf)`, conducting the Kolmogorov-Smirnov test to compare the distribution of *t* with the hypothesized distribution defined by *cdf*. The null hypothesis for the Kolmogorov-Smirnov test is that *t* has a distribution defined by *cdf*. The alternative hypothesis is that *x* does not have that distribution. Result *h* is equal to '1' if we can reject the hypothesis, or '0' if we cannot. The function also returns the *p*-value which is the probability that *x* does not contradict the null hypothesis. We reject the hypothesis if the test is significant at the 5% level (if *p*-value is less than 0.05).

The *p*-value returned by `kstest` was used to estimate the goodness-of-fit of the hypothesis. As a result of hypothesis testing we found out that the *Weibull* distribution fits well the response time of the first (Oregon) and the third (Tokyo) replicas. The response time of the second replica (Sao Paulo) can be accurately approximated by the *Gamma* distribution.When the commonly used probability density functions like Weibull or Gamma are not able to approximate the experimental data with the sufficient accuracy, the distribution fitting for heavy-tailed delays in the Internet can be done using more sophisticated Phase-type distribution [43].

7.2 Deducing Probability Density Functions of System Response Time

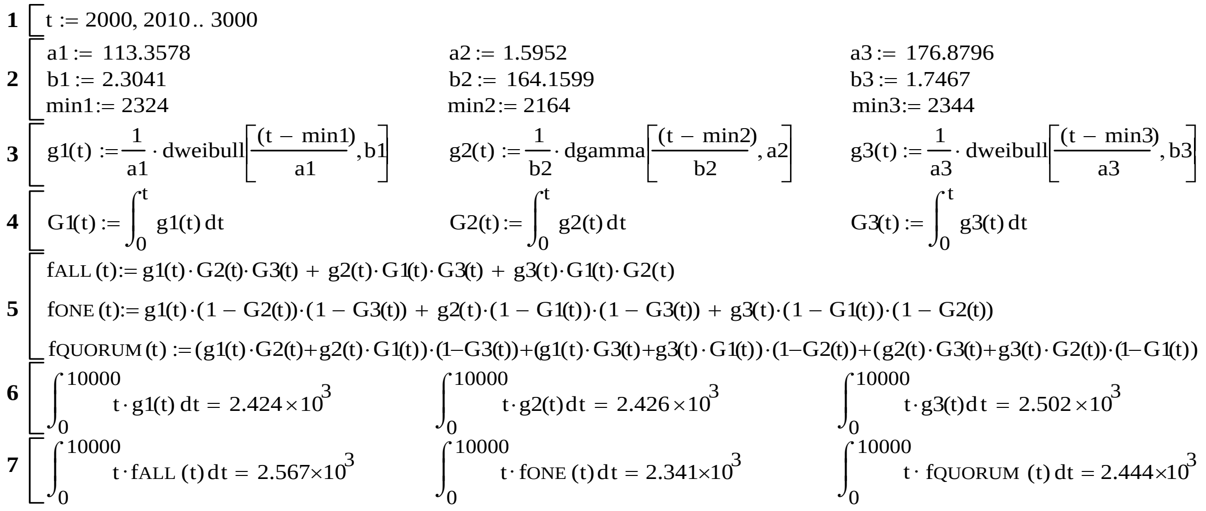
MathCAD has been used to deduce theoretical distributions of system response times for different consistency levels. It also allows to estimate the average system latency and to plot probability density functions. The MathCAD worksheet is shown in Fig. 11. It includes seven modelling steps.

At the 1st step we define abscissa axis *t* and its dimension in milliseconds. Secondly, we set up parameters of replicas response time distribution functions estimated in Matlab and also their shifts on the abscissa axis (i.e. minimal response time values).

At the 3rd and 4th steps the replica response time probability density functions $g_1(t)$, $g_2(t)$, $g_3(t)$ and the corresponding cumulative distribution functions $G_1(t)$, $G_2(t)$, $G_3(t)$ are defined using MathCAD library functions *dweibull* and *dgamma*.

At the 5th step we define probability density functions of the system response time corresponding to different consistency levels by combining replicas probability density functions *pdf* and cumulative distribution functions *cdf* according to (7), (8) and (9). Probability density functions of replicas and system response times are shown in Figs. 12 and 13. The bulk of the values of probability density function $f_{ALL}(t)$ is shifted to the right on the abscissa axis as it was expected. The shapes of the $f_{ONE}(t)$ and $f_{QUORUM}(t)$ probability density functions are also in line with the reasonable expectations and experimentally obtained probability density series (see Fig. 9). It is worth noting that the $f_{ONE}(t)$ showed 'camel' humped because of a considerably high influence of the second (the fastest) replica which *pdf* g2(t) is shifted significantly to the left on the time axis as compared to g1(t) and g3(t).

