IEICE TRANS. COMMUN., VOL.E101–B, NO.8 AUGUST 2018

INVITED PAPER *Special Section on Autonomous Decentralized Systems Technologies and Approaches Innovation through Structure Change of Society and Life*

Facilitating Dynamic RT-QoS for Massive-Scale Autonomous Cyber-Physical Systems*

David W. McKEE[†], **Xue OUYANG**[†], ***and*** **Jie XU**[†a)], ***Nonmembers***

SUMMARY With the evolution of autonomous distributed systems such as smart cities, autonomous vehicles, smart control and scheduling systems there is an increased need for approaches to manage the execution of services to deliver real-time performance. As Cloud-hosted services are increasingly used to provide intelligence and analytic functionality to Internet of Things (IoT) systems, Quality of Service (QoS) techniques must be used to guarantee the timely service delivery. This paper reviews state-of-the-art QoS and Cloud techniques for real-time service delivery and data analysis. A review of straggler mitigation and a classification of real-time QoS techniques is provided. Then a mathematical framework is presented capturing the relationship between the host execution environment and the executing service allowing the response-times to predicted throughout execution. The framework is shown experimentally to reduce the number of QoS violations by 21% and provides alerts during the first 14*ms* provide alerts for 94% of future violations.

key words: *cloud, SOA, services, autonomous systems, real-time, straggler, IoT, IoS, simulation*

1. Introduction

With the rapid rise of large-scale autonomous systems as part of the era of Internet of Things (IoT) [1]; Internet of Simulation (IoS) [2]; Edge, Cloud and Fog computing; as well as Big Data with Deep Learning and high performance computing (HPC) there is desperate need to develop techniques to dynamically manage the execution performance of intelligent and data processing services. As these intelligent cyber-physical systems become pervasive through domains of manufacturing, healthcare, transport, and power management [3] the supporting services must provide on-demand decision support in a timely and dependable fashion [4]. This paper therefore presents a framework to model the temporal execution behaviour of these services and looks at the impact on data processing for these domains.

As the technologies within each of these domains have advanced allowing integration as System of Systems (SoS) there remain significant limitations and constraints due to the performance requirements within each domain which are not guaranteed across the entire spectrum. Therefore in order to develop techniques to handle the explosion of the big data streams [5] in a timely fashion new techniques bringing together real-time stream processing [6], straggler mitigation [7], and QoS prediction [8] must developed. Furthermore with the emergence of the IoS paradigm where simulations are deployed as intelligence services (SIMaaS) [2] interacting with in-the-loop systems — such as hardware, model, or human — the Service Level Agreement (SLA) and QoS must provide real-time guarantees. The resulting action or data may therefore be incorrect or unsafe [9].

In this paper we look at two major areas of service performance management: Real-Time QoS (RT-QoS) prediction and straggler mitigation. A review of existing approaches is outlined and a mathematical framework for online QoS is detailed and implemented. This framework takes into account the underlying host resources such as CPU, memory, and network bandwidth to model the response-time behaviour under real world circumstances. The framework manages the allocation of resources to reduce QoS violation and provides warnings during potential violations. Experimental results demonstrate a 94% of violations can be predicted within the first 14% of their execution time.

The rest of the paper is as follows: in Sect. 2 the motivation for real-time SoS integration is presented. In Sect. 3 the state-of-the-art is studied before the framework is presented in Sect. 4. Experiment results are shown in Sect. 5 before conclusions are drawn in Sect. 6.

2. Background and Motivation

With the paradigm shift in the computing landscape over the recent years towards distributed computing, low powered IoT devices [10], and the availability of cloud computing [11], as well as the increased usage of simulation in both engineering and intelligent services domains [12] traditionally isolated domains are beginning to merge and interact brining numerous challenges. With this the already exponential growth of data will become evermore rapid whilst needing to be processed rapidly [3].

This section introduces autonomous cloud-based systems, smart cities, and connected autonomous vehicles followed by a review of QoS approaches and challenges.

Manuscript received September 7, 2017.
Manuscript revised December 22, 2017.
Manuscript publicized February 22, 2018.

[†]The authors are with the School of Computing, Faculty of Engineering, University of Leeds, UK.

*This work is based on "Massive-Scale Automation in Cyber-Physical Systems: Vision & Challenges", by D. McKee et al. which appeared in Proc. IEEE International Symposium on Autonomous Decentralized Systems (ISADS 2017), Bangkok, Thailand, March 2017, ©2017 IEEE and on "*n*-Dimensional QoS Framework for Real-Time Service-Oriented Architectures", by D. McKee et al. which appeared in Proc. IEEE International Symposium on Real-time Data Processing for Cloud Computing, June 2017, ©2017 IEEE.

a) E-mail: j.xu@leeds.ac.uk

DOI: 10.1587/transcom.2017ADI0001

2.1 Autonomous Cloud-Based Systems

With the recent advances in cooperative robotics towards autonomous systems [13], the combination of cloud computing with robotics [3], and the development of augmented reality [14] it is anticipated that within the next 10-15 years there will be ubiquitous and intelligent computing systems managing and augmenting most of the systems we interact with on a daily basis. These massive-scale cyber-physical systems will have to trade-off user experience and computational efficiency and make use of techniques for massive-scale data processing for autonomous decision making systems.

