Dynamic Cohesion Measurement for Distributed System

Wuxia Jin, Ting Liu, Yu Qu, Jianlei Chi, Di Cui, Qinghua Zheng

Ministry of Education Key Lab for Intelligent Networks and Network Security, Xi'an Jiaotong University, Xi'an, China

wx_jin@stu.xjtu.edu.cn, {tingliu, yuqu}@mail.xjtu.edu.cn,
{chijianlei7, cuidi}@sei.xjtu.edu.cn, qhzheng@mail.xjtu.edu.cn

ABSTRACT

Instead of developing single-server software system for the powerful computers, the software is turning from large single-server to multi-server system such as distributed system. This change introduces a new challenge for the software quality measurement, since the current software analysis methods for single-server software could not observe and assess the correlation among the components on different nodes. In this paper, a new dynamic cohesion approach is proposed for distributed system. We extend Calling Network model for distributed system by differentiating methods of components deployed on different nodes. Two new cohesion metrics are proposed to describe the correlation at component level, by extending the cohesion metric of single-server software system. The experiments, conducted on a distributed systems-Netflix RSS Reader, present how to trace the various system functions accomplished on three nodes, how to abstract dynamic behaviors using our model among different nodes and how to evaluate the software cohesion on distributed system.

CCS Concepts

• Software and its engineering~Software~creation and management • Software and its engineering~Abstraction, modeling and modularity

Keywords

Extended Calling Network; Cohesion Metric; Distributed System; Static Metric; Dynamic Metric

1. INTRODUCTION

Over the past several decades, software system have enjoyed significant performance benefiting from the fast-developing hardware techniques, such as computing, storage, communication etc. However, it has become increasingly more difficult in recent years to exploit higher CPU speed or larger memory due to various new applications which are impossible to run on a single machine, such as urban computing, gene analysis, social network and big data etc. Instead of developing single-server software for the powerful computers, software is turning from large single-server to multi-server system such as distributed system [1]. One large software system is widely deployed in multiple nodes, and provides service through communicating and interacting with each other across nodes (physical machines, Virtual Machines, Docker container, etc.). This change introduces a new challenge for the software quality measurement, since the current analysis methods for single-server software could not observe and assess the correlation among components on different nodes.

In Software Engineering, one of the main goals is the high quality assurance of software. Software with high quality is easy for comprehension and maintenance. Software quality metrics are popularly used by developers and QAs during software development. For Object-Oriented software system, the widely accepted metrics include cohesion, coupling and complexity. The cohesion of a module indicates the extent to which the components of the module are related. A highly cohesive module performs a set of closely related actions and cannot be split into separate modules [2].

There are a lot of static cohesion metrics. Chidamber and Kemerer [3] proposed Object-Oriented design metric suite, including Lack of Cohesion in Methods metric (LCOM). Briand [4] and Counsell [5] proposed cohesion metrics based on information available in high-level design phase. Dallal [6] proposed a class cohesion metric (LSCC) based on the degree of interaction between each pair of methods from source code. It also verified LSCC usefulness in improving class cohesive. Qu [7] proposed cohesion metric MCC and MCEC using Calling Network model extracted from software source code. MCC and MCEC are measured based on community structure of software system.

Although static software metric is widely used in software engineering, existing studies show the static metric is not enough for software analysis. Yacoub [8] explicitly "distinguished static and dynamic metrics by differentiating between measuring what is actually happening (dynamic) rather than what may happen (static)". Because of the characteristics of inheritance, polymorphism, and runtime binding in Object-Oriented paradigm, static metric is not enough without dynamic one [9, 10]. Dynamic metric is more accurate than static one, and static cohesion metric significantly overestimates the cohesion of software due to the limitation of static analysis methods [11,12]. Specially, Objected-Oriented distributed system has the distinguished character of concurrency and asynchronism [13], and there are plenty of comprehensive and important interacting behaviors across nodes. Therefore, it is not reasonable and not accurate to measure the cohesion of distributed system in a static way.

Recently, there are emerging studies focusing on dynamic cohesion metrics [12,16-18]. Zheng [14] and Tian [15] analyze the dynamic behaviors based on the interaction among methods. Mitchell [16] proposed two run-time Object-Oriented cohesion metrics RLCOM and RWLCOM. Both are direct extensions of LCOM. RLCOM measures the count of instance variables which are actually accessed at run-time, while RWLCOM measures cohesion by weighting each instance variable by the number of times it is

SCTDCP'16, September 3, 2016, Singapore, Singapore
© 2016 ACM. 978-1-4503-4510-1/16/09...$15.00
http://dx.doi.org/10.1145/2975954.2975956

accessed at run-time. Gupta [12] proposed dynamic cohesion metrics for Object-Oriented software at object level instead of class level. Compared with RLCOM [16] and LCOM, the experiments suggested that RLCOM is quite similar to LCOM and the proposed ones can better capture the dynamic information. Mathur [17] defined runtime cohesion metric RuCIVA, similar to RLCOM [16]. Desouky [18] proposed a runtime cohesion metric RLCOM-DESOUKY, an extension to Mathur's RuCIVA metric. It presented an empirical comparison with the existing runtime cohesion metrics suggested by Mitchell et.al. and Mathur et.al. in one case.

