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

David W. MCKEE†, Xue OUYANG†, and Jie XU†, 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 being used to provide intelligence and analytics 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 be predicted throughout execution. The framework is shown experimentally to reduce the number of QoS violations by 21% and provides alerts during the first 14ms 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 a 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 Section 2 the motivation for real-time SoS integration is presented. In Section 3 the state-of-the-art is studied before the framework is presented in Section 4. Experiment results are shown in Section 5 before conclusions are drawn in Section 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 bringing 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.

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 wellbeing 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 Figure 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 Figure 1 with the addition of five further fault classes spread across the major categories resulting in the final failure of 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 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 siblings [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 performance 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 Section 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 Longest Approximate Time to End 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.

\[ ECT_{ji} = t + \frac{1 - PS_{ji}^t}{PS_{ji}^t}(t - t_0) \]  

Current parallel jobs normally uses Progress Score (PS) of a task and corresponding elapsed time \((t - t_0)\) when calculating the estimated completion using Equation 1 [28] [7], where PS is given in systems such as Hadoop and YARN \((PS_{ji}^t)\) 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 Figure 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 matrix is used to identify the most similar
users using collaborative filtering. These approaches do 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 $s$, and define the set $d_r$ of available resources as a discrete range $\delta$. We can therefore calculate the resource availability on the host as:

$$\alpha(h(s_n), s_n)_{r,t} = 1 - \frac{U(h(s_n))_{r,t} - U(s_n)_{r,t}}{U(s_n)_{r,t}}$$

And convert this into matrix coordinates:

$$j = \{r \in R : |A(h, s_n)_r \times |d_r|\}$$

The model can then be defined as multi-dimensional space
A forecast times takes the observed availability consumed resources dex (ECT) in Equation 1 or the estimated progress. The time-to-finish total resources consumed over time observations progress. The model is populated by taking the resource W. MCKEE et al.: FACILITATING DYNAMIC RT-QOS FOR MASSIVE-SCALE AUTONOMOUS CYBER-PHYSICAL SYSTEMS

\[ TTF = \text{Time-to-Finish for Micro-Service execution.} \]

Algorithm 1: Estimating Execution Progress

begin
    \( p_{\text{temp}} = \infty \)
    foreach resource \( r \) in \( R \) do
        if \( I_1 \cdot \text{Sum} \equiv 0 \) then Initial Case
        \[ h = \text{MIN}(h, \Xi, \text{Max}) \]
        \[ F[j, r] = \text{MIN}(U_{\text{provision}}, h \times \text{RTT}) \]
        end
        \( \text{temp} = \text{FLOOR}(k \times \Omega[r]), \text{Sum} \times F[j, r] + k \)
        \( p_{\text{temp}} = \text{MIN}(p_{\text{temp}}, \text{temp}) \)
        end
        \( p = \text{MAX}(p, p_{\text{temp}}) \)
end

for each resource type and the dimension of time or execution progress. The model is populated by taking the resource observations \( \omega \) throughout execution and calculating the total resources consumed over time \( U \):

\[ \{ \forall r, \forall \omega \in \Omega(s_i) : \| \omega_r \|_{0.1} \} \quad \| \|_{\text{p}} \quad \{ u_r \in U(s_i) \} \quad (4) \]

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 run-times takes the observed availability \( \alpha \), 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 \( \Omega \) and the forecast model:

\[ p(s_j) = \max \left\{ p(s_{j-1}, \min \left\{ \frac{1}{k} \cdot \sum_{r=0}^{\infty} \| \Omega(s_{i}, r, s) \|_{0.1} \right\} ) \right\} \quad (5) \]

And the time-to-finish \( TTF \) is estimated as being \( TTF = (1 - \frac{p}{R}) \text{RTT}(s_j) \).

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

### Table 2 QoS Framework Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma )</td>
<td>The observed resource availability on the host</td>
</tr>
<tr>
<td>( D )</td>
<td>The Micro-Service or Service Deadline</td>
</tr>
<tr>
<td>( d_r )</td>
<td>Discrete set of resource values</td>
</tr>
<tr>
<td>( f )</td>
<td>Observation and monitoring frequency.</td>
</tr>
<tr>
<td>( F )</td>
<td>Forecast of the resources required until execution completes.</td>
</tr>
<tr>
<td>( \Xi )</td>
<td>Set of all Hosts</td>
</tr>
<tr>
<td>( h )</td>
<td>Host machine</td>
</tr>
<tr>
<td>( j )</td>
<td>The model coordinate values for each resource dimension.</td>
</tr>
<tr>
<td>( k )</td>
<td>The number of observation points.</td>
</tr>
<tr>
<td>( \Omega )</td>
<td>Set of all observations for a given Micro-Service instance.</td>
</tr>
<tr>
<td>( \omega )</td>
<td>A resource observation.</td>
</tr>
<tr>
<td>( p )</td>
<td>Execution Progress</td>
</tr>
<tr>
<td>( R )</td>
<td>Set of all Resource Types</td>
</tr>
<tr>
<td>( r )</td>
<td>A resource type with a capacity and measure of performance</td>
</tr>
<tr>
<td>RTT</td>
<td>Response-Time</td>
</tr>
<tr>
<td>( s )</td>
<td>Micro-Service</td>
</tr>
<tr>
<td>TTF</td>
<td>Time-to-Finish for Micro-Service execution.</td>
</tr>
<tr>
<td>( U )</td>
<td>The utilisation model.</td>
</tr>
</tbody>
</table>