Finally, at steps 6 and 7 we estimate the average system and replicas response time by integrating their theoretical probability density functions.



1 $t := 2000, 2010 .. 3000$

2 $a1 := 113.3578$ $\quad a2 := 1.5952$ $\quad a3 := 176.8796$

$b1 := 2.3041$ $\quad b2 := 164.1599$ $\quad b3 := 1.7467$

$min1 := 2324$ $\quad min2 := 2164$ $\quad min3 := 2344$

3 $g1(t) := \frac{1}{a1} \cdot \mathrm{dweibull}\left[\frac{(t - min1)}{a1}, b1\right]$ $\quad g2(t) := \frac{1}{b2} \cdot \mathrm{dgamma}\left[\frac{(t - min2)}{b2}, a2\right]$ $\quad g3(t) := \frac{1}{a3} \cdot \mathrm{dweibull}\left[\frac{(t - min3)}{a3}, b3\right]$

4 $G1(t) := \int_0^t g1(t)\,dt$ $\quad G2(t) := \int_0^t g2(t)\,dt$ $\quad G3(t) := \int_0^t g3(t)\,dt$

5 $f_{ALL}(t) := g1(t) \cdot G2(t) \cdot G3(t) + g2(t) \cdot G1(t) \cdot G3(t) + g3(t) \cdot G1(t) \cdot G2(t)$

$f_{ONE}(t) := g1(t) \cdot (1 - G2(t)) \cdot (1 - G3(t)) + g2(t) \cdot (1 - G1(t)) \cdot (1 - G3(t)) + g3(t) \cdot (1 - G1(t)) \cdot (1 - G2(t))$

$f_{QUORUM}(t) := (g1(t) \cdot G2(t) + g2(t) \cdot G1(t)) \cdot (1 - G3(t)) + (g1(t) \cdot G3(t) + g3(t) \cdot G1(t)) \cdot (1 - G2(t)) + (g2(t) \cdot G3(t) + g3(t) \cdot G2(t)) \cdot (1 - G1(t))$

6 $\int_0^{10000} t \cdot g1(t)\,dt = 2.424 \times 10^3$ $\quad \int_0^{10000} t \cdot g2(t)\,dt = 2.426 \times 10^3$ $\quad \int_0^{10000} t \cdot g3(t)\,dt = 2.502 \times 10^3$

7 $\int_0^{10000} t \cdot f_{ALL}(t)\,dt = 2.567 \times 10^3$ $\quad \int_0^{10000} t \cdot f_{ONE}(t)\,dt = 2.341 \times 10^3$ $\quad \int_0^{10000} t \cdot f_{QUORUM}(t)\,dt = 2.444 \times 10^3$

Fig. 11. The MathCAD worksheet

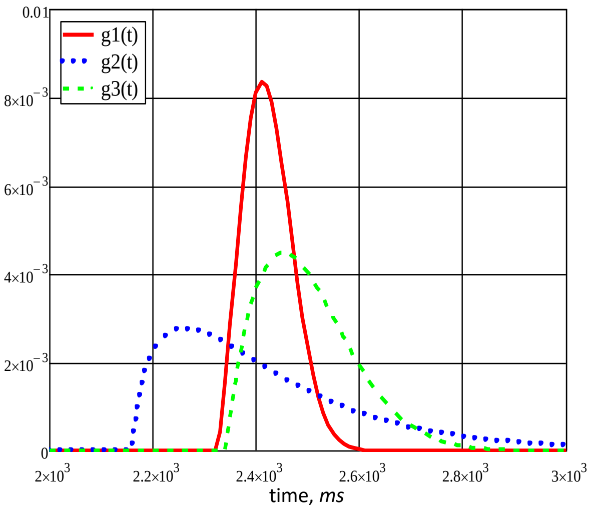


Fig. 12. Probability density functions of replicas response times

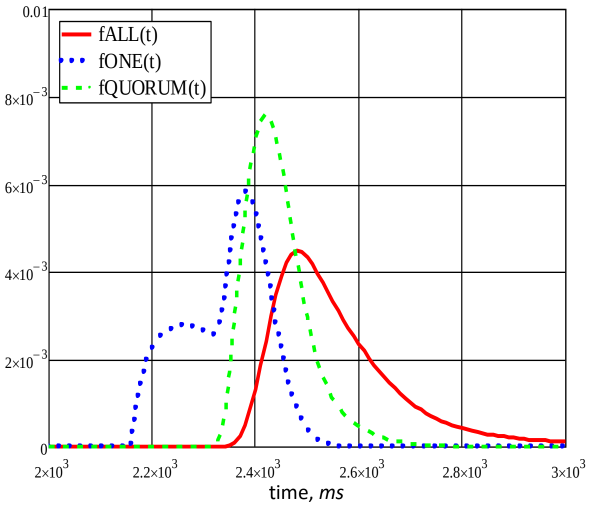


Fig. 13. Probability density functions of system response time for different consistency levels