2.1.1 Smart Cities and Autonomous Vehicles

The first and most prominent domain is that of smart cities which can be considered as a "*cyber-physical System of Systems (SoS) heavily reliant on intelligent autonomy, distributed computing, IoT and IoS such that it brings together technology, governance, and society to manage and monitor power and communication infrastructure, the environment, traffic and other aspects of the city for the benefit and well-being of its inhabitants through ubiquitous sensing and embedded intelligence, and facilitates economic growth through innovation, connectivity and data aggregation*" [3].

In this context services hosted primarily in the cloud provide decision support and vital software functionality to the city in the way that drivers support computer operating systems [15]. Specifically these may include robotics for repair and maintenance [16], driverless transportation [17] and power management [18] among others.

In terms of transportation, the current drive to reach level-5 autonomy with connected autonomous vehicles [19] cloud-based services will form the basis of a *Vehicular Cloud* [20]. In this context these services will provide augmented reality to the vehicles to improve decision making on-board the vehicles. A particular example would be simultaneous localization and mapping (SLAM) which used in both vehicle autonomous driving systems as well as robotic arm planning. These systems require image and sensor data to be rapidly processed and matched against massive datasets in a timely manner.

2.2 QoS and Fault-Propagation

Underpinning service-orientation and QoS in particular are the concepts of dependability [21]. In the domain of Service Oriented Architecture (SOA) Bruning et al. [22] provide a taxonomy of faults and their propagation through a system. A subset of this taxonomy is shown in Fig. 1. With regards to publishing faults the service descriptions should present the expected level of QoS which can be used to define a SLA. Many technologies do not facilitate this in the service description semantics. Further unless there are mechanisms to guarantee that the described QoS is accurate there may be a mismatch between the service implementation and the advertised description. A fault of this type may result in a faulty workflow composition that is unable to meet the specified SLA. During binding if the service description is incorrect the system may bind to the wrong service. Each of these may cause the workflow producing an incorrect result.

QoS properties are typically specified using the WSLA or WS-Agreement standards in XML format where each property is named, has a type, and has a value typically expressed numerically using a double. For example if a *performance* parameter could be specified as a *double* with a *response time* metric specified as a *double* representing *seconds*.

Additionally most prevalent to individual service QoS, and specifically the focus of this paper, are the challenges relating to execution timing due to server crashes and communication failures. This can be extended with additional detail for the specific description fault whereby the specified response-time can not be delivered due to resource limitations within either the host server or across the network. These additions are shown in Fig. 1 with the addition of five further fault classes spread across the major categories resulting in the final failure of the the QoS deadline not being adhered to [8]. The objective with dynamic QoS techniques is therefore to intercept the fault propagation before a failure is observed. This paper specifically looks at intercepting the

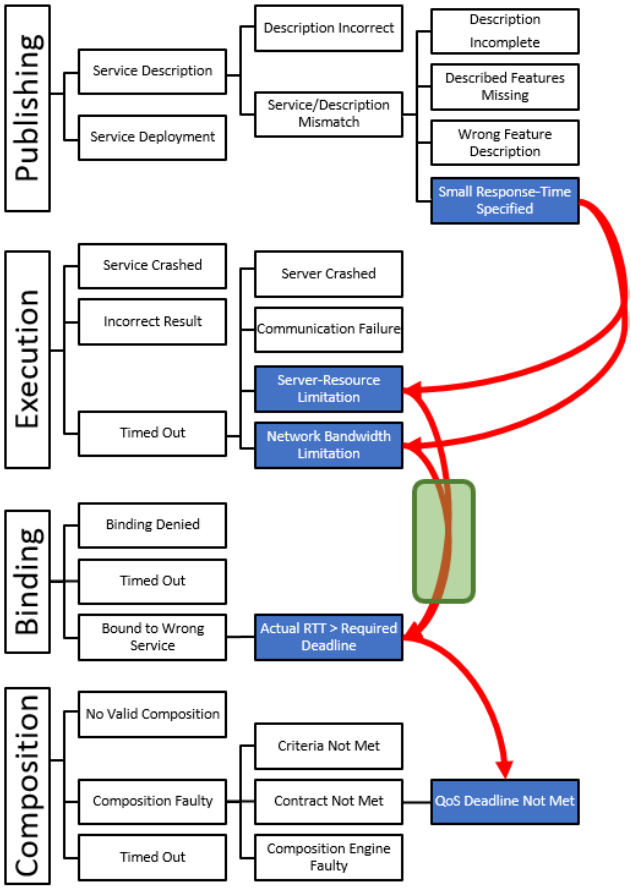


Fig. 1 Top three layers of the SOA fault tree with additional real-time faults added.

highlighted propagation caused by resource availability.

2.3 QoS Challenge: The Straggler Syndrome

Cloud computing [23] has emerged as a means to implement parallel applications on massive-scale commodity clusters, in which tasks are executed on multiple server nodes by systems that automatically provide scheduling, fault tolerance, and load balancing. MapReduce [24] framework pioneered this computing model, and systems like Hadoop YARN [25] and Spark [26] generalized its population. Despite the success, there are lots of challenges toward reliable and predictable service delivery within such systems, especially with increasing system scale and autonomous features. One such challenge is the straggler syndrome.

