



A RESTful Web Service for Non-overlapping Community Quality Assessment with MPI

Yuhong Feng¹, Song She¹, Yuanshi Wu¹, Mingyang Zhou¹, Zhong Ming¹(✉),
and Haoming Zhong²(✉)

¹ National Engineering Laboratory for Big Data System Computing Technology,
College of Computer Science and Software Engineering, Shenzhen University,
Shenzhen 518060, China

mingz@szu.edu.cn

² Data Science and Application, Intelligence Department, WeBank, Shenzhen, China
hmzhong@webank.com

Abstract. Network community quality assessment (CQA) is essential for many applications. However, large scale communities detected from nowadays rapidly growing social networks present great challenge to its computation efficiency. Though parallel algorithms using message passing interface (MPI) have recently been introduced into the field, its computation efficiency needs further improvement. Meanwhile, the complexity of the MPI implementation handicaps data scientists from adopting it. In this paper, we first design a fast MPI-based metrics computation algorithm. Then we propose a RESTful framework to wrap the non-overlapping CQA metrics computation as a Web service, i.e., non-overlapping CommuMetrics, which makes it easy to use. Finally, experiments in empirical networks demonstrate the effectiveness of our method in terms of *execution time* and *speedup*.

Keywords: Web services · RESTful
Community quality assessment · MPI

1 Introduction

Networks or graphs usually display *community structure*, i.e., their vertices are aggregated into *communities* (or *groups*, *clusters*, *modules* in literature) [1], where the vertices in the same community have higher similarity scores or denser connections than those between them. Detecting the community structure for a network, i.e., community detection, has been regarded as one of the key issues in the study of networked systems and has been widely applied in various fields. Many algorithms have been proposed to detect communities, e.g., modularity-based [2], network partition based [3], clustering (or spectral) based [4], statistical inference-based [5], dynamic process based [6] algorithms. This gives rise to the requirement of community quality assessment (CQA for short).

Girvan and Newman (GN) [7] and Lancichinetti–Fortunato–Radicchi (LFR) proposed algorithms for generating networks with benchmark community structure according to specified parameters, e.g., edge density, strength of community structure. Detection algorithms can be applied on the generated networks to detect communities, which will be compared with the benchmark community structure for CQA. Various CQA metrics have been proposed to evaluate the quality of communities. In general, communities can be categorized into *non-overlapping* or *overlapping*, where a vertex in non-overlapping communities belongs to only one community and a vertex in overlapping ones can belong to more than one community. The metrics for non-overlapping CQA and overlapping CQA are different, in this paper, we focus on the non-overlapping CQA metrics computation. In the rest of the paper, “CQA metrics” means non-overlapping CQA metrics and “communities” means non-overlapping communities.

Sequential algorithms for computing CQA metrics, e.g., *scikit-learn* [8] and *igraph* [9], have high computation complexity. Parallel algorithms using message passing interface (MPI) [10] and Pthreads have recently been proposed to expedite the computation over large scale communities [11], where MPI is an efficient distributed memory programming model using message passing communication mechanisms. However, the rapid growth of application data like Facebook which have more than 2.2 billion monthly active users [12] presents great challenges to the CQA execution efficiency. Meanwhile, the complexity of MPI implementation details, e.g. efficient resource utilization, load balancing, execution management, handicap data scientists from adopting it for data analysis because of the high learning curve.

Web services¹ are a set of standards and open protocols for encapsulating application functions to be programmatically invoked over the Internet. Representational State Transfer (REST) [13] is a web standard based architectural style where HTTP protocol and uniform interfaces like *GET*, *POST* are used for data communication over the Internet. Data analytics provisioned via RESTful Web services [14–16] can remove the implementation and deployment details, and ease the data scientists’ developing job. Therefore, RESTful Web services for CQA metrics provisioning are desired.

This paper designs, implements and evaluates *Non-overlapping CommuMetrics* (CommuMetrics for short), an effective RESTful Web service for non-overlapping CQA metrics provisioning. We first propose an improved MPI based CQA metrics computing algorithm to expedite its execution, then we design a RESTful framework to wrap the improved MPI-based CQA computation as a Web service to make it easy to use. Finally, an empirical comparison study has been carried out to demonstrate that the services are provisioned in an efficient way.