7.3 Accuracy of Mathematical Modelling

Table 3 shows the deviation between the average values of 3-replicated system and replicas response time calculated for real data (see Tables 1 and 2) and by means of the obtained probability density functions. These results suggest that the proposed modelling techniques of timing characteristics are sound. To be certain that not only the average value can be accurately predicted we compare theoretical system probability density functions (see Fig. 13) and practically obtained probability density series (Fig. 9).

With this purpose we estimated experimental and theoretical probabilities that system latency at different consistency levels is less than the specified time. The results of this comparison, presented in [40] (see Table 3), show a close approximation of the experimental data by the proposed analytical models, especially for consistency levels ONE and QUORUM. The probabilistic model of the system response time for consistency level ALL gives slightly optimistic prediction, though the average deviation from the experimental data is only 2.7% in case of using *pds* and 3.5% if *pdf* is used which is considerably low.

TABLE 3. ACCURACY OF MATHEMATICAL MODELLING

	Replica1 (Oregon)	Replica2 (Sao Paulo)	Replica3 (Tokyo)	System consistency		
				ONE	QUORUM	ALL
Approximating theoretical distributions and their parameters						
	Weibull	Gamma	Weibull			
alpha	113.3578	1.5952	176.8796			
beta	2.3041	164.1599	1.7467			
x-shift	2324	2164	2344			
Average response time estimation, ms						
Measured	2428	2434	2588	2342	2449	2660
Modelled with *pdf*	2424	2426	2502	2341	2444	2567
deviation	0.18%	0.34%	3.32%	0.03%	0.19%	3.51%
Modelled with *pds*	2427	2430	2517	2339	2446	2589
deviation	0.06%	0.17%	2.76%	0.03%	0.12%	2.69%

8 MODELLING SYSTEMS WITH MULTIPLE REPLICAS

Sometimes, different replicas can have similar timing characteristics so that their response times can be approximated by the same distribution function. This can happen, for instance, if multiple replicas are deployed in the same public or private data centre and run on similar hardware with the standard operating environment. For these generic replicas equations (10) and (11) can be simplified as following:

$$f_{ALL}^{n}(t) = n \cdot g(t) \cdot G(t)^{n-1}, \tag{15}$$

$$f_{ONE}^{n}(t) = n \cdot g(t) \cdot \left(1 - G(t)\right)^{n-1}. \tag{16}$$

Besides, it becomes possible to define a probability density function of the system response time for consistency level QUORUM:

$$f_{QURUM}^{n}(t) = k \cdot C_{n}^{n-k} \cdot g(t) \cdot G(t)^{k-1} \cdot \left(1 - G(t)\right)^{n-k}, \text{ where } k = \left\lfloor \frac{n}{2} + 1 \right\rfloor. \tag{17}$$

The results of this comparison (see Table 4) show a close approximation of the experimental data by the proposed analytical models, especially for the consistency levels ONE and QUORUM. In turn, the cumulative distribution functions of system response time for different consistency levels can be explicitly defined as:

$$F_{ALL}^{n}(t) = G(t)^{n}, \tag{18}$$

$$F_{ONE}^{n}(t) = \left(1 - \left(1 - G(t)\right)^{n}\right), \tag{19}$$

$$F_{QUORUM}^{n}(t) = \sum_{k=\left\lfloor \frac{n}{2}+1 \right\rfloor}^{n} G(t)^{k} \cdot \left(1 - G(t)\right)^{n-k}. \tag{20}$$

Note that the derived models for system response time are similar to those estimating reliability of series, parallel and majority voting systems [44]. In the rest of this Section we demonstrate the applicability of the proposed models for predicting latency of systems with multiple replicas and estimating the optimal replication factor. As a generic replica for our simulation, we selected Replica2 deployed in South America (Sao Paulo), whose response time is characterised by the largest uncertainty of the three investigated in Section 7. It was shown that the response time of Replica2 can be approximated by the *Gamma* distribution:

$$g_{Replica2}(t) = \frac{1}{164.1599} \cdot dgamma\left(\frac{t-2164}{64.1599}, 1.5952\right). \tag{21}$$

Using MathCAD to substitute (21) into (15)–(17) we are able to derive system response time *pdf* for different consistency levels depending on the number of replicas.