Straggler syndrome is used to define the phenomenon that occurs when a distributed job - composed of multiple tasks executing in parallel - incurs significant delay in completion due to a small subset of its parallelized tasks - known as stragglers - performing much slower than the other sibling ones [24]. After analyzing the data from a production cluster of Microsoft Bing, Mantri [29] claims that 80% of the stragglers have a uniform probability of being delayed by 150% to 250% compared to the medium task duration, while 10% take more than 10 times the median duration.

The QoS breakdown and the late timing failure [21] are the most explicit consequence if the rapidness of service response cannot be guaranteed [33]. Google measures from its Cloud service and report that, the slowest 5% responses is responsible for half of the total 99%-percentile latency, and the probability of longer duration increases in the face of system scale growth [34]. Straggler problem is against the purpose of parallel computing, which is to speed up job execution performce and ensure timing attributes of QoS can be fulfilled.

3. Existing Techniques

Given the challenges with regards to handling service faults in terms of QoS and also stragglers and their cascading impact of service and system performance this section details some of the prevalent existing contributions and techniques. Firstly a brief summary of the straggler mitigation techniques is presented and then in a classification and review of RT-QoS approaches is presented in Sect. 3.2.

3.1 Straggler Mitigation Approaches

Various straggler detection and mitigation approaches are developed over the last years, such as simple cloning [30], blacklisting [35] and speculation [27]. Among them, speculative execution is the dominant method type. It functions in a three-phase manner: firstly, identifies task stragglers, then launches redundant task copy for an identified straggler, and finally adopts whichever result that comes out first.

Representative speculative based variations include LATE [28], Mantri [29], Dolly [30], Adaptive Speculator [7], SkewTune [31], CREST [32], etc. Each of the related work has its own characteristic in terms of suitable target environments or straggler types. Table 1 illustrates the comparison details.

The *Speculative Execution Metrics* in Table 1 indicates how a specific method identifies stragglers. For example, the LATE [28] speculator will identify the task with the *L*ongest *A*pproximate *T*ime to *E*nd as the straggler. In other words, LATE cares about the estimated finish time of parallel tasks. For most methods, to estimate the duration of tasks forms the foundation for straggler mitigation.

$$ECT^t_{ji} = t + \frac{1 - PS^t_{ji}}{PS^t_{ji}}(t - t_0) \tag{1}$$

Current parallel jobs normally uses *Progress Score* (PS) of a task and corresponding elapsed time $(t - t_0)$ when calculating the estimated completion using Eq. (1) [7], [28], where PS is given in systems such as Hadoop and YARN (PS^t_{ij} is the PS for the i_{th} task in parallel job j at time stamp t). But they ignore the influence brought by resource availability and node heterogeneity, which should provide additional knowledge for a more accurate prediction of task completion time.

3.2 RT-QoS Approaches

Having reviewed over 80 existing approaches for RT-QoS, they can be categorised into the following groups with 84% neatly fitting the categories (also shown in Fig. 2):

1. **Correlation** These approaches primarily use Pearson's Correlation Coefficient and build on work Zheng et al. [36] where they predict QoS based on user experience. A *user-service* matric is used to identify the most similar users using collaborative filtering. These approaches do

Table 1 Various straggler mitigation approaches.

Comparison	[27]	[28]	[29]	[30]	[7]	[31]	[32]
Speculative Execution Metrics	Progress Score	Finish Time	t_{rem} and t_{new}	Simple Cloning	Finish Time	t_{rem} and t_{par}	Progress Rate
Server Node Heterogeneity	✗	✓	✓	✓	✓	✓	✓
Adaptive Straggler Threshold	✗	✗	✓	✗	✓	✗	✗
Dynamic Node Performance	✗	✗	✓	✗	✓	✓	✗
Cap on Speculation Number	✗	✓	✗	✓	✓	✓	✗
Data Skew Type of Straggler	✗	✗	✗	✗	✗	✓	✗

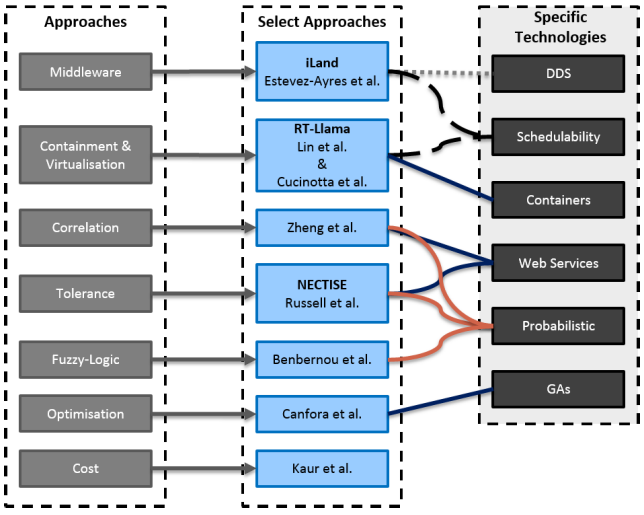


Fig. 2 Classification of existing major RT-QoS techniques.

not however support recalculation of the QoS definition and do not provide the mathematics to support time guarantees. 10% of approaches were correlation based.