The rest of the paper is organized as follows: Sect. 2 describes various metrics and their computation for non-overlapping CQA. Section 4 presents our improved MPI based metrics computation. Section 3 introduces the architecture

¹ https://www.w3schools.com/xml/xml_services.asp.

design and prototype implementation of the RESTful CQA metrics based web service provision. Section 5 describes our empirical experimental performance evaluation on the service provision. Finally, Sect. 6 concludes the paper.

2 Metrics Computation for Non-overlapping Community Quality Assessment (CQA)

A network \mathcal{N} consists of n vertices that are represented as a set $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$. Let $\mathcal{X} = \{X_1, X_2, \dots, X_K\}$ represent the communities detected over \mathcal{N} , where K denotes the number of communities. Then for any $1 \leq r \neq r' \leq K$, $X_r, X_{r'} \subseteq \mathcal{V}$, we have $X_r \cap X_{r'} = \emptyset$ and $\sum_{r=1}^K |X_r| = n$. Let $\mathcal{Y} = \{Y_1, Y_2, \dots, Y_{K'}\}$ represent the benchmark communities for \mathcal{N} , where K' can have different value from K .

Values for CQA metrics are between 0 and 1, measuring two communities' similarity. The higher the value is, the closer two communities are. Particularly, the value is 0 when the detected communities are completely different from the benchmark, and the value is 1 when the detected communities are exactly the same to that. According to how metrics are computed, CQA metrics can be categorized into *pair counting based* (PC-based for short), *cluster matching based* (CM-based for short) and *mutual information based* (MI-based for short), which have been described in detailed in papers [11, 17]. To make the paper self-contained, we first give a summarized introduction to the metrics. The PC-based metrics, e.g., *Rand Index* (RI) [18], *Adjusted Rand Index* (ARI) [19], *Jaccard Index* (JI) [20] and Fowlkes–Mallows Index (FMI) [21], represent the fraction of pairwise vertices which are aggregated to the same community. For \mathcal{X} and \mathcal{Y} , let N_{11} and N_{00} represent the number of pairwise vertices categorized into the same community and different communities respectively, N_{10} (N_{01}) denote the number of pairwise vertices that are categorized to the same community in \mathcal{X} (\mathcal{Y}) and different communities in \mathcal{Y} (\mathcal{X}). Then RI, ARI, JI and FMI can be calculated using Eqs. 1–4 respectively.

$$RI(\mathcal{X}, \mathcal{Y}) = \frac{N_{11} + N_{00}}{\binom{n}{2}} \quad (1)$$

$$ARI(\mathcal{X}, \mathcal{Y}) = \frac{N_{11} - M}{\frac{1}{2} [(N_{11} + N_{10}) + (N_{11} + N_{01})] - M} \quad (2)$$

where the generalized hypergeometric distribution is assumed to be a randomness model, i.e., K for \mathcal{X} , K' for \mathcal{Y} and the size of each community are fixed, and $M = \frac{1}{2}(N_{11} + N_{10})(N_{11} + N_{01})$ is the expected value under null hypothesis.

$$JI(\mathcal{X}, \mathcal{Y}) = \frac{N_{11}}{N_{11} + N_{10} + N_{01}} \quad (3)$$

$$FMI(\mathcal{X}, \mathcal{Y}) = \frac{N_{11}}{\sqrt{(N_{11} + N_{10})(N_{11} + N_{01})}} \quad (4)$$

In order to compute CM-based metrics, e.g., *F-Measure* and *normalized Van Dongen* (NVD for short) [22], the best matched community in the benchmark \mathcal{Y} is needed to be found for each community in \mathcal{X} . For each $X_r \in \mathcal{X}$ and $Y_s \in \mathcal{Y}$, let $pre = \frac{|X_r \cap Y_s|}{|Y_s|}$ and $rec = \frac{|X_r \cap Y_s|}{|X_r|}$ measure the precision and recall respectively, and $\mathcal{F}(X_r, Y_s) = \frac{2 \cdot pre \cdot rec}{pre + rec} = \frac{2|X_r \cap Y_s|}{|X_r| + |Y_s|}$ represents the harmonic average between pre and rec , then F-Measure and NVD can be calculated using Eqs. 5 and 6 respectively.

