Identifying and Estimating Technical Debt for Service Composition in SaaS Cloud

Satish Kumar and Rami Bahsoon
School of Computer Science
University of Birmingham
Birmingham, U.K.
{s.kumar.8,r.bahsoon}@cs.bham.ac.uk

Tao Chen
Department of Computing and Technology
Nottingham Trent University
Nottingham, U.K
tao.chen@ntu.ac.uk

Rajkumar Buyya
School of Computing and Information
System, The University of Melbourne
Melbourne, Australia
rbuyya@unimelb.edu.au

***Abstract*—A composite service in multi-tenant SaaS cloud would inevitably operate under dynamic changes on the workload from the tenants, and thus it is not uncommon for the composition to encounter under-utilization and over-utilization on the component services. However, both of those cases could be good or bad: the former implies that although there is under-utilization, the pay-off afterwards are more significant; the latter, in contrast, refers to the over-utilization that leads to trivial pay-off, or nothing at all. Such a notion perfectly matches with the Technical Debt (TD) metaphor in Software Engineering. As a result, it is necessary to identify the root causes of the debts and where the debt can be manifested in the service composition, which, in turn, would offer great helps on the decision making process of service composition. In this paper, we propose a novel approach for identifying the technical debt in service composition under SaaS cloud. The approach combines time series forecasting and a newly proposed technical debt model to estimate the future debt and utility in the service composition. Through a real world case study, we demonstrate that our approach can successfully identify both the good and bad debts, while producing satisfactory accuracy on estimating the technical debt involved in the service composition under SaaS cloud.**

Index: Service composition, Technical debt, Service utility, Quality of Service, Multi-tenant.

I. Introduction

Service composition is a logical combination of multiple abstract services resulting into a single unit (e.g., an application) for performing complex requests submitted by the users in the multi-tenant SaaS cloud [1]. An abstract service can be realized by a set of candidate component services [1], each of which comes with different capacities to process n requests per second. However, uncertainty in workload generated by the tenants may affect the overall Quality of Service (QoS), and more importantly, it may cause under-utilization or over-utilization on the component services with respect to their capacities. Consequently, the operational cost could outweigh the service revenue or violates the tenants' Service Level Agreement (SLA), which may trigger a recomposition.

A service composition is sub-optimal from the utility point of view when the selected component services are under-/over-utilized during execution. While over-utilization generally implies negative impacts, under-utilization could be good or bad: if the pay off in the future is more significant, then we can temporally accept under-utilization; otherwise, the under-utilization would only incur unneeded cost. Such a notion perfectly matches with the Technical Debt (TD) metaphor [2][3] in Software Engineering. In particular, there may be an imperfect composition decision, leading to a new forms of technical debts that explicate this category of systems. The debt can be intentional; it can be due to recomposition plans that provide higher services capacity than what is currently demanded by the users. Technical debt could be incurred unintentionally in the service composition, for example, when a component service receives a high volume of requests workload, and the underlying component service cannot process all the requests,consequently, violating the SLA. In this example, the penalty costs against the response time violation and the eventuality recomposition costs can be viewed as interest on the debt for a given execution instance.

To better support the decision making process of service composition in SaaS cloud, in this paper, we propose an approach that combines technical debt metaphor and time series forecasting for identifying and estimating technical debt in service composition. Notably, we have made the following contributions:

- We tailor a time series forecasting method, namely ARFIMA model [4], into the debt model for estimating future debt.
- We propose a model that explicitly maps the concepts of TD in the contexts of service composition. Such a model is capable of quantifying both good and bad debt.
- The proposed debt model, enhanced by the time series forecasting method, allows us to build a utility model that provides more informed insights to the decision making process of service composition.

II. Background and Related work

Technical debt can be attributed to sub-optimal decisions, shortcut on decisions, and/or deferred activities that can incur extra cost/rework, if it would be carried in the future as when compared the current time. Technical debt metaphor was initially coined by Cunningham in 1992 [2]. Software engineering community presented this metaphor and discussed its applicability to many software artifacts, covering code, requirements, architecture, testing and documentation, among the other. The common understanding is that technical debt is the result of making technical compromises that are expedient in short-term but that create a technical context that increases complexity and cost in the long term [3]. If these technical compromises are not paid back than technical debt can be incurred and degrade the system quality or the development team productivity in long term. By incurring technical debt is not always bad, if organization makes informed decisions or strategic reasons about to incur debt [5]. McConnell [6] classified the term "technical debt" into intentional and unintentional. An intentional debt is the debt which is taken by an organization to optimize the present value in the software project rather than future value or to make informed decisions for gaining short-term benefits. On the other hand, unintentional debt can be incurred unknowingly when an organization makes non-strategic or inappropriate decisions in software project.