2. **Optimisation** This is the second category where the QoS definition is evolved to provide a more accurate and representative definition. The majority of these techniques use genetic algorithms and often focus on service selection and initial definition specification rather than online adaptation. Work by the likes of Canfora et al. [37] does look at optimising the parametrisation of a QoS definition. 16% of reviewed approaches were some form of optimisation it is not applied to real-time QoS as the evolution requires numerous generations to reach even a reasonable definition and there is no guarantee that a satisfactory solution will be found within a specified timeframe.

3. **Fuzzy-logic** These approaches are currently not heavily adopted but have growing interest in many disciplines of computing. Within this topic they currently account for only 3% of approaches but do provide a very flexible approach and therefore can consider a wide range of situations and influencing factors. The work specifically be Benbernou et al. [38] looks at rating performance as either *good* or *bad* alongside *high*, *medium*, or *low* resource utilisation.

4. **Cost** 13% of approaches look specifically at the cost of running a service and many of the other approaches consider cost as a parameter in their definition. The most prominent work is by Kaur et al. [39] which looks at the trade-off between performance, power or energy, resource utilisation, and infrastructure pricing.

5. **Tolerance and Probability** These approaches look at tolerating, often using probabilistic methods, a level of service unreliability. These take into account the likelihood of timely service delivery and propose methods to cope with it rather than solve the problem. 22% of reviewed approaches fall in this category and they can roughly be split into redundancy and probe-based techniques. In the former case work such as Liu et al. [40] use n-versioning and n-copy based on a the probability of untimely service delivery. In the later case probes are used to monitor the response-times of services over time and a probabilistic model is built using that data.

6. **Containment** A further 10% of techniques fall into the category of using virtual machines or containers to manage the resources allocated and consumed by a service. These techniques lend themselves to Cloud hosted services which run on virtualised infrastructure. However they remain susceptible to interference on the host server with the possibility of stragglers. As such, situations requiring real-time performance still require the host servers to be running real-time operating systems [41].

7. **Middleware** Again looking at the underlying compute and communication infrastructure are approaches that require more fine grained control of the host systems. In this realm are some of the most prominent and earliest techniques for real-time SOA that all build on the Data Distribution Service (DDS) standard using the Publish/Subscribe pattern for communication [42] These 11% of approaches facilitate at least 21 standard parameters and if the entire system is managed exclusively by the technique can provide real-time formal guarantees of timely service delivery.

4. *n*-Dimensional QoS Framework

This section details the QoS framework that captures the relationship between response-times and the resource utilisation and availability in order to predict the *time to finish* for a service throughout its execution.

4.1 Mathematical Formalisms

Table 2 details the notation and constituent mathematical parts of the framework. This framework extends our previous mathematical work [8].

First we consider only micro-services $\mathbf{s}$, and define the set d_r, of available resources as a discrete range δ. We can therefore calculate the resource availability on the host as:

$$\alpha(\mathbf{h}(\mathbf{s}_n), \mathbf{s}_n)_{r,t} = 1 - \overbrace{(U(\mathbf{h}(\mathbf{s}_n))_{r,t}}^{\text{Host utilisation}} - \overbrace{U(\mathbf{s}_n)_{r,t})}^{\mu sutilisation} \quad (2)$$

And convert this into matrix coordinates:

$$j = \{r \in \mathcal{R} : \lfloor A(h, \mathbf{s}_n)_r \times |d_r| \rfloor\} \quad (3)$$

The model can then be defined as multi-dimensional space for each resource type and the dimension of time or execution progress. The model is populated by taking the resource

Table 2 QoS framework notation.

Symbol	Definition
α	The observed resource availability on the host
D	The Micro-Service or Service Deadline
d_r	Discrete set of resource values
f	Observation and monitoring frequency.
F	Forecast of the resources required until execution completes.
$\mathbb{H}$	Set of all Hosts
h	Host machine
j	The model coordinate values for each resource dimension.
k	The number of observation points.
Ω	Set of all observations for a given Micro-Service instance.
ω	A resource observation.
p	Execution Progress
$\mathcal{R}$	Set of all Resource Types
r	A resource type with a capacity and measure of performance
RTT	Response-Time
s	Micro-Service
$\mathcal{TTF}$	Time-to-Finish for Micro-Service execution.
U	The utilisation model.

Algorithm 1: Estimating Execution Progress

begin Estimate progress
  $p_{temp} = \infty$
  foreach *resource r in* $\mathcal{R}$ **do**
    if $I.Sum == 0$ **then** Initial Case
      $h = MIN(h, \mathbb{H}.Max)$
      $F[j,r] = MIN(U_{provided}, h \times \mathbf{RTT})$
    end
    $temp = FLOOR((k \times \Omega[r].Sum) \div F[j,r]) \div k$
    $p_{temp} = MIN(p_{temp}, temp)$
  end
  $p = MAX(p, p_{temp})$
end

observations ω throughout execution and calculating the total resources consumed over time U:

$$\{\forall r, \forall \omega \in \Omega(\mathbf{s}_n) : [\![\omega_r]\!]_{0..1}\} \xrightarrow{[\![\cdot]\!]_{ip}} \{u_r \in U(\mathbf{s})_j\} \quad (4)$$

A forecast F, is then available of the resource required to finish from a given execution progress point p.