However, on one hand, all existing dynamic metrics are just only validated on single-server software system, not including distributed system. On the other hand, most analysis approaches for dynamic cohesion metrics are debugging, dynamic slicing, and modifying source code. These dynamic analysis methods are high-cost, not suitable for large-scale software and not capturing the dynamic behaviors of distributed system. In recent years, many good monitoring tools for distributed system are emerging, such as Dapper, Zipkin, and so on [19-21]. By software instrumentation, the runtime trace information can be acquired in monitoring log. Through analyzing monitoring log, dynamic behaviors can be obtained, and they can help in bug analysis and performance assessment. This mechanism brings low overhead. It is appropriate and efficient for analyzing large-scale complex software system.

In general, existing static cohesion metrics are not accurate and lose the significant dynamic behaviors for distributed system. In addition, compared with other dynamic analysis methods, monitoring is a high-efficient mechanism to capture the runtime information of large-scale distributed software system. Therefore, in this paper focusing on distributed system, we propose a dynamic cohesion metric at component level for distributed system. Firstly, we extend Calling Network model for distributed system by differentiating the methods executed on different nodes. Secondly, two extended Calling Network generation schemes are proposed: Growing Calling Network and Partitioned Calling Network. Thirdly, we present two new cohesion metrics for distributed software system. Finally, the experiments, conducted on a distributed systems-Netflix RSS Reader, present how to trace the various system functions accomplished on three nodes, how to abstract dynamic behaviors of software using our model and how to evaluate the software cohesion on distributed system.

The rest of this paper is organized as follows. Section 2 introduces the extended Calling Network model and cohesion metrics. Section 3 demonstrates the experiments and analysis results, and make conclusion in Section 4.

2. MODEL AND METHODS

2.1 Extended Calling Network Model

Qu [22] proposed Calling Network for software system, we extend the model considering distributed system. The extended Calling Network model for distributed system is given as follows:

Definition 1. **Calling Behaviors** cb. cb is a trace record of one method call.

$cb_k = (t_k, Caller_k, Callee_k, Param_k, VM_{Caller_k}, VM_{Callee_k})$. Where t_k is the timestamp of the method call. $Caller_k$ and $Callee_k$ are self-descriptive. $Param_k$ is the parameter list of $Callee_k$. VM_{Caller_k} is the name of node (physical machine, virtual machine, container, etc.) which the method is running on. So as VM_{Callee_k}.

Definition 2. Calling Behavior Set CB. $CB = \{cb_k \mid k \in \mathbb{N}\}$. CB is an ordered set. According to the execution timestamp of all method calls, k is the sequence number of cb.

Definition 3. Calling Graph. $CG = (V, E)$. CG is a directed graph, in which V stands for the method set and E stands for the method call relations. Let L^V be the set of vertex labels, and L^Ebe the set of edge labels. $L^V \neq \emptyset, L^E \neq \emptyset$. Let A^D be the set of discrete attribute values and $A^N \subset \mathbb{R}$ be the set of numeric attribute values, such that $A = A^N \cup A^D$. The label-to-value mapping function for vertex is denoted as $f_v: V \times L^V \rightarrow$ A. The label-to-value mapping function for edge is denoted as $f_e: E \times L^E \rightarrow$ A. The vertex label and edge label can have various meanings in different scenario. In this paper, vertex label is the name of node which the method is running on, edge label is weight, the method call frequency. That is, $f_v: V \times L^V \rightarrow A^D$, $f_e: E \times L^E \rightarrow A^N$.

Definition 4. **Calling Graph generation function**: $f_{CG\_Gen}: CB \rightarrow CG$.

Definition 5. **Calling Network (CN)**. *CN* is an ordered set of *CG*: $CN = \{CG_i | i \in N\}$, Where $CG_i = f_{CG\_Gen}(CB_i)$, $CB_i \subseteq CB$. *CB* can be partitioned into CB_i using some strategies. In this paper, we discuss two strategies. Other strategies also fit extended *CN* model.

Strategy 1. Use fixed interval and quantity of cbs to generate CG. Two parameters are needed in this strategy: N_{Itv} and N_{CG}. N_{Itv} represents interval between two consecutive CBs, and N_{CG} represents the number of *cbs* in each CB_i. Then,

$CB_i = \{cb_k | (\mathrm{i}-1) \cdot N_{Itv} < \mathrm{k} \leq (\mathrm{i}-1) \cdot N_{Itv} + N_{CG}\}$.