In recent years, researchers applied technical debt metaphor in cloud computing based services. Alzaghoul et. al.[7] applied real option approach for managing technical debt at the service selection for the cloud-based service oriented architecture. They identified technical debt of substitution decisions bease on the Binomial Real Option approach. Skourletopoulos et. al. [8] described an approach for evaluating the technical debt for the selection of mobile cloud-based service. The problem is formulated based on the cost-benefits analysis and considered linear growth of users in system modelling. However, the proposed approach did not consider time sensitivity and runtime service execution environment. The effective management of technical debt needs to consider these attributes because runtime environment changes must have the root of causing technical debt and time sensitivity guides to make debt-aware decision about when to pay-off the accumulated debt in service composition under SaaS cloud.

III. Technical Debt in Service Composition

Technical debt in service composition can be observed at different levels (e.g., service utility, recomposition decisions, or SLA violation etc.). Technical debt can be attributed to ill-informed design decisions that can, for instance, relate to sub-optimal capacity planning and be incurred when a composite service is not fully utilized. This, for example, can be due to significant drop in requests workload in SaaS cloud. As a consequences, the operational cost may exceed the service's revenue. Furthermore, a composite service can bear a technical debt by design when environmental changes (e.g., partner service failure or QoS fluctuation etc.) put pressure on the system to recompose the composite service. Additionally, technical debt can be associated with an inappropriate engineering decisions or poorly justified run-time decisions for recomposing the composite services. These decisions can carry short-term benefits in terms of improving service utility, but they might not be geared towards long-term benefits or future value creation.

In summary, we argue that a technical debt-aware recomposition decision is needed for managing the above described issues. We motivate the need for treating these accrued debt as a "time-sensitive moving target" in service composition that needs to dynamically monitored for transforming the accumulated debt into future value.

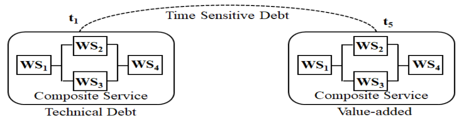

Figure 1: Time series forecasting based debt estimation

Technical debt is not always bad, if it can help the developers to speed development process [9]. We see this argument as a valid point in service composition for improving composition utility or avoiding needless recomposition. In Figure 1, an intentionally technical debt is incurred in service composition, when we decide to postpone the service recomposition at time t_1 and look the possibility of generating future value in the current service composition plan using time-series based predictive value. Technical debt can be manage proactively, if it is visible in service composition. In this case, technical debt is clearly visible as a reengineering cost of service recomposition. An interest may be accumulated over the incurred technical debt in service composition at time t_1 due to poorly selected services in the composition. Interest indicate the composite service utility (e.g., cost of unused service capacity or penalty) which may be underutilized or overutilized due to dynamic changes in requests workload.

A. Technical Debt Indicators in Service Composition

Technical Debt Indicators consist information about what type of technical debt (good or bad debts) is, why and when was incurred, how much debt was estimated, when it will be pay off in future [10]. We identified following key TD indicators in service composition.

SLA Violation: SLA violation constitutes the unintentional technical debt in service composition when a composite service does not satisfy the predefined response time mentioned in end-users SLA then a penalty cost against each request violation would be count as interest over the technical debt.

Runtime decisions: An inappropriate or poorly justified runtime decision for service recomposition may lead the technical debt in a way to select unsuitable component services for composing a new composite service which can not support the scalability requirements in changing requests workload.

Service utility: Service utility constitutes the technical debt when a composite service is sub-optimal from the utility point of view. For example, a sub-optimal composite service can incur an intentional debt by getting service scalability benefits in the future.

Moreover, a decision making needs to know the nature of accumulated debt in terms of good or bad debts. We describe the good and bad debts in composite service perspective and identifying their consequences.