4.2 System Architecture

The above mathematical formalisms can be represented algorithmically and deployed in the cloud architecture shown in Figure 3 where agents send resource observations and up-to-date \( TTF \) 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 \( \Omega \) is compared against the forecast \( F \) in each resource dimension. With the calculated availability coordinates \( j \) this used to estimate the \( TTF \) in Algorithm 2. This uses the euclidean distance \( D^{-1} \) within the matrix model between resource configurations in the following form: \( \text{RTT} = \sum \left( I \cdot M \cdot D^{-1} \right) \). Then if the \( TTF \) is greater than the required deadline \( D \) 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 swill 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 65ms with a standard deviation of 26ms.
Observations were taken with frequency $f = 13\, ms$ in a configuration with the resource fidelity $|d_r| = 4$ and $23\, ms$ with $|d_r| = 20$ providing an average of between 5 and 3 observations per service execution respectively as shown in Figure 4(a). Figure 4(b) depicts the execution time of the online algorithm to predict the $T_{TFL}$. 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 1ms 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 $14\, ms$ 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 preemptively 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
Algorithm 2: Estimating the time-to-finish

\[
\text{if } I > 0 \text{ then Standard Model}
\]

\[
\begin{align*}
RTT &= M[j] \\
\end{align*}
\]

\[
\text{end}
\]

\[
\text{else if } I \text{. Sum } == 0 \text{ then Initial Case}
\]

\[
\begin{align*}
h &= MIN(h, E\text{. Max}) \\
U &= MIN(U_{\text{provided}}, h \times RTT) \\
RTT &= U + j \\
\end{align*}
\]

\[
\text{else Sparse Model}
\]

\[
\begin{align*}
&\text{begin Calculate } D^{-1} \text{ from } j \text{ of all points in the matrix}
\end{align*}
\]

\[
\begin{align*}
&\text{foreach } i \text{ do}
\end{align*}
\]

\[
\begin{align*}
&\text{ Calculate } D[i] = 1 \div \text{ABS}(i - j)
\end{align*}
\]

\[
\begin{align*}
&\text{end}
\end{align*}
\]

\[
\begin{align*}
&\text{begin Calculate RTT}
\end{align*}
\]

\[
\begin{align*}
&\text{num} = 0 \\
&\text{denom} = 0
\end{align*}
\]

\[
\begin{align*}
&\text{foreach } i \neq j \text{ do}
\end{align*}
\]

\[
\begin{align*}
&\text{num} += I[j] \times M[i] \times D[i]
\end{align*}
\]

\[
\begin{align*}
&\text{denom} += I[i] \times D[i]
\end{align*}
\]

\[
\begin{align*}
&\text{end}
\end{align*}
\]

\[
\begin{align*}
&RTT = \text{num} \div \text{denom}
\end{align*}
\]

\[
\begin{align*}
&TTF = (p \div k) \times RTT
\end{align*}
\]

Algorithm 3: Deadline miss alert

\[
\text{begin Initial deadline check}
\]

\[
\begin{align*}
&TTF > D \text{ then}
\end{align*}
\]

\[
\begin{align*}
&\text{begin Re-configuration Check}
\end{align*}
\]

\[
\begin{align*}
&j_{\text{target}} = \text{NULL}
\end{align*}
\]

\[
\begin{align*}
&\text{foreach } i \geq j \text{ do}
\end{align*}
\]

\[
\begin{align*}
&TTF_{\text{temp}} = \text{ESTIMATE}_{TTF}(i, p, Q, [U, RTT, h])
\end{align*}
\]

\[
\begin{align*}
&\text{if } TTF_{\text{temp}} < D \text{ then}
\end{align*}
\]

\[
\begin{align*}
&s_{\text{target}} = i
\end{align*}
\]

\[
\begin{align*}
&\text{BREAK LOOP}
\end{align*}
\]

\[
\begin{align*}
&\text{end}
\end{align*}
\]

\[
\begin{align*}
&\text{if } j_{\text{target}} = \text{NULL} \text{ then}
\end{align*}
\]

\[
\begin{align*}
&\text{ALER(T(NULL)}
\end{align*}
\]

\[
\begin{align*}
&\text{end}
\end{align*}
\]

\[
\begin{align*}
&\text{else Report the required configuration}
\end{align*}
\]

\[
\begin{align*}
&\text{ALER(T\text{[target]})}
\end{align*}
\]

\[
\begin{align*}
&\text{end}
\end{align*}
\]

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)

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

<table>
<thead>
<tr>
<th>Historical</th>
<th>Correlation</th>
<th>Real-Time</th>
<th>Fuzzy-Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Mean</td>
<td>Count</td>
<td>Mean</td>
</tr>
<tr>
<td>Violation</td>
<td>Waste</td>
<td>Violation</td>
<td>Waste</td>
</tr>
<tr>
<td>Hard-RT</td>
<td>-1.8%</td>
<td>19.8%</td>
<td>34.7%</td>
</tr>
<tr>
<td>Firm-RT</td>
<td>0.1%</td>
<td>18.4%</td>
<td>3131.4%</td>
</tr>
<tr>
<td>Soft-RT</td>
<td>2.1%</td>
<td>17.0%</td>
<td>6228.1%</td>
</tr>
<tr>
<td>Not RT</td>
<td>4.0%</td>
<td>15.7%</td>
<td>9324.8%</td>
</tr>
<tr>
<td>Count</td>
<td>Mean</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Traj-off</td>
<td>Hard-RT</td>
<td>-43</td>
<td>-3</td>
</tr>
<tr>
<td>Firm-RT</td>
<td>433</td>
<td>494</td>
<td>579</td>
</tr>
<tr>
<td>Soft-RT</td>
<td>908</td>
<td>991</td>
<td>842</td>
</tr>
<tr>
<td>Not RT</td>
<td>1384</td>
<td>1488</td>
<td>1105</td>
</tr>
</tbody>
</table>

References