The predictive model then during future service runtimes takes the observed availability α, calculates the index j, to get the relevant forecast F for the given progress p. The progress is estimated from the observed cumulative consumed resources Ω and the forecast model:

$$p(\mathbf{s})_t = \max\left\{p(\mathbf{s})_{t-1}, \min\left\{\frac{1}{k}\left\lfloor \frac{k \cdot \sum_{x=0}^{t} [\![\Omega(\mathbf{s})_{j,r,x}]\!]_{0..1}}{F(\mathbf{s})_{j,r}}\right\rfloor\right\}\right\} \quad (5)$$

And the time-to-finish $\mathcal{TTF}$ is estimated as being $\mathcal{TTF} = \left(1 - \frac{p}{k}\right)\mathbf{RTT}(\mathbf{s}_n)_j$.

The *p*score can then be fed into the speculative execution prediction progress (PS) in Eq. (1) or the estimated $\mathcal{TTF}$ can be used in place of the *estimated completion time* (ECT).

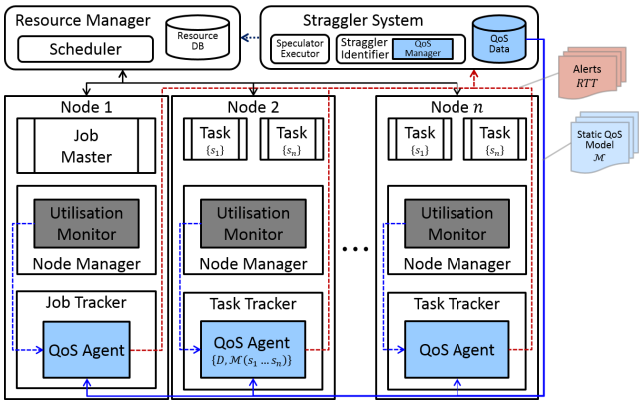


Fig. 3 Cloud architecture with RT-QoS agents.

Algorithm 2: Estimating the time-to-finish

if $I_j > 0$ **then** Standard Model
  $RTT = M[j]$
end
else if $I.Sum == 0$ **then** Initial Case
  $h = MIN(h, \mathbb{H}.Max)$
  $U = MIN(U_{provided}, h \times RTT)$
  $RTT = U \div j$
end
else Sparse Model
  begin Calculate $\mathcal{D}^{-1}$ from $j$of all points in the matrix
    foreach i **do**
      $D[i] = 1 \div ABS(i - j)$
    end
  end
  begin Calculate **RTT**
    $num = 0$
    $denom = 0$
    foreach $i \neq j$ **do**
      $num+ = I[j] \times M[i] \times D[i]$
      $denom+ = I[i] \times D[i]$
    end
    $RTT = num \div denom$
  end
end
$\mathcal{TTF} = (p \div k) \times \mathbf{RTT}$

4.2 System Architecture

The above mathematical formalisms can be represented algorithmically and deployed in the cloud architecture shown in Fig. 3 where agents send resource observations and up-to-date $\mathcal{TTF}$ predictions back to the execution monitor. This approach minimises communication to only alerts and observed execution data once execution has finished.

The framework operates in two phases with an *online* and *update* phase. In the former case the resource utilisation and availability is monitored to provide an updated *time-to-finish* prediction periodically over the execution duration.

In the online phase the execution progress is estimated using Algorithm 1 where the observed resource utilisation Ω is compared against the forecast F in each resource dimension. With the calculated availability coordinates j

this used to estimate the $\mathcal{TTF}$ in Algorithm 2. This uses the euclidean distance $\mathcal{D}^{-1}$ within the matrix model between resource configurations in the following form:

Algorithm 3: Deadline miss alert

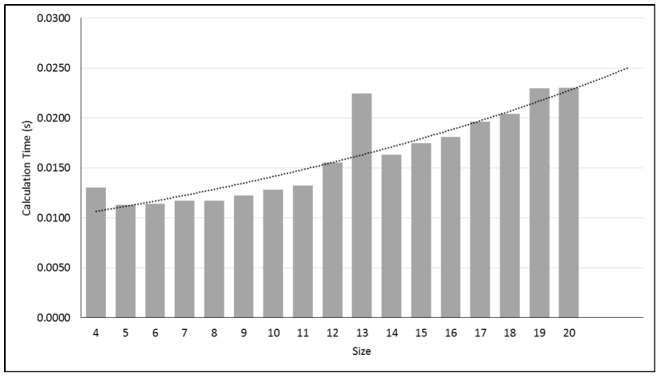
```
begin Initial deadline check
    if TTF > D then
        begin Re-configuration Check
            j_target = NULL
            foreach i >= j do
                TTF_temp =
                  ESTIMATE_TTF(i, p, Ω, [U, RTT, h])
                if TTF_temp <= D then
                    j_target = i
                    BREAK LOOP
                end
            end
        end
        if j_target = NULL then
            ALERT(NULL)
        end
        else Report the required configuration
            ALERT(j_target)
        end
    end
end
```

$\mathbf{RTT} = \frac{\sum(I \cdot \mathcal{M} \cdot \mathcal{D}^{-1})}{\sum(I \cdot \mathcal{D}^{-1})}$. Then if the $\mathcal{TTF}$ is greater than the required deadline $\mathbf{D}$ an alert is provided with the resource configuration that would be required to meet the deadline, as shown in Algorithm 3. The alert is then passed onto the speculation manager which creates appropriate replicas [7].