Good Debt: A good technical debt in service composition is viewed as "time-sensitive moving target" that needs to monitor for transforming the accumulated debt into future values creation. For example, Figure 2 shows that a composite service is underutilized in a way to deliver more than the required demand of the users at time t_1 and intentionally accumulates the debt for a time period (e.g., t_1 to t_5). We may accept such debt in a way to consider the future demand for scaling-up the service capacity that transforms the accumulated debt into future values.

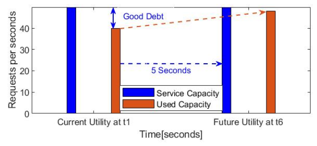

Figure 2: Good Debt

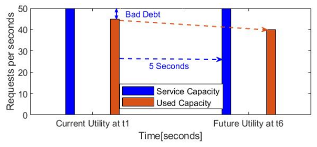

Figure 3: Bad Debt

Bad Debt: A bad debt in service composition may leads the situation of continuous under-utilization of composite service and will not be able to pay off the accumulated debt in future as shown in Figure 3. As consequences, such accumulated debt negatively impacts the service utility that needs to manage by taking proactive decisions.

IV. Measuring Technical Debt in Service Composition

In this section, we describe how ARFIMA, a time series forecasting method, can be used to predict workload of a component service. Drawing on the prediction, we then present a formal technical debt model in the context of service composition under SaaS cloud. Such a model identifies and estimates the possible technical debt with respect to the overall utility, which would provide greater insights to the decision making process of recomposition.

1) Requests workload prediction: Undoubtedly, the dynamic changes of workload on a component service is the fundamental causes of possible technical debt. To predict such a workload, we use Autoregressive Fractionally Integrated Moving Average model (ARFIMA) [4], a widely used time series model that guarantee the prediction accuracy when a time series contains long memory pattern. We prepared the requests workload time series that contains the number of observed requests at each time interval (e.g., seconds) and feed it as an input to ARFIMA for predicting the future requests workload at every second. Formally, the general expression of $ARFIMA(p, d, q)$ can be expressed as:

$$(1 - \sum_{i=1}^{p} \phi_i B^i)(1 - B)^d W_t = (1 + \sum_{i=1}^{q} \theta_i B^i)\varepsilon_t \qquad (1)$$

whereby W_t is the workload for a component service at time t. ε_t is a white noise process. B is the backward shift operator and $(1 - B)^d$ is the fractional differencing operator. The fractional number d is the memory parameter and $d \in [-0.5, 0.5]$. $1 - \sum_{i=1}^{p} \phi_i B^i$ is the autoregressive polynomial of order p and $1 + \sum_{i=1}^{q} \theta_i B^i$ is the moving average polynomial of order q in the lag operator B. We estimate the value of memory parameter d using `fdGPH()` function in the `R` forecast package proposed by [11]. The value of p is the autoregressive order that indicates the number of distinct lags appearing in the forecasting equation, and q is the moving average order that shows the number of lagged forecast error in the prediction equation.

A. Technical Debt Computing Model

To estimate technical debt in service composition, we adopt the notions of principal and interest [3][10] from technical debt metaphor into a contextualized model for the analysis.

1) Recomposition Principal: In the context of service recomposition, we use principal to denote the invested cost of recomposing the entire composite service for improving service utility. The principal can be derived from the resources usages, such as the CPU time or the effort spent by software engineer for the decision making of the service composition. Specifically, we compute the principal for recomposing a service using equation 2.

$$Principal = E \times C_{cpu} \qquad (2)$$

Suppose that the recomposition process requires 2 seconds (denoted as E) and the execution cost of CPU is \$ 0.0025 per second (denoted as C_{cpu}), then it takes a principal as 2 $\times$ 0.0025 = \$ 0.005.

2) Interest: An interest can be accumulated over time on the component service which may be under-utilized or over-utilized. In such context, the interests may be accumulated over time on the yth component service for the xth abstract service (denoted as CS_{xy}). For such a component service, the interests accumulated up to the future n timesteps can be derived from the actual service capacity (i.e., service throughput denoted as T) and the predicted workload at time t (i.e., W_t) from equation 1, as shown below:

$$Int(CS_{xy}) = \begin{cases} \sum_{t=1}^{n}((T - W_t) \times C) & \text{if } W_t \leq T \\ \sum_{t=1}^{n}((\frac{W_t}{T} - R_{SLA}) \times P) & \text{otherwise} \end{cases} \qquad (3)$$