Strategy 2. Use time interval to partition *CB*. Two parameters are set: T_i and Δt. T_i is the i-th time point, Δt is the time window for selecting *cbs*. Then $CB_i = \{cb_k | T_i - \Delta t < t_k \leq T_i\}$.

In general, the Calling Network model is formalized as (Strategy 1 is included):

$$\begin{cases} CN = \{CG_i \mid i \in N\} \\ CG_i = f_{CG_{Gen}}(CB_i), \quad CB_i \subseteq CB \\ CB_i = \{cb_k \mid (\mathrm{i}-1) \cdot N_{Itv} < k \leq (\mathrm{i}-1) \cdot N_{Itv} + N_{CG}\} \\ CG = (V, E), f_v: V \times L^V \rightarrow A^D, f_e: E \times L^E \rightarrow A^N \\ CB = \{cb_k \mid k \in \mathrm{N}\} \\ cb_k = (t_k, Caller_k, Callee_k, Param_k, L^{V_{Caller_k}}, L^{V_{Callee_k}}) \end{cases}$$

2.2 Extended CN Generation Schemes

In this paper, two *CN* generation schemes are proposed by setting different parameters.

Scheme1. Growing Calling Network (Growing CN)

Here, use **Strategy 1** to partition *CB*. Assuming $N_{Itv} = 0$ and $N_{CG} = i \cdot N_{Const}$, where N_{Const} is a constant value. Then $CB_i = \{cb_k | 0 < k \leq \mathrm{i} \cdot N_{Const}\}$. Obviously, this sequence of *CGs* represents the growing process of Calling Graph over time.

Scheme2. Partitioned Calling Network (Partitioned CN)

Here, use **Strategy 2** to partition *CB*. T_i is the end time of one business functionality, and Δt is the processing time window. For i-th business functionality, $T_i - \Delta t$ is the starting time, and T_i is the end time of processing. $CB_i = \{cb_k | T_i - \Delta t < t_k \leq T_i\}$.