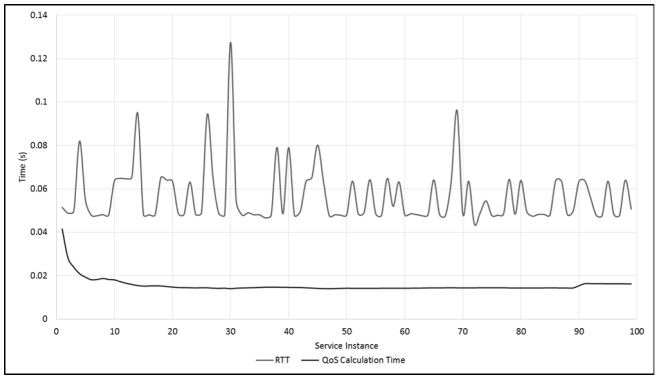
5. Experimental Results

In this section the metrics of QoS violation (Mean Percentage Violation (MPV)), over allocation Mean Percentage Waste (MPW), and absolute error (Mean Percentage Error (MPE)) are used whereby the defined QoS is compared against the actual response-time. Most **s**will provide mathematical or textual data processing functions such as a word-count or mathematical functions such as calculating derivatives, products, summations etc. Therefore for the purposes of this paper the experiments were conducted using 20 services solving Euler mathematical problems and repeated 100 times. Figure 5(a) shows the average observed response-times of the services with an overall average of 65 ms with a standard deviation of 26 ms.

Observations were taken with frequency $f = 13ms$ in a configuration with the resource fidelity $|d_r| = 4$ and $23ms$ with $|d_r| = 20$ providing an average of between 5 and 3

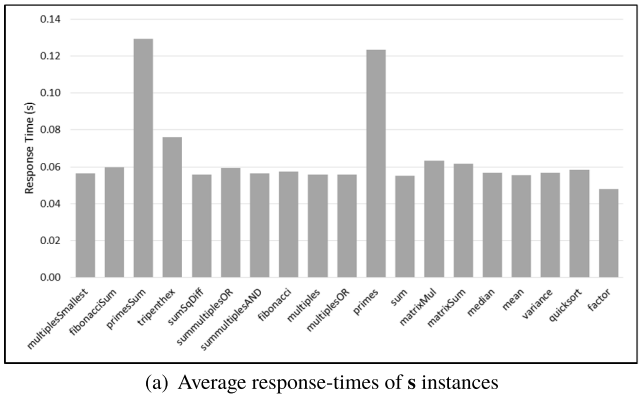


(a) Increasing QoS calculation time with resource fidelity

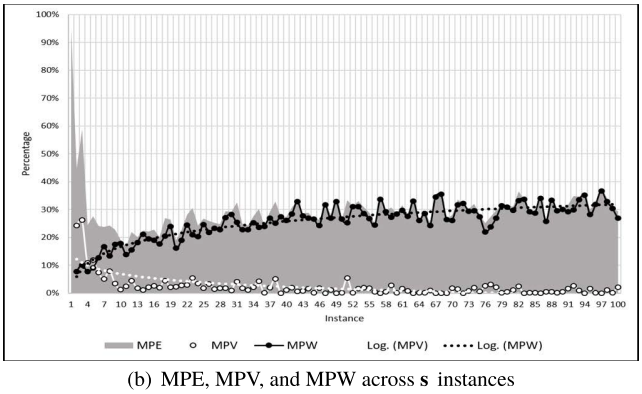


(b) QoS calcuation time for *Smallest Multiple* service and average calculation time across all services

Fig. 4 QoS calculation time.



(a) Average response-times of **s** instances



(b) MPE, MPV, and MPW across **s** instances

Fig. 5 Response-times, predictions, and error rate.

Table 3 Combined difference in QoS violation and waste between existing and proposed approaches.

		Historical	Correlation	Real-Time	Fuzzy-Logic
Waste	*Count*	1384	1488	1105	-954
	Mean	4.04%	15.65%	9324.83%	-32.81%
Violat	*Count*	-43	-3	316	529
	Mean	-1.84%	19.78%	34.71%	12.01%
MP Trade-off	*Hard-RT*	-1.8%	19.8%	34.7%	12.0%
	Firm-RT	0.1%	18.4%	3131.4%	-2.9%
	Soft-RT	2.1%	17.0%	6228.1%	-17.9%
	Not RT	4.0%	15.7%	9324.8%	-32.8%
Count Trade-off	*Hard-RT*	-43	-3	316	529
	Firm-RT	433	494	579	35
	Soft-RT	908	991	842	-460
	Not RT	1384	1488	1105	-954

observations per service execution respectively as shown in Fig. 4(a). Figure 4(b) depicts the execution time of the online algorithm to predict the $\mathcal{TTF}$. Most notably during the first 10 execution instances the calculation time is significantly slower as the approach must account for a sparse/empty data matrix.

Figure 5(b) depicts error-rates across all the service types where the resulting MPV is by less than 2%, representing a deadline miss of only 1 ms and occurring in less than 10% of service instances and the overallocation (MPW) is approaching 35%. With this configuration, with a resource fidelity of $|d_r| = 9$ alerts were provided within the first 14ms and alerts were raised for 94% of violations.