Table 4 presents estimated values of system average response time and its standard deviation. As shown in Fig. 14, the QUORUM setup demonstrates convergent oscillations of the average system response time around the average replica response time. At the same time the average system response time increases considerably if a system is configured to provide the strongest consistency.

The standard deviation of system response time gradually decreases for the ONE and QUORUM consistency levels. For the ALL consistency level, the standard deviation increases at the beginning and reaches its maximal value when the replication factor becomes equal 10. After that its value gradually decreases.

TABLE 4. AVERAGE SYSTEM RESPONSE TIME

Replication factor	Average response time, ms			Standard deviation, ms		
	ALL	ONE	QUORUM	ALL	ONE	QUORUM
1	2425.87	2425.87	2425.87	207.34	207.34	207.34
2	2534.21	2317.56	2534.21	221.45	116.13	221.45
3	2602.62	2277.70	2397.35	226.09	83.88	129.71
4	2652.48	2256.49	2452.76	228.13	66.99	135.51
5	2692.06	2242.70	2388.89	229.22	56.45	101.41
6	2723.40	2233.22	2426.17	229.83	49.19	104.71
7	2752.03	2226.19	2384.74	230.14	43.83	85.90
8	2776.04	2220.72	2412.95	230.33	39.72	88.10
9	2797.33	2216.45	2382.50	230.43	36.43	75.81
10	2816.40	2212.94	2405.04	230.48	33.74	77.42
11	2833.71	2209.95	2380.78	230.48	31.49	68.58
12	2849.56	2207.42	2399.77	230.47	29.58	69.83
13	2864.15	2205.15	2379.01	230.42	27.94	63.10
14	2877.67	2203.14	2395.84	230.40	26.50	64.09
15	2890.46	2201.49	2378.60	230.33	25.24	58.76
16	2902.07	2200.16	2392.83	230.26	24.12	59.55
17	2913.18	2198.97	2377.91	230.23	23.12	55.17
18	2923.68	2197.85	2390.74	230.16	22.22	55.86
19	2932.72	2196.98	2378.02	230.09	21.42	52.18
20	2942.71	2196.07	2388.80	230.05	20.69	52.77

Taking into account the Central Limit Theorem we can assume that if there is a further increase in the number of replicas, the average response time of the system which provides the strongest consistency becomes normally distributed regardless of the replicas distributions (see Fig. 15). At the same time, if a system is configured to provide the weakest consistency, its response time demonstrates a tendency to become a deterministic variable that approaches the minimal observed value of replicas response time (see Fig. 16).

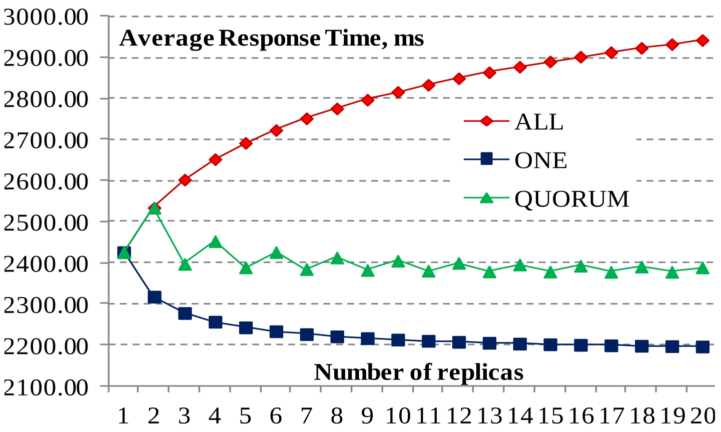


Fig. 14. Response time dependency on a number of generic replicas

Probabilistic models can help in estimating the optimal replication factor in distributed systems. If a system has to provide strong consistency, the maximal acceptable number of replicas which still guarantees that the system response time does not exceed a certain value with the required probability P^{req} can be calculated using (18) as follows:

$$n_{ALL} = \lfloor \log_{G(response\_time)} P^{req} \rfloor. \tag{22}$$

For systems that do not have consistency constraints, the minimal number of replicas required to reduce the response time to a certain value with the required probability P^{req} can be derived from (19):

$$n_{ONE} = \lceil \log_{1-G(response\_time)}(1 - P^{req}) \rceil. \tag{23}$$

To apply these techniques the system engineer should first define the desired system response time and the required probability of getting the response by this time P^{req}. Secondly, the probability that a generic replica returns the response by that time, i.e. *G*(*response_time*), has to be estimated using known *pdf* or *cdf*. Finally, after applying the corresponding equation, the obtained value (e.g. number of replicas) has to be rounded up to the integer value for the consistency level ONE or down for the consistency level ALL.