$$F\text{-measure}(\mathcal{X}, \mathcal{Y}) = \frac{1}{n} \sum_{X_r \in \mathcal{X}} |C_i| \max_{Y_s \in \mathcal{Y}} \frac{2|X_r \cap Y_s|}{|X_r| + |Y_s|} \quad (5)$$

$$NVD(\mathcal{X}, \mathcal{Y}) = 1 - \frac{1}{2n} \left[\sum_{X_r \in \mathcal{X}} \max_{Y_s \in \mathcal{Y}} |X_r \cap Y_s| + \sum_{Y_s \in \mathcal{Y}} \max_{X_r \in \mathcal{X}} |Y_s \cap X_r| \right] \quad (6)$$

Finally, the MI-based metrics, e.g., *Normalized Mutual Information (NMI)* [23] and *Variation of Information (VI)* [24], concern more about the entropy and mutual information between communities. When such metrics are computed, each vertex is assumed to be put in a community $X_r \in \mathcal{X}$ with the same probability $P(r) = \frac{|X_r|}{n}$. Let $H(\mathcal{X})$ denote the entropy for \mathcal{X} , then we have $H(\mathcal{X}) = -\sum_{r=1}^K P(r) \log P(r)$, which measures the expected uncertainty in \mathcal{X} . Mutual information between \mathcal{X} and \mathcal{Y} , denoted as $I(\mathcal{X}, \mathcal{Y})$, measures the mutual dependencies between \mathcal{X} and \mathcal{Y} , which can be calculated by $I(\mathcal{X}, \mathcal{Y}) = \sum_{r=1}^K \sum_{s=1}^{K'} P(r, s) \log \frac{P(r, s)}{P(r)P(s)}$, where $P(r, s) = \frac{|X_r \cap Y_s|}{n}$. NMI between \mathcal{X} and \mathcal{Y} , denoted as $NMI(\mathcal{X}, \mathcal{Y})$, normalizes the mutual information $I(\mathcal{X}, \mathcal{Y})$ and concerns the maximum average normalized mutual information of all communities in both \mathcal{X} and \mathcal{Y} [23], which can be calculated using Eq. 7.

$$NMI(\mathcal{X}, \mathcal{Y}) = \frac{2I(\mathcal{X}, \mathcal{Y})}{H(\mathcal{X}) + H(\mathcal{Y})} \quad (7)$$

VI between \mathcal{X} and \mathcal{Y} , denoted as $VI(\mathcal{X}, \mathcal{Y})$, measures the mutual dependencies based on the information loss [24], which can be calculated using Eq. 8.

$$VI(\mathcal{X}, \mathcal{Y}) = H(\mathcal{X}) + H(\mathcal{Y}) - 2I(\mathcal{X}, \mathcal{Y}) = [H(\mathcal{X}) - I(\mathcal{X}, \mathcal{Y})] + [H(\mathcal{Y}) - I(\mathcal{X}, \mathcal{Y})] \quad (8)$$

The workflow of computing aforementioned CQA metrics includes two main steps: (1) Compute all pair-community intersection sizes between \mathcal{X} and \mathcal{Y} ; and (2) Compute the rest part of the metrics, which can also involve high complexity computation, e.g., finding all the best-matched pair-community for NVD. The computation complexity of both steps are all high. The rapid growth on the size of social networks presents great challenges to the metrics computation, *MPI based* algorithm [11] uses parallel and distributed computation model MPI to decompose the computation task into smaller ones, and then distribute them to execute on multiple distributed hosts simultaneously, which improves their efficiency and scalability. When this approach is applied, communities in \mathcal{X} and \mathcal{Y}

are partitioned into p groups respectively, each one will be sent to a process for *pair-community comparison-based* [25] all pair-community intersection size computation. After a process completes its processing on the local data, it will exchange its corresponding part of \mathcal{Y} to each others and continues to compute the intersection size between its corresponding group communities from \mathcal{X} and the newly received group ones from \mathcal{Y} . After a process completes the intersection size computation on its group communities of \mathcal{X} and those of \mathcal{Y} , it will calculate the rest part of metrics based on the local data. Finally all the local results will be accumulated for the final metrics computation. The pair-community comparison-based all pair-community intersection size computation in this approach requires $K_L \times K'$ times comparison, where K_L is the size of the partition of \mathcal{X} for a process. This high computation complexity degrade its efficiency.