When comparing this against the existing techniques described earlier we see that the proposed method improves on each of the existing techniques. As can be seen in Table 3 the proposed method sees 21% less violations than either the real-time middleware (iLand method [43]) or fuzzy-logic techniques and overallocates by 10% less than either the probabilistic/historical or correlation based methods.

6. Conclusion

This paper has presented the need for a robust and automated technique for ensuring reliable and timely service delivery to support intelligent services for smart cities and autonomous vehicles. A background discussion around the challenges of providing such services across the heterogeneous compute platforms has been provided. Specifically the challenges of straggler mitigation and service Quality of Service (QoS) have been reviewed and an analysis of the most significant approaches has been provided and a classification for real-time QoS has been provided.

Then a multi-dimensional framework for real-time QoS has been detailed mathematically and algorithmically. It has been shown experimentally to reduce the number of QoS violations by 21% and reduce resource overallocation by 10%. Furthermore 94% of QoS violations were pre-emptively identified and alerts generated.

There is further work to evaluate the proposed QoS framework at larger scale across a range of compute platforms, including IoT devices, taking into account further dimensions such as network latency. Additionally combining the automated straggler mitigation techniques with this framework has the potential to provide a powerful framework for supporting real-time big data processing in a dependable fashion.

Acknowledgements

This work has been supported by Jaguar Land Rover, UK-EPSRC grant EP/K014226/1 and other grants including the China National Key Research and Development Program (No. 2016YFB1000101 and 20016YFB1000103).

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," Future Generation Computer Systems, vol.29, no.7, pp.1645–1660, 2013.

[2] D. McKee, S.J. Clement, X. Ouyang, J. Xu, R. Romanoy, and J. Davies, "The Internet of simulation, a specialisation of the internet of things with simulation and workflow as a service (SIM/WFaaS)," 2017 IEEE Symposium on Service-Oriented System Engineering (SOSE), IEEE, 2017.

[3] D. McKee, S.J. Clement, J. Almutairi, and J. Xu, "Massive-scale automation in cyber-physical systems: Vision & challenges," 2017 IEEE 13th International Symposium on Autonomous Decentralized System (ISADS), June 2017.

[4] D.W. McKee, D. Webster, and J. Xu, "Enabling decision support for the delivery of real-time services," 2015 IEEE 15th Int. Symp. High-Assurance Syst. Eng., IEEE, Jan. 2015.

[5] P. Zikopoulos, C. Eaton, et al., Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, McGraw-Hill Osborne Media, 2011.

[6] P. Garraghan, S. Perks, X. Ouyang, D. McKee, and I.S. Moreno, "Tolerating transient late-timing faults in cloud-based real-time stream processing," 2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC), July 2016.

[7] X. Ouyang, P. Garraghan, B. Primas, D. Mckee, P. Townend, and J. Xu, "Adaptive speculation for efficient internetware application execution in clouds," ACM Trans. Internet Technol., vol.18, no.2, 2018.

[8] D. McKee, S. Clement, J. Xu, and D. Battersby, "n-dimensional QoS framework for real-time service-oriented architectures," RTDPCC 2017: 2nd International Symposium on Real-time Data Processing for Cloud Computing, IEEE Computer Society Press, June 2017.

[9] M. Dooner, J. Wang, and A. Mouzakitis, "Development of a simulation model of a windshield wiper system for hardware in the loop simulation," Autom. Comput. (ICAC), 2013 19th Int. Conf., 2013.

[10] R. Buyya and A.V. Dastjerdi, Internet of Things: Principles and Paradigms, Elsevier Science & Technology, 2016.

[11] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, Special Publication 800-145, 2011.

[12] J.-R. Martinez-Salio and J.-M. Lopez, "Future of LVC simulation: Evolving towards the MSaaS concept," Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC), 2014.

[13] R. Doriya, S. Mishra, and S. Gupta, "A brief survey and analysis of multi-robot communication and coordination," Proc. Communication Automation Int. Conf. Computing, pp.1014–1021, May 2015.

[14] W. Barfield, Fundamentals of Wearable Computers and Augmented Reality, CRC Press, 2015.