Clearly, the interests are different depending on two different scenarios of utilizing the capacity of a component service:

(a) Service under-utilization: When the component service is under-utilized, i.e., the predicted workload is smaller than or equals to the capacity of component service ($W_t \leq T$), interest can be calculated as the accumulated cost of unused service capacity. For example, on a component service, suppose that the execution cost of processing each request is \$0.00015 (denoted as C), and a component service has the capacity to process 55 requests per second while the predicted workload on this component service is 48 requests per second. Assuming that the accumulated interests till now is \$1.02, then this component service will carry the interest as \$1.02 + (55-48) $\times$ 0.0015= \$1.0305.

(b) Service over-utilization: When the component service is over-utilized, i.e., the predicted workload is greater than the capacity of component service ($W_t > T$), the SLA requirement on latency (denoted as R_{SLA}) would often be violated [12], and thus a penalty rate (denoted as P) would be used to compute the extra cost to be paid. Suppose again, for a component service, that the accumulated interests till now is \$1.02, and that a given SLA contains the requirement of 2 seconds latency and the penalty rate of latency violation is \$ 0.0025 per second. Now, assuming that the average service latency, derived from the predicted workload and its capacity,

is 3.5 seconds, then the interest would be \$1.02 + (3.5-2.0) $\times$ 0.0025 = \$1.0237.

Finally, up to the future n timesteps, the overall technical debt (denoted as $Debt$) of a decision for recomposing the services can be identified and estimated according to the principal and interests, as shown in equation 4:

$$Debt = Principal + \sum_{x=1}^{k} Interest(CS_{xy}) \tag{4}$$

where k is the total number of abstract services and CS_{xy} is the selected component service for the xth abstract service

B. Calculating Debt Aware Utility for Service Composition

The utility of a service composition consists of the revenue and the fundamental operation cost. In particular, the revenue and cost accumulated for future n timesteps can be calculated as the following:

$$Revenue(CS_{xy}) = \sum_{t=1}^{n} W_t \times C_{tenants} \tag{5}$$

$$Cost(CS_{xy}) = \sum_{t=1}^{n} W_t \times C \tag{6}$$

whereby $C_{tenants}$ is the charge to the tenants per request, which directly contribute to the revenue generated by the composite service. C is again the cost per request to the SaaS provider for using a component service and its infrastructure. W_t is the predicted workload at time t. Combining with the debt model, the utility of a service composition (denoted as U) decision for future n timesteps can be calculated as:

$$U = \sum_{x=1}^{k} Revenue(CS_{xy}) - \sum_{x=1}^{k} Cost(CS_{xy}) - Debt \tag{7}$$

Such a utility model is debt-aware and predictive, which helps to consolidate the decision making process of service composition.

Table I: Simulation Parameters

Parameters	Numerical Value
C_{cpu}: Recomposition cost (e.g., engineering efforts plus CPU execution cost)	0.0025 (\$)
C: Per request execution cost	0.00015 (\$)
$C_{tenants}$: Per request tenant's cost	0.00025 (\$)
P: Penalty per request	200% of its cost
R_{SLA}: Response time (SLA)	1 seconds

Table II: Evaluation of Predication Accuracy

	MAE	RMSE	Theil Coefficient
Request Workload	4.0190	5.0817	0.7532

Table III: Number of Good and Bad Debt for All 3600 Seconds.

	Total good debt	Total bad debt
Composite Service	414	385

C. Identifying Good and Bad Debt

According to our definition about the good and bad debt in section IV, the debt depends on the service utility over execution time. Our approach dynamically monitors the service utility for the future timesteps. The incurred debt can be considered as good or bad based on equation 7, suppose the overall utility (U_t) generated at time t is \$1.5 along with debt acceptance and for the next monitoring period, the predictive utility at t+n is \$ 2.0, which implies that the accumulated debt at time t is good and it should be accepted. This is because the accumulated debts in the past would be paid off by $t+n$, leading to an anticipated improvement on the overall utility. Otherwise, the service capacity can become underutilized and accumulates the bad debt that is unlikely to be paid off in future. Specifically, We calculate the good and the bad debt using following equation:

$$Debt = \begin{cases} Debt_{good} & \text{if } U_t \leq U_{t+n} \\ Debt_{bad} & \text{otherwise} \end{cases} \tag{8}$$

The notion of good and bad debt provide a simple, intuitive, yet effective way for the service provider to make more informed decision on the recomposition of service.