We notice that *igraph* [9] and *scikit-learn* [8] use *pair-cid counting based* approach for the all-pair community intersection size computation. Let the *community id* (*cid* for short) of X_r be r , and let $C_i^{\mathcal{X}}$ represent the cid of v_i in \mathcal{X} . For any v_i , its pair-cid for \mathcal{X} and \mathcal{Y} is denoted as a tuple $(C_i^{\mathcal{X}}, C_i^{\mathcal{Y}})$. Let sequence $S = \langle (C_1^{\mathcal{X}}, C_1^{\mathcal{Y}}), \dots, (C_n^{\mathcal{X}}, C_n^{\mathcal{Y}}) \rangle$, S is a repeat sequence, $(C_i^{\mathcal{X}}, C_i^{\mathcal{Y}}) = (C_k^{\mathcal{X}}, C_k^{\mathcal{Y}})$ means $\{v_i, v_k\} \subset (X_{C_i^{\mathcal{X}}} \cap Y_{C_i^{\mathcal{Y}}})$ since $v_i, v_k \in X_{C_i^{\mathcal{X}}}$ and $v_i, v_k \in Y_{C_i^{\mathcal{Y}}}$. That is, the frequency of tuple $(C_i^{\mathcal{X}}, C_i^{\mathcal{Y}})$ in S is $|X_{C_i^{\mathcal{X}}} \cap Y_{C_i^{\mathcal{Y}}}|$. Therefore, $|X_{C_i^{\mathcal{X}}} \cap Y_{C_i^{\mathcal{Y}}}|$ can be calculated by counting the occurrence frequency of $(C_i^{\mathcal{X}}, C_i^{\mathcal{Y}})$ in S . The computation complexity of all-pair community intersection size using this approach is $O(n)$. But *igraph* and *scikit-learn* are all sequential algorithms.

The objectives of our research include two folds: (1) Design an improved MPI based CQA metrics computation, combing pair-cid counting based all pair-community intersection size computation and MPI technologies to further expedite the CQA metrics computation. (2) Design a RESTful framework to wrap the CQA metrics computation as a Web service, i.e., *Non-overlapping Commu-Metrics*, to offload the computation complexities from data scientists and help them focus on data analysis, since the management of MPI based algorithm execution needs extensive computational domain knowledge, e.g., configuration, deployment, etc.

3 An Improved MPI-Based Algorithm for Non-overlapping CQA Metrics Computation

Before the description of the algorithm, we first present the relevant notations. Let p denote the number of processes, tN_{11} , tN_{10} , tN_{01} , $tNVD(\mathcal{X}, \mathcal{Y})$, $tF\text{-measure}(\mathcal{X}, \mathcal{Y})$, $t\mathcal{I}(\mathcal{X}, \mathcal{Y})$, $tH(\mathcal{X})$, and $tH(\mathcal{Y})$ denote the corresponding intermediate N_{11} , N_{10} , N_{01} and $NVD(\mathcal{X}, \mathcal{Y})$, $F\text{-measure}(\mathcal{X}, \mathcal{Y})$, $\mathcal{I}(\mathcal{X}, \mathcal{Y})$, $H(\mathcal{X})$ and $H(\mathcal{Y})$ by each process computing the local datasets respectively.

To improve the CommuMetrics CQA metrics Web service provisioning, we improve the metrics computation efficiency by two ways. First of all, it computes all the metrics introduced in Sect. 2, which involves the all pair-community

intersection size computation. If the results can be shared by all the metrics to be computed, the overall execution time can certainly be reduced. However, igrph and scikit-learn only output some of the metrics, e.g., scikit-learn output only ARI, FMI, and NMI. MPI-based algorithm compute 7 metrics, but it uses three algorithms to compute the PC-based, CM-based and MI-based metrics respectively. That is, when all the metrics are needed, multiple algorithms are needed for execution, which further prolongs the execution time. Second, we will combine the advantages of the pair-cid counting based all pair-community intersection size computation and MPI-based parallel and distributed computation. To put the discussion into perspective, a small artificial dataset