[15] S. Clement, D.W. McKee, and J. Xu, "Service-oriented reference

architecture for smart cities," IEEE International Symposium on Service-Oriented System Engineering, IEEE, 2017.
[16] I. Gatsoulis et al., "Learning the repair urgency for a decision support system for tunnel maintenance," ECAI 2016: 22nd European Conference on Artificial Intelligence, G. Kaminka, M. Fox, P. Bouquet, et al., eds., Frontiers in Artificial Intelligence and Applications, no.285, pp.1769–1774, IOS Press, Amsterdam, Netherlands, Aug. 2016.
[17] Z. Xiong, H. Sheng, W. Rong, and D.E. Cooper, "Intelligent transportation systems for smart cities: A progress review," Science China Information Sciences, vol.55, no.12, pp.2908–2914, Nov. 2012.
[18] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," IEEE Internet Things J., vol.1, no.1, pp.22–32, 2014.
[19] U.P. House of Lords, "Connected and autonomous vehicles: The future?," 2017.
[20] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds," 2014 IEEE World Forum on Internet of Things (WF-IoT), IEEE, 2014.
[21] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," IEEE Trans. Dependable and Secure Computing, vol.1, no.1, pp.11–33, 2004.
[22] S. Bruning, S. Weissleder, and M. Malek, "A fault taxonomy for service-oriented architecture," 10th IEEE High Assurance Systems Engineering Symposium (HASE'07), pp.367–368, IEEE, 2007.
[23] R. Buyya, C.S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on, pp.5–13, IEEE, 2008.
[24] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Commun. ACM, vol.51, no.1, pp.107–113, 2008.
[25] V.K. Vavilapalli, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, E. Baldeschwieler, A.C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, and H. Shah, "Apache hadoop YARN: Yet another resource negotiator," Proc. 4th annual Symposium on Cloud Computing, p.5, ACM, 2013.
[26] M. Zaharia et al., "Spark: Cluster computing with working sets," HotCloud, vol.10, no.10-10, p.95, 2010.
[27] T. White, Hadoop: The Definitive Guide, O'Reilly Media, 2012.
[28] M. Zaharia et al., "Improving MapReduce performance in heterogeneous environments," OSDI, p.7, 2008.
[29] G. Ananthanarayanan et al., "Reining in the outliers in map-reduce clusters using Mantri," OSDI, p.24, 2010.
[30] G. Ananthanarayanan et al., "Effective straggler mitigation: Attack of the clones," NSDI, pp.185–198, 2013.
[31] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "SkewTune: Mitigating skew in mapreduce applications," Proc. 2012 ACM SIGMOD International Conference on Management of Data, pp.25–36, ACM, 2012.
[32] L. Lei, T. Wo, and C. Hu, "CREST: Towards fast speculation of straggler tasks in mapreduce," e-Business Engineering (ICEBE), 2011 IEEE 8th International Conference on, pp.311–316, IEEE, 2011.
[33] X. Ouyang et al., "Reducing late-timing failure at scale: Straggler root-cause analysis in cloud datacenters," Fast Abstracts in the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, 2016.
[34] J. Dean and L.A. Barroso, "The tail at scale," Commun. ACM, vol.56, no.2, pp.74–80, 2013.
[35] U. Kumar and J. Kumar, "A comprehensive review of straggler handling algorithms for mapreduce framework," International Journal of Grid and Distributed Computing, vol.7, no.4, pp.139–148, 2014.
[36] Z. Zheng, H. Ma, M.R. Lyu, and I. King, "QoS-aware Web service recommendation by collaborative filtering," IEEE Trans. Serv. Comput., vol.4, no.2, pp.140–152, April 2011.
[37] G. Canfora, M.D. Penta, R. Esposito, and M.L. Villani, "An approach for QoS-aware service composition based on genetic algorithms," Proc. 2005 conference on Genetic and evolutionary computation - GECCO'05, pp.1069–1075, ACM Press, New York, New York, USA, 2005.
[38] S. Benbernou et al., "Managing QoS acceptability for service selection: A Probabilistic description logics based approach," Proc. 28th International Workshop on Description Logics, 2015.
[39] T. Kaur, D. Kaur, and A. Aggarwal, "Cost model for software as a service," 2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence), pp.736–741, IEEE, Sept. 2014.
[40] L. Liu, D. Russell, D. Webster, Z. Luo, C. Venters, J. Xu, and J.K. Davies, "Delivering sustainable capability on evolutionary service-oriented architecture," 2009 IEEE Int. Symp. Object/Component/Service-Oriented Real-Time Distrib. Comput., pp.12–19, March 2009.
[41] K.J. Lin, M. Panahi, Y. Zhang, J. Zhang, and S.-H. Chang, "Building accountability middleware to support dependable SOA," IEEE Internet Comput., vol.13, no.2, pp.16–25, March 2009.
[42] W. Tsai, Y.-H. Lee, Z. Cao, Y. Chen, and B. Xiao, "RTSOA: Real-time service-oriented architecture," Service-Oriented System Engineering, 2006. SOSE'06. Second IEEE International Workshop, pp.49–56, IEEE, 2006.
[43] I. Estévez-Ayres, P. Basanta-Val, and M. García-Valls, "Composing and scheduling service-oriented applications in time-triggered distributed real-time Java environments," Concurr. Comput. Pract. Exp., vol.26, no.1, pp.152–193, Jan. 2014.

David McKee is a Research Fellow in the School of Computing, University of Leeds. He has industrial experience in large-scale automated systems and is particularly interested in the fields of distributed fault tolerant real-time systems with their effects on Industry 4.0 and the Internet of Simulation (IoS). He has been involved with IEEE HASE, SOSE, ISORC, etc. and leads developments on several research projects and publications.

Xue Ouyang is a Ph.D. student in the School of Computing, University of Leeds. She received her B.Eng. degree in Network Engineering and M.Eng. degree in Software Engineering from National University of Defense Technology, China. Her research interest lies in improving performance and efficiency for parallel jobs within large-scale distributed systems.

Jie Xu is Chair of Computing at the University of Leeds and Director of the UK EPSRC WRG e-Science Centre. He has industrial experience in building largescale networked systems and has worked in the field of dependable distributed computing for over 30 years. He is a Steering/Executive Committee member of IEEE SRDS, ISORC, HASE, SOSE, etc. and a co-founder of IC2E. He has led or co-led many research projects to the value of over $30M, and published over 300 research papers.