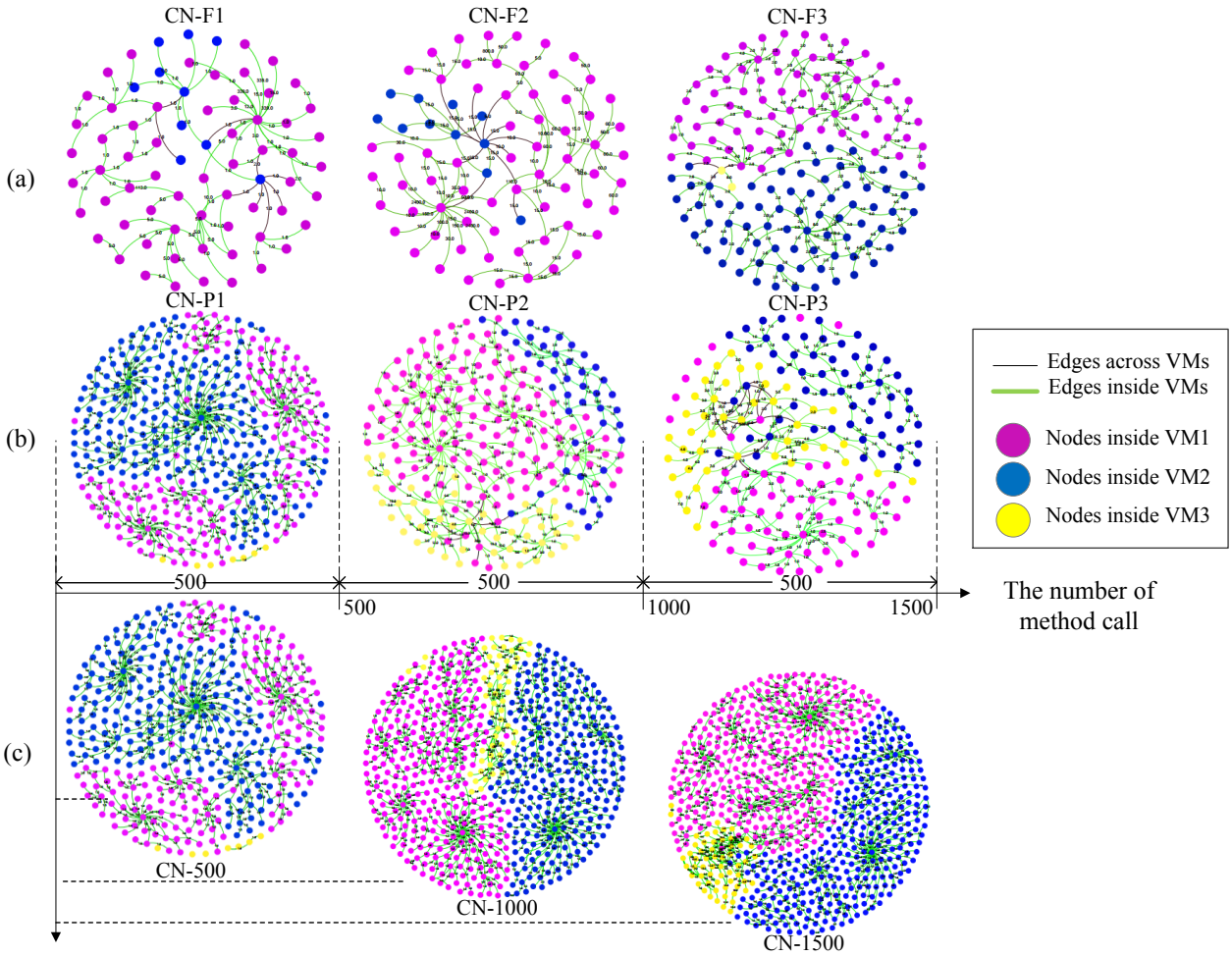


Figure 1. An illustration of Partitioned CNs and Growing CNs of *com.netflix.recipes.rss*

Figure1 shows CNs generated from two schemes above. Package *com.netflix.recipes.rss* [1] is instrumented and the whole software system is deployed on three Virtual Machines (VMs). After the system is launched, it performs business functions according to users' requests over time. The system' dynamic behaviors are modeled in two perspectives: Partitioned CN and Growing CN. Partitioned CNs are illustrated in Figure 1(a). F1, F2 and F3 stand for three kinds of business functionalities. The corresponding CN-F1, CN-F2, and CN-F3 are partitioned CNs sliced by the performed business functionality. Figure 1(c) shows the Growing CNs including CN-500, CN-1000 and CN-1500, where $N_{Const} = 500$. Each Growing CN is generated from the system launching time to the end time when a certain number of method call is reached. Figure 1(b) presents the incremental part from the previous Growing CN to the current Growing CN. For example, CN-P2 is the incremental part from CN-500 to CN-1000. Using Partitioned CN and Growing CN in Figure 1, we can observe the change process of dynamic behaviors of the software system.

2.3 Dynamic Cohesion Metric

Employing the similar idea with Lack of Cohesion in Methods metric (LCOM)[3], two cohesion metric at component level ***CC*** and ***CCW*** are proposed. Both are measured based on the extended Calling Network model. The cohesion idea is: compared with vertices (methods) without edge between them, if there is an edge between two vertices (methods), there are strong correlation between them. Also, take into weight of edge, the more edge weight, the stronger correlation between the two vertices.

$CG = (V, E), f_v: V \times L^V \rightarrow A^D, f_e: E \times L^E \rightarrow A^N.$

$P_i = \{e_j \mid e_j = (v_a, v_b), f_v(L^{v_a}) = f_v(L^{v_b}) = C_i\}$

$Q_i = \{e_j \mid e_j = (v_a, v_b),\ (f_v(L^{v_a}) = C_i \text{ and } f_v(L^{v_b}) \neq C_i) \text{ or } (f_v(L^{v_a}) \neq C_i \text{ and } f_v(L^{v_b}) = C_i)\}$

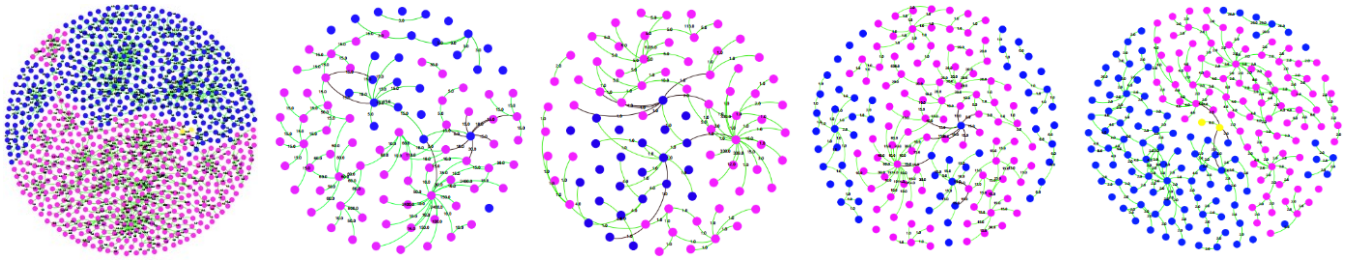
P_i is set of edge which is within Components C_i. Q_i is set of edge which is across Component C_i.