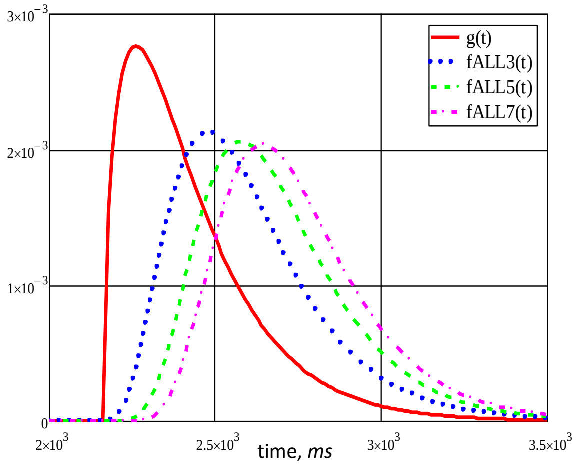


Fig. 15. Probability density functions of system response time for consistency level ALL and different replication factors (3, 5, 7)

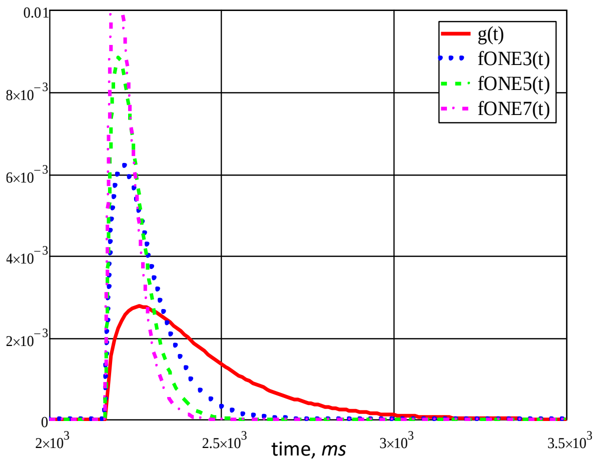


Fig. 16. Probability density functions of system response time for consistency level ONE and different replication factors (3, 5, 7)

9 Practical Application of the Proposed Theories

A set of the proposed time-probabilistic models provides a crucial support for predicting dependability and timing characteristics of globally-distributed fault-tolerant systems. The process of predicting system availability and timing characteristics includes four steps:

Monitoring the response time of system replicas.

1. Finding the continuous theoretical distributions of replica response times using either the technique, proposed in [16], or a practical estimation of response time probability density series (see Fig. 10) which is a discrete form of *pdf*.
2. Deducing the probability density function/series of the system response time by using the analytical models proposed in Section 6. At this step users and system providers are able to trade-off system latency versus consistency by making use of models (7)–(14).
3. Estimating system availability (probability of getting response from a system until the specified timeout) and timing characteristics by making use of the deduced *pdf/pds* of the system response time and models (5) and (6). The proposed models allow users and system providers to trade-off system availability versus latency by the optimal timeout setup.

The uniqueness of the proposed approach is that it allows predicting system latency, availability and consistency during system design and trading-off these characteristics at run-time. Besides, models (22) and (23) will help to calculate the optimal number of replicas to meet timing constraints.

10 Conclusion and Discussion

When employing fault-tolerance techniques over the Internet and clouds, engineers need to deal with delays, their uncertainty, timeouts, adjudication of asynchronous replies from replicas, and other issues specific to global distributed systems. The overall aim of this work is to introduce a time-probabilistic failure model and to study the impact of consistency on system latency in fault tolerant Internet computing. The proposed failure model and mathematical equations can help in choosing the right application timeouts which are the fundamental part of all distributed fault tolerance mechanisms working over the Internet and used as the main error detection mechanism here. With the help of the proposed models software developers can solve a trade-off problem between maximizing the probability of a correct servicing and minimizing the latency of a distributed system.

Our experimental results clearly show that improving system consistency makes system latency worse. This finding confirms one of the generally accepted qualitative implications of the CAP theorem [21, 22]. However, so far system developers have not had any mathematical tools to help them to accurately predict the response time of large-scale replicated systems.

While estimating the system worst-case execution time remains common practice for many applications (e.g. embedded computer systems, server fault-tolerance solutions, like STRATUS, etc.), this is no longer a viable solution for the wide-area service-oriented systems in which components can be distributed all over the Internet.