V. Experimental Results

A. Experimental Setup

We extended the Service Composition Middleware [13], a tool for modelling and simulating multi-tenant service composition. Such a tool exploits evolutionary algorithm to optimize service composition in the SaaS cloud. On top of that, we implemented our approach to identify and estimate technical debt throughout the life-cycle of a service composition. In particular, our experiments aim to answer the following research questions:

- **RQ1:** Whether the approach is sufficiently accurate in estimating the technical debt (and utility) for service composition in SaaS cloud?
- **RQ2:** Whether our approach can successfully identify good and bad debts?

We conducted all experiments on the same machine with Intel Core i7 2.60 Ghz. Processor, 8GB RAM and windows 10. We use Sales CRM, a real-world application, as our testing environment (shown in Figure 3). The Sales CRM application processes the incoming requests workload (actual) as shown in Figure 6. In our experiments, the workload is collected from the 1998 FIFA World Cup website trace [14] for the length of 7200 seconds. To evaluate the prediction quality, we preprocess the workload by using the first half as the samples for training the forecasting model, while the remaining workload data is used for testing the accuracy. We conduct a monitoring every 5 seconds. The ARFIMA model is implemented using the `arfima` package in `R` [15]. We run a simulation on multi-tenant middleware by implementing our technical debt approach with the simulation parameters shown in Table I.

B. Results Discussion for RQ1

To predict the workload for each component service, we fit the ARFIMA model and evaluate the prediction accuracy using common accuracy metrics[11]. These metrics contain Mean Absolute Error (MAE) measures the prediction accuracy by averaging the absolute value of the difference between actual value and predicted value, Root Mean Square Error (RMSE) is a standard deviation which is measured by the difference between the actual value and predicted value. Moreover, Theil's coefficient indicates the good forecasting if Theil value lies between 0 and 1, otherwise shows the poor prediction. Table II provides a summary of the mean accuracy of predicting

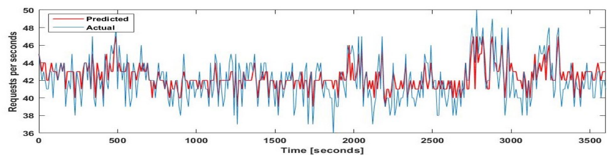

Figure 4: Predicted and actual workload for a component service

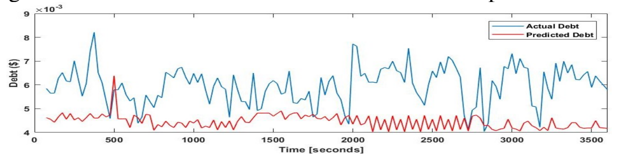

Figure 5: Predicted and actual debt that is accumulated for all component services

workload for all the component services. From the table, we see that the MAE and RMSE is within 15% of the general workload, which has a value between 35 and 50 request per second. This is considered are relatively low error and thus the accuracy is acceptable. As a more detailed example, Figure 4 illustrates the workload trace for a component service. As we can see, the results of the predicted workload generally match with the actual one.

Drawing on the predicted workload, the technical debt can be identified and estimated. Figure 5 shows the predicted and actual debt for all component services. Clearly, the two traces do not match exactly. However, we see that the slope generally follow similar patterns, but differs only in terms of the magnitudes. The deviations between the two traces are also acceptable. In Figure 6, we also plot the predicted and actual utility of the service composition, and again we see generally similar trace. This implies that the predicted workload can also help to estimate the revenue and cost, not only the likely debt.

C. Results Discussion for RQ2

Throughout the entire 3600 seconds run of the service composition and drawing from equation 8, we were able to identify numbers of good and bad debt incurred by the decision of recomposition (monitor every 5 seconds), shown in Table III. Clearly, in our case study, the number of good debt is higher than that of the bad ones, implying a generally healthier status of the service composition.

To provide a more detailed analysis of the good and bad debt, in Figure 7, we illustrate the total utilization of all the component services involved. In particular, we highlight two sets of example: one set is classified as 60 bad debt and another is considered as good (20 debts). At time horizon 1200s to 1500s, the debt accumulated till every 5 seconds length is consider as bad, and hence we have 60 bad debt as there are 60 monitoring. This is because from a point to the next time point (5 seconds later), the utility is decreasing. In contrast, between 2400s and 2500s time points, we found 20 good debt as the accumulated debt at every point would be paid off by the next time point.