For Component C_i, the Lack cohesion of method $LCOM\_com_i$ is denoted as:

$$LCOM\_com_i = \frac{|Q_i|}{|P_i| + |Q_i|}$$

The cohesion metric $\boldsymbol{CC}_i$ is denoted as:

[1] https://github.com/Netflix/recipes-rss/tree/master/rss-middletier/src/main/java/com/netflix/recipes/rss



(a)CN-Start (b)CN-Add (c)CN-Get (d)CN-Delete (e)CN-Idle

Figure 3. An illustration of five Partitioned CNs

$$\boldsymbol{CC}_i = 1 - LCOM\_com_i = \frac{|P_i|}{|P_i| + |Q_i|}$$

CCW is similar with ***CC***, only considering weight of edge. Let

$$P\_W_i = \{f_e(e_j) \mid e_j = (v_a, v_b), f_v(L^{v_a}) = f_v(L^{v_b}) = C_i\}$$

$$Q\_W_i = \{f_e(e_j) \mid e_j = (v_a, v_b),\ (f_v(L^{v_a}) = C_i \text{ and } f_v(L^{v_b}) \neq C_i) \text{ or } (f_v(L^{v_a}) \neq C_i \text{ and } f_v(L^{v_b}) = C_i)\}$$

P_W_i is weight set of edge which is within Components C_i. Q_i is weight set of edge which is across Component C_i.

For Component C_i, the Lack cohesion of method $LCOM\_W\_com_i$ is denoted as:

$$LCOM\_W\_com_i = \frac{\sum_{Q\_W_i} x_j}{\sum_{P\_W_i} x_j + \sum_{Q\_W_i} x_j}$$

The cohesion metric $\boldsymbol{CCW}_i$ for a Component C_i is denoted as:

$$\boldsymbol{CCW}_i = 1 - LCOM\_W\_com_i = \frac{\sum_{P\_W_i} x_j}{\sum_{P\_W_i} w_i + \sum_{Q\_W_i} x_j}$$

3. EVALUATION

3.1 Setup

We conduct a study on RSS Reader application. RSS Reader[2] is a distributed enterprise application developed by Netflix. This application provides web service for users to get, add, and delete RSS feeds. It contains three main service components, including Middletier, Edge, and Eureka. These components are deployed in three different nodes (here is VMs) hosted on Ubuntu15.10 server. The whole service framework is showed in Figure 2.

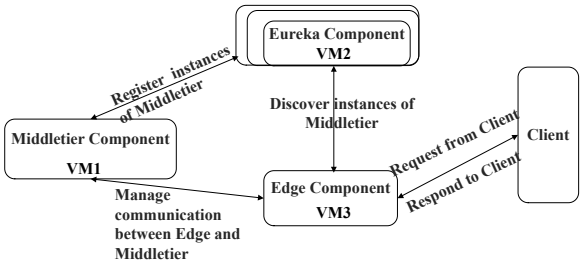


Figure 2. The service framework of RSS Reader application

In order to collect the dynamic traces, we use Kieker 1.12 [23], which monitors software system's runtime behaviors and stores trace log into memory, file system or database. For distributed system, a big feature is Inter-Process Communication (IPC). Different type of system adopt different Communication model. Jersey [3] is one of the implementation of REST-based communication environment and is used by RSS Reader application. We collect distributed traces using probes provided by Kieker 1.12 for REST-based environments with Jersey. In our experiment, the generated CN is extracted from the trace log using Python script, and CN is analyzed using Networkx 1.11[4] Library.

3.2 Partitioned CN

In this section, use **Scheme 2** to generate Partitioned CN. The RSS Reader application provides users with three business functionalities, including getting, adding, deleting Feeds. We Use JMeter [5] to separately send Get, Add, and Delete requests concurrently in five times. So CN-Delete, CN-Add, CN-Get can be generated. Besides, CN-Start is generated during the period before the whole service launched successfully, and CN-Idle is generated lasting one minute during which the application does not have requests to process. Figure 3 illustrates the five Partitioned CNs. Methods located on Edge node, Middletier node and Eureka node are separately rendered in blue, red and yellow color. Each edge is labeled with weight, call frequency. Edge within node is rendered in green while edge across nodes is rendered in black. Figure3 shows that the Eureka component does not participate in business functionality in most cases, so we only focus on Edge and Middletier components in Partitioned CNs. ***CC*** and ***CCW*** metrics are measured and results are listed in Table 1.

Table 1. Measurement result of Partitioned CNs

Partitioned CN	*CC*		*CCW*	
	Middletier	**Edge**	**Middletier**	**Edge**
CN-Start	0.9982	0.9957	0.9939	0.9939
CN-Delete	0.8220	0.6039	0.9811	0.5773
CN-Add	0.8684	0.6154	0.9857	0.5134
CN-Get	0.8732	0.625	0.9924	0.5674
CN-Idle	0.9898	0.9898	0.9848	0.9897

Table 1 shows that each component has high cohesion in different Partitioned CNs, so the component is well designed and implemented in this open source distributed system. Specially, CN-Delete, CN-Add and CN-Get show similar cohesion. The reason is that these three CNs are generated from runtime traces presenting similar business logic of the software system. This phenomena motivates us that we can try to do service partitioning and