In our previous works [14, 16] we demonstrated that extreme unpredictable delays exceeding the value of ten average response times can happen in such systems quite often. In this paper we have proposed a set of novel analytical models providing a *quantitative basis* for the system response time prediction depending on the timeout settings and the consistency level provided for (or requested by) clients. The models allow us to derive the probability density function of the system response time which corresponds to a particular consistency level by incorporating the probability density functions of the replica response times. The validity of the proposed models has been verified against the experimental data reported in Section 7. It has been demonstrated that the proposed models ensure a significant level of accuracy in the system average response time prediction, especially in case of ONE and QUORUM consistency levels. The proposed models provide a mathematical basis for predicting latency of distributed fault and intrusion tolerance techniques operating over the Internet and clouds. The models take into account the probabilistic uncertainty of replicas' response time and the required consistency level.

The practical application of our work is in allowing practitioners to predict performance of service-oriented systems, and in offering them a crucial support in setting up the optimal timeout and replication factor and in understanding the trade-off between system consistency and latency. Trading off system latency against availability and consistency requires the knowledge of probability density functions that accurately approximate replicas' response time. These probabilistic characteristics, which can be obtained by testing or during the trial, will need to be corrected at run-time or at tune-time to improve prediction accuracy.

We have demonstrated that it is possible to use both continuous and discrete forms of response time probability density functions to accurately predict system latency (i.e. average response time). Although a continuous *pdf* allows calculating a confidence probability of getting response from a system by any given time, using discrete probability density series is easier in practice. It does not require complex calculations or the use of the third party tools like Matlab and MathCAD, which is important for run-time optimisation.

The proposed models could be also applied in the context of edge and fog computing [45] where the client interacts over time with multiple replicas located in different data centers, either as a result of application partitioning, or client mobility. Besides, they will help developers of distributed data storages to quantify how different consistency settings affect the system latency. Understanding this trade-off is also a key for the effective usage of modern NoSQL solutions [35].

Large-scale distributed systems composed out of a significant number of Internet services and their replicas ('particles' of this 'infinite' Internet 'universe') has strong resemblance with the theoretical Quantum Physics fundamentals of the atomic-level universe, including the Heisenberg uncertainty principle [46]. Introduced in 1927 the principle states that the more precisely the position of a particle is determined, the less precisely its momentum can be known, and vice versa.

The analogy between the latency/consistency probabilistic space of replicated distributed services and the atomic particle position/momentum continuum, includes a similar calculus of response times (vs electron position/momentum) based on temporal probability distributions and a similar view on the intrinsic uncertainty between the latency of client requests and distributed system consistency (vs the known Heisenberg's uncertainty principle).

This paper discusses a framework which shows that systems' latency and consistency cannot be simultaneously and accurately determined due to the uncertainty of highly distributed replicated systems. Table 2 reports our experimental results which show that the weakest consistency setting ONE causes the lowest response time on average which is characterised by low uncertainty (i.e. a standard deviation of the response time). Vice versa, the strongest consistency setting ALL causes the highest latency and the largest uncertainty. This relation becomes stronger with the increase of a number of replicas used.

Thus, by following the above analogy, our experimental and theoretical results demonstrate that the more certain data are (i.e. the higher level of consistency is chosen which reduces the probability of reading stale data), the less certain the latency of a replicated system is (i.e. the higher its variance is) and vice versa.

Ultimately, we believe this work could pave a way to studying the similarities between the intrinsic processes happening in ubiquitous massive-scale Internet computing systems and the Nuclear Physics, where a large number of experiments is typically necessary to uncover new phenomena and to understand the foundational theories.

ACKNOWLEDGMENTS

We would like to thank the editor of the journal and the reviewers for their helpful comments and feedbacks. We are grateful to our students Yuhui Chen, Batyrkhan Omarov, Vitaliy Ruban, and Seyran Mamutov for help with running some of the experiments.

REFERENCES

[1] J. Condliffe, “Amazon’s $150 Million Typo Is a Lightning Rod for a Big Cloud Problem,” 3 March 2017. [Online]. Available: https://www.technologyreview.com/s/603784/amazons-150-million-typo-is-a-lightning-rod-for-a-big-cloud-problem/.

[2] L. Chen, “Microservices: Architecting for Continuous Delivery and DevOps,” in *The IEEE International Conference on Software Architecture (ICSA'2018),* Seattle, USA, 2018.

[3] Y. Chawathe and E. Brewer, “System support for scalable and fault tolerant Internet services,” *Distributed Systems Engineering,* vol. 6, no. 1, pp. 23-33, 1999.

[4] V. Cardellini, E. Casalicchio, K. Branco, J. Estrella and F. Monaco, Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions, Hershey, Pennsylvania, USA: IGI Global, 2012.