VI. Conclusion and Future Work

This paper leverages the notion of technical debt to model the utility of service composition. Specifically, we identified

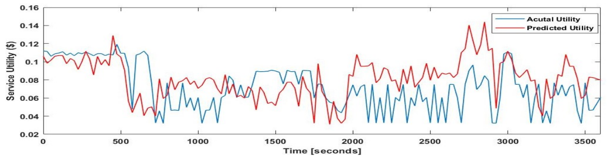

Figure 6: Predicted and actual utility for a service composition

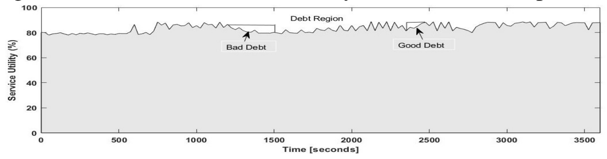

Figure 7: Example of good and bad debt

key technical debt indicators that contribute to the accumulation of technical debt during the execution of a composite service. The model, enhanced by time series forecasting of requests workload, can identify and estimate the future debt and utility for service composition. In future work, we will study the time-sensitivity and its impacts over taking technical debt aware proactive decision for dynamic service recomposition.

References

[1] L. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-Aware Middleware for Web Service Composition,IEEE Transactions on Software Engineering, vol.30,pp. 311-327, 2004.
[2] W. Cunningham. The WyCash portfolio management system. Addendum to the proceedings on Object-oriented programming systems, languages, and applications, pp. 29-20, 1992.
[3] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman. Managing technical debt in software engineering, Dagstuhl Report, 2016.
[4] K. Liu, Y.Q. Chen, and X. Zhang. An Evaluation of ARFIMA (Autoregressive Fractional Integral Moving Average) Programs, Axioms Journal, vol. 6, no. 2, 2017.
[5] R. Marinescu. Assessing technical debt by identifying design flaws in software systems. IBM Journal of Research and Development, vol. 56, no. 5, pp. 9-13, 2012.
[6] S. McConnell. Managing Technical Debt, Construx Software,pp. 1-14, 2008.
[7] E. Alzaghoul and R. Bahsoon. CloudMTD: Using real option to manage technical debt in cloud-based service selection, 4th International Workshop on Managing Technical debt, pp. 55-62, 2013.
[8] G. Skourletopoulos, C.X. Mavromoustakis, G. Mastorakis, E. Pallis, J.M. Batalla, and G. Kormentzas. Quantifying and Evaluating the Technical Debt on Mobile Cloud-based Service Selection Level,IEEE ICC 2016- Communication QoS, Reliability and Modeling Symposium, pp. 1-7, 2016.
[9] Y. Guo and C. Seaman. A portfolio approach to technical debt management. 2nd International Workshop on Managing Technical Debt, pp. 31-34, 2011.
[10] Z. Li, P. Avgeriou, and P. Liang. A systematic mapping study on technical debt and its management, The Journal of Systems and Software, Elsevier, pp. 193-220, 2015.
[11] R.J. Hyndman and A.B. Koehler. Another look at measures of forecast accuracy. International, Journal of Forecasting, Elsevier, pp. 679-688, 2006.
[12] T. Chen, R. Bahsoon, S. Wang, and X. Yao. To Adapt or Not to Adapt?: Technical Debt and Learning Drivent Self-Adaptation for Managing Runtime Performance, ACM/IEEE International Conference on Performance Engineering, pp. 48-55, 2018.
[13] S.Kumar, R. Bahsoon, T. Chen, K. Li, and R. Buyya. Multi-Tenant Cloud Service Composition using Evolutionary Optimization, 24th IEEE International Conference on Parallel and Distributed Systems, 2018.
[14] M. Arlitt and T. Jin. A workload characterization study of the 1998 world cup web site, IEEE network, pp. 30-37, 2000.
[15] J. Veenstra and A.I. Mcleod. Package 'arfima'. version- 1.7-0, https://cran.r-project.org/web/packages/arfima/arfima.pdf, 2018. Accessed: 05-12-2018.