[2] https://github.com/Netflix/recipes-rss

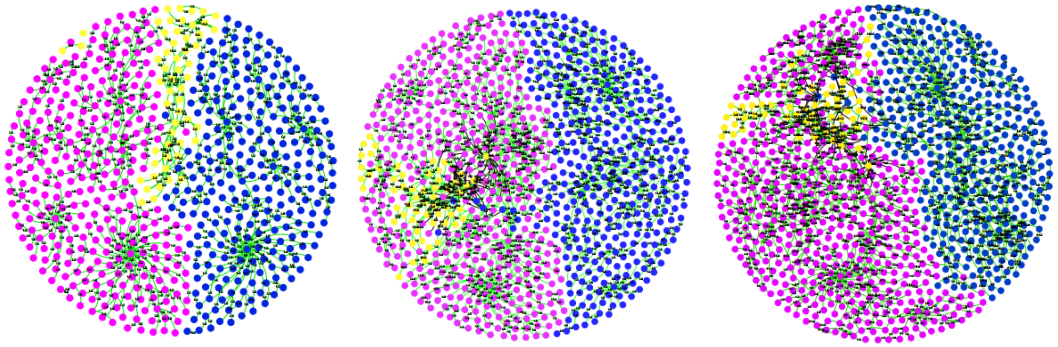
[3] https://jersey.java.net

[4] http://networkx.github.io

[5] http://jmeter.apache.org

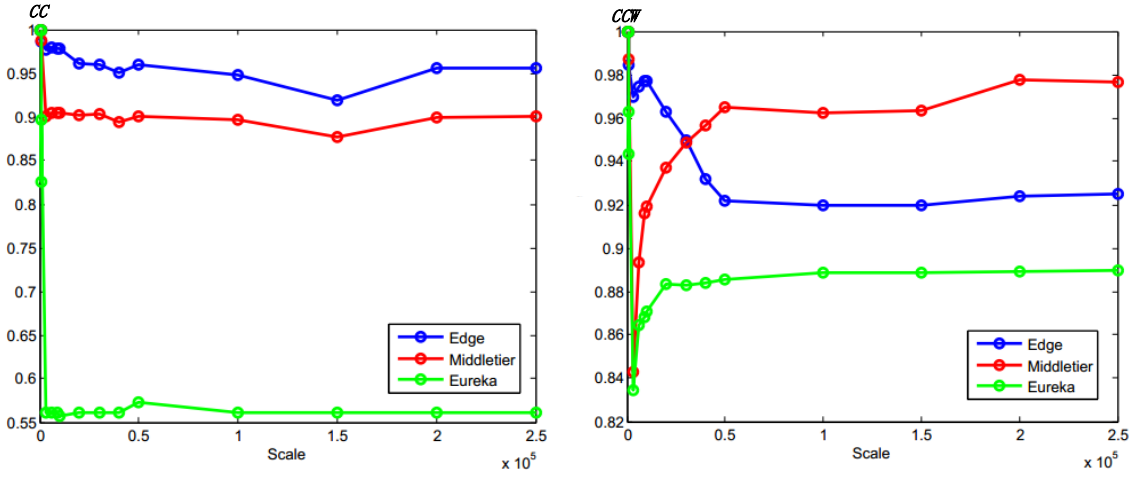
Table 2. Measurement result of Growing CNs

Growing CN	*CC*			*CCW*		
	Edge	Middletier	Eureka	Edge	Middletier	Eureka
CN-100	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
CN-300	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
CN-600	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
CN-900	0.9860	0.9877	0.8261	0.9846	0.9873	0.9432
CN-1000	0.9861	0.9877	0.8961	0.9846	0.9873	0.9627
CN-3000	0.9774	0.9002	0.5608	0.9700	0.8428	0.8347
CN-6000	0.9792	0.9050	0.5608	0.9747	0.8935	0.8645
CN-9000	0.9787	0.9050	0.5608	0.9769	0.9160	0.8681
CN-10000	0.9787	0.9051	0.5570	0.9769	0.914	0.8709
CN-20000	0.9615	0.9017	0.5608	0.9627	0.9372	0.8836
CN-30000	0.9596	0.9031	0.5608	0.9498	0.9489	0.8831
CN-40000	0.9501	0.8947	0.5608	0.9319	0.9567	0.8841
CN-50000	0.9596	0.9005	0.5724	0.9217	0.9650	0.8856
CN-100000	0.9482	0.8973	0.5608	0.9200	0.9626	0.8888
CN-150000	0.9189	0.8766	0.5608	0.9200	0.9632	0.8888
CN-200000	0.9558	0.8999	0.5608	0.9240	0.9778	0.8892
CN-250000	0.9558	0.9009	0.5608	0.9250	0.9764	0.8900



(a)CN-1000 **(b)CN-10000** **(c)CN-100000**

Figure 4. An illustration of three Growing CNs



(a)*CC* value of Growing CNs **(b)*CCW* value of Growing CNs**

Figure 5. The cohesion changing chart of Growing CNs

deployment optimization based on our model and methods in the future.