[5] Z. Amin, N. Sethi and H. Singh, “Review on Fault Tolerance Techniques in Cloud Computing,” *Int. Journal of Computer Applications,* vol. 116, no. 18, pp. 11-17, 2015.

[6] Y. Izrailevsky and C. Bell, “Cloud Reliability,” *IEEE Cloud Computing,* vol. 5, no. 3, pp. 39-44, 2018.

[7] S. Gill and R. Buyya, “Failure Management for Reliable Cloud Computing: A Taxonomy, Model and Future Directions,” *Computing in Science & Engineering,* no. Early Access, pp. 1-10, 2018.

[8] M. Alshayeji, M. Al-Rousan, E. Yossef and H. Ellethy, “A Study on Fault Tolerance Mechanisms in Cloud Computing,” *International Journal of Computer Electrical Engineering,* vol. 10, no. 1, pp. 62-71, 2018.

[9] P. Garraghan, R. Yang, Z. Wen, A. Romanovsky, J. Xu, R. Buyya and R. Ranjan, “Emergent Failures: Rethinking Cloud Reliability at Scale,” *IEEE Cloud Computing,* vol. 5, no. 5, pp. 12-21, 2018.

[10] F. Alturkistani and S. Alaboodi, “An Analytical Model for Availability Evaluation of Cloud Service Provisioning System,” *International Journal of Advanced Computer Science and Applications,* vol. 8, no. 6, pp. 240-247, 2017.

[11] M. Vargas-Santiago, S. Pomares-Hernandez, L. Morales Rosales and H. Hadj-Kacem, “Survey on Web Services Fault Tolerance Approaches Based on Checkpointing Mechanisms,” *Journal of Software,* vol. 12, no. 7, pp. 507-525, 2017.

[12] R. Behera and K. H. K. Reddy, “Modeling and assessing reliability of service-oriented internet of things,” *International Journal of Computers and Applications,* vol. 41, no. 3, pp. 195-206, 2019.

[13] P. Lee and T. Anderson, Fault Tolerance. Principles and Practice, Wien - New-York: Springer-Verlag, 1990, p. 320.

[14] A. Gorbenko, A. Romanovsky, O. Tarasyuk, V. Kharchenko and S. Mamutov, “Exploring Uncertainty of Delays as a Factor in End-to-End Cloud Response Time,” in *9th European Dependable Computing Conference (EDCC’2012),* Sibiu, Romania, 2012.

[15] O. Bakr and I. Keidar, “Evaluating the running time of a communication round over the Internet,” in *21th Annual ACM Symposium on Principles of Distributed Computing (PODS'2002),* Monterey, California, 2002.

[16] Y. Chen, A. Gorbenko, A. Romanovsky and V. Kharchenko, “Measuring and Dealing with the Uncertainty of the SOA Solutions,” in *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions,* V. Cardellini, Ed., Hershey, USA, IGI Global, 2011, p. 265–294.

[17] P. Reinecke, A. van Moorsel and K. Wolter, “Experimental Analysis of the Correlation of HTTP GET invocations,” in *European Performance Engineering Workshop (EPEW’2006),* Budapest, Hungary, 2006.

[18] R. Potharaju and N. Jain, “When the Network Crumbles: An Empirical Study of Cloud Network Failures and their Impact on Services,” in *4th ACM Symposium on Cloud Computing (SOCC'2013),* Santa Clara, CA, 2013.

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

[20] A. Gorbenko, A. Romanovsky, O. Tarasyuk and V. Kharchenko, “Dependability of Service-Oriented Computing: Time-Probabilistic Failure Modelling,” in *Software Engineering for Resilient Systems: Lecture Notes in Computer Science (LNCS),* vol. 7527, P. Avgeriou, Ed., Berlin, Springer, 2012, pp. 121-133.

[21] E. Brewer, “Towards Robust Distributed Systems,” in *19th Annual ACM Symposium on Principles of Distributed Computing,* Portland, USA, 2000.

[22] S. Gilbert and N. Lynch, “Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services,” *ACM SIGACT News,* vol. 33, no. 2, pp. 51-59, 2002.

[23] L. Lamport, R. Shostak and M. Pease, “Byzantine Generals Problem,” *ACM Transactions on Programming Languages and Systems,* vol. 4, no. 3, pp. 382-401, 1982.

[24] D. Smith and K. Simpson, Safety Critical SystemsHandbook: A Straightforward Guide to Functional Safety, IEC 61508 and Related Standards, Oxford: Butterworth-Heinemann, 2016, p. 288.

[25] Z. Zheng, H. Ma, M. Lyu and I. King, “WSrec: A collaborative filtering based web service recommender system,” in *IEEE 7th International Conference on Web Services (ICWS’2009),* Los Angeles, CA, 2009.