3.3 Growing CN

In this section, use **Scheme 1** to generate Growing CN. After RSS Reader application start successfully, it always remains in running status. In our experiment, the trace log is generated when the system is in operation for five hours. During system's operation, it can either be in busy or idle status. Because of too many traces, we divide the scale of growing CN into four level forming four groups in terms of the number of method call. The first group includes from CN-100 to CN-900. The second group includes from CN-1000 to CN-9000. The third one includes from CN-10000 to CN-50000, and the others forms the fourth group. Figure 4 illustrate three Growing CNs, and the color interpretation is same as Figure 3. ***CC*** and ***CCW*** metrics are also measured and results are listed in Table 2. The changing charts of cohesion metric of the Growing CNs are showed in Figure 5.

Table 2 shows that each component has high cohesion in different Growing CNs. It is discovered that the measured cohesion metric of Eureka component is relatively lower than ones of Middletier and Edge component. This feature is related with the role or function of Eureka Component. Eureka is responsible for locating service component and load balancing [6], so it performs more interaction behaviors across components than other components. Also, Figure 5 shows that cohesion of each component tends to be a stable state with the scale increase of growing CNs over time. It motivates us whether the cohesion metric fluctuates dramatically or not if some faults happen in the software system, and we will conduct such study in the future.

4. CONCLUSION AND DISCUSSION

In this paper, we have proposed an extended Calling Network model to abstract dynamic behaviors of distributed system. Based on this model, two dynamic cohesion metric ***CC*** and ***CCW***, extensions from LCOM, are proposed at component level. We conducted one study on a distributed system-RSS Reader. It has been showed that our model and cohesion metrics, extracted from trace log by monitoring, can well present the dynamic behavioral characteristic for a distributed system. It has been discovered that component is highly cohesive in a distributed system with high quality, and the dynamic cohesion metric of Growing CNs tends to be stable with the growing of CNs over time. In addition, it has been discovered that Partitioned CNs with similar kinds of business functionality show similar cohesion feature.

Of course, there are some points to be improved in this paper. In the future, on one hand more experiments will be conducted on diverse distributed systems to validate our model and methods. On the other hand, we will do comparison with the existing study with respect to dynamic cohesion metric.

5. ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (91218301, U1301254, 91418205, 61472318, 61428206, 61532015), Fok Ying-Tong Education Foundation (151067), Key Project of the National Research Program of China (2013BAK09B01), Ministry of Education Innovation Research Team (IRT13035) and the Fundamental Research Funds for the Central Universities.

6. REFERENCES

[1] Y. Han. 2010. On the clouds: a new way of computing. *Information Technology and Libraries*, 29, 2 (June. 2010), 87–92. DOI= http://dx.doi.org/10.6017/ital.v29i2.3147.

[2] J. Bieman and L. Ott. 1994. Measuring functional cohesion. *IEEE Trans. Software Eng.* 20, 8 (August. 1994), 644-657. DOI= http://dx.doi.org/10.1109/32.310673.

[3] Chidamber SR, Kemerer CF. 1994. A metrics suite for object oriented design. *IEEE Trans. Software Eng.* 20, 6 (June. 1994), 476-493. DOI= http://dx.doi.org/10.1109/32.295895.

[4] L. C. Briand , S. Morasca , and V. R. Basili. 1999. Defining and validating measures for object-based high-level design. *IEEE Trans. Software Eng.* 25, 5 (September. 1999), 722-743. DOI= http://dx.doi.org/10.1109/32.815329.

[5] S. Counsell, S. Swift, and J. Crampton. 2006. The interpretation and utility of three cohesion metrics for object-oriented design. *ACM Trans. Softw. Eng. Methodol. (TOSEM)*, 15, 2 (April. 2006), 123-149. DOI= http://doi.acm.org/10.1145/1131421.1131422.

[6] Al Dallal J, Briand LC. 2012. A precise method-method interaction-based cohesion metric for object-oriented classes. *ACM Tran. Softw. Eng. Methodol. (TOSEM)*, 21, 2 (March. 2012), 8-8. DOI=http://dx.doi.org/10.1145/2089116.2089118.

[7] Yu Qu, Xiaohong Guan, Qinghua Zheng, Ting Liu, Lidan Wang, Yuqiao Hou, Zijiang Yang. 2015. Exploring community structure of software Call Graph and its applications in class cohesion measurement. *J. Syst. Softw.* 108 (October. 2015), 193-210. DOI=http://dx.doi.org/10.1016/j.jss.2015.06.015.