[26] A. Van Moorsel and K. Wolter, “Analysis of Restart Mechanisms in Software Systems,” *IEEE Transactions on Software Engineering,* vol. 32, no. 8, pp. 547-558, 2006.

[27] J. Brutlag, “Speed Matters for Google Web Search,” Google, Inc., 22 June 2009. [Online]. Available: http://services.google.com/fh/files/blogs/google_delayexp.pdf. [Accessed 01 07 2019].

[28] E. Brewer, “CAP twelve years later: How the "rules" have changed,” *Computer,* vol. 45, no. 2, pp. 23-29, 2012.

[29] A. Lakshman and P. Malik, “Cassandra: a decentralized structured storage system,” *ACM SIGOPS Operating Systems Review,* vol. 44, no. 2, pp. 35-40, 2010.

[30] D. Abadi, “Consistency Tradeoffs in Modern Distributed Database System Design,” *IEEE Computer,* vol. 45, no. 2, pp. 37-42, 2012.

[31] A. Gorbenko and A. Romanovsky, “Time-Outing Internet Services,” *IEEE Security & Privacy,* vol. 11, no. 2, pp. 68-71, 2013.

[32] F. Dabek, M. Kaashoek, D. Karger, R. Morris and I. Stoica, “Wide-area cooperative storage with CFS,” *ACM SIGOPS Operating Systems Review,* vol. 35, no. 5, pp. 202-215, 2001.

[33] Z. M., K. Shen and J. Seiferas, “Replication Degree Customization for High Availability,” *ACM SIGOPS Operating Systems Review,* vol. 42, no. 4, pp. 55-68, 2008.

[34] A. Gorbenko, V. Kharchenko and A. Romanovsky, “Using Inherent Service Redundancy and Diversity to Ensure Web Services Dependability,” in *Methods, Models and Tools for Fault Tolerance, Lecture Notes in Computer Science (LNCS),* vol. 5454, M. Butler, C. Jones, A. Romanovsky and E. Troubitsyna, Eds., Berlin, Springer, 2009, pp. 324-341.

[35] Y. Mansouri, A. Toosi and R. Buyya, “Data Storage Management in Cloud Environments: Taxonomy, Survey, and Future Directions,” *ACM Computing Surveys,* vol. 50, no. 6, pp. 91:1-91:51, 2018.

[36] A. Tanenbaum and M. Van Steen, Distributed systems: Principles and Paradigms, Pearson Prentice Hall, 2006, p. 704.

[37] Amazon Web Services, “MongoDB on AWS: Guidelines and Best Practices,” April 2016. [Online]. Available: https://aws.amazon.com/s3/faqs/. [Accessed 01 07 2019].

[38] T. Baker, “The 3-2-1 Rule for Cloud Backup,” 31 October 2018. [Online]. Available: https://www.keepitsafe.com/blog/post/3-2-1-rule-for-cloud-backup/. [Accessed 01 July 2019].

[39] J. Rao, E. Shekita and S. Tata, “Using Paxos to Build a Scalable, Consistent, and Highly Available Datastore,” *Proceedings of the VLDB Endowment,* vol. 4, no. 4, pp. 243-254, 2011.

[40] O. Tarasyuk, A. Gorbenko and A. Romanovsky, “The Impact of Consistency on System Latency in Fault Tolerant Internet Computing,” in *Distributed Applications and Interoperable Systems, Lecture Notes in Computer Science (LNCS),* vol. 9038, Berlin, Springer, 2015, pp. 179-192.

[41] R. Walpole, R. Myers, S. Myers and K. Ye, Probability and Statistics for Engineers and Scientists, Pearson, 2010, p. 816.

[42] G. Privitera, Statistics for the Behavioral Sciences, SAGE Publications, 2015, p. 736.

[43] P. Reinecke and K. Wolter, “Phase-Type Approximations for Message Transmission Times in Web Services Reliable Messaging,” in *Performance Evaluation: Metrics, Models and Benchmarks. Lecture Notes in Computer Science (LNCS),* vol. 5119, 2008, pp. 191-207.

[44] E. Elsayed, Reliability Engineering, Wiley, 2012, p. 792.

[45] P. Varshney and Y. Simmhan, “Demystifying Fog Computing: Characterizing Architectures, Applications and Abstractions,” in *IEEE 1st International Conference on Fog and Edge Computing (ICFEC'2017),* Madrid, Spain, 2017.

[46] W. Heisenberg, “Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik,” *Zeitschrift für Physik,* vol. 43, no. 3-4, p. 172–198, 1927.