[8] S. M. Yacoub, H. H. Ammar, T. Robinson. 1999. Dynamic metrics for object oriented designs. *In Proceedings of the Sixth International Software Metrics Symposium* (Florida, USA, November 04 - 06, 1999). Metrics'99. IEEE Computer Society, California, CA, 50-61. DOI=http://dx.doi.org/10.1109/METRIC.1999.809725.

[9] Amjed Tahir and Stephen G. MacDonell. 2012. A systematic mapping study on dynamic metrics and software quality. *In Proceedings of the 28th IEEE International Conference on Software Maintenance* (Trento, Italy, September 23 – 28, 2012). ICSM'12. IEEE Computer Society, Washington DC, 326-335. DOI= http://dx.doi.org/10.1109/ICSM.2012.6405289.

[10] E. Arisholm, L. C. Briand , A. Foyen. 2004. Dynamic coupling measurement for object-oriented software. *IEEE Trans. Software Eng.* 30, 8 (August. 2004), 491-506. DOI= http://dx.doi.org/10.1109/TSE.2004.41.

[11] N. Gupta, Tucson, AZ, P. Rao. 2001. Program execution based module cohesion measurement. *In Proceedings of the 16th Annual International Conference on Automated Software Engineering* (California, USA, November 26 - 29, 2001). ASE'01. IEEE Computer Society, US, 144 – 153. DOI= http://dx.doi.org/10.1109/ASE.2001.989800.

[6] https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance

[12] Varun Gupta, Jitender Kumar Chhabra. 2011. Dynamic cohesion measures for object-oriented software. *J. Syst. Archit*. 57, 4 (April. 2011), 452–462. DOI=http://dx.doi.org/10.1016/j.sysarc.2010.05.008.

[13] Andrew S. Tanenbaum, Maarten Van Steen. 2006. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Upper Saddle River, NJ, USA.

[14] Zheng Q H, Ou Z J, Liu T, et al. 2012. Software structure evaluation based on the interaction and encapsulation of methods. *Sci. China Inf. Sci.* 55, 12 (December. 2012), 2816-2825. DOI= http://dx.doi.org/10.1007/s11432-012-4542-4.

[15] Tian, Z., Zheng, Q., Liu, T., Fan, M., Zhuang, E. and Yang, Z. 2015. Software plagiarism detection with birthmarks based on dynamic key instruction sequences. *IEEE Trans. Software Eng.* 41, 12 (December. 2015), 1217-1235. DOI= http://dx.doi.org/10.1109/TSE.2015.2454508.

[16] Aine Mitchell, James F. Power. 2004. Run-time cohesion metrics: an empirical investigation. *In Proceedings of International Conference on Software Engineering Research and Practice* (Nevada, USA, June 21 – 24, 2004). SERP'04. CSREA Press, 532-537. DOI= http://dx.doi.org/10.1.1.100.7997.

[17] Mathur, R., Keen, K. J., and Etzkorn, L. H. 2011. Towards a measure of object oriented runtime cohesion based on number of instance variable accesses. *In Proceedings of the 49th Annual Southeast Regional Conference* (Kennesaw, USA, March 24 – 26, 2011). ACM-SE '11. ACM, New York, NY, 255-257. DOI= http://dx.doi.org/10.1145/2016039.2016105.

[18] Amr F. Desouky, Letha H. Etzkorn. Object oriented cohesion metrics: a qualitative empirical analysis of runtime behavior. *In Proceedings of the 49th Annual Southeast Regional Conference* (Kennesaw, GA, USA, March 28 – 29, 2014). ACM SE '14. ACM, New York, NY, 58-63. DOI= http://dx.doi.org/10.1145/2638404.2638464.

[19] Benjamin H. Sigelman, Luiz Andre Barroso, Mike Burrows, etc.. 2010. *Dapper, A Large-Scale Distributed Systems Tracing Infrastructure*. Technical Report. Google.

[20] Aniszczyk, C. 2012. *Distributed Systems Tracing with Zipkin*. Technical Report. Twitter Blog.

[21] Caitle Mccaffrey. 2015. The verification of a distributed system. *ACM Queue*, 13, 9 (November-December. 2015), 150-161. DOI= http://dx.doi.org/10.1145/2857274.2889274.

[22] Yu Qu, Xiaohong Guan, Qinghua Zheng, Ting Liu, Jianliang Zhou, Jian Li. 2015. Calling network: a new method for modeling software runtime behaviors. *ACM SIGSOFT Softw. Eng. Notes*, 40, 1 (January. 2015), 1-8. DOI= http://dx.doi.org/10.1145/2693208.2693223.

[23] A. van Hoorn, J. Waller, and W. Hasselbring. 2012. Kieker: a framework for application performance monitoring and dynamic software analysis. *In Proceedings of the Third Joint WOSP/SIPEW International Conference on Performance Engineering*(Boston, USA, April 22 – 25, 2012). ICPE'12. ACM, New York, NY, 247–248. DOI= http://doi.acm.org/10.1145/2188286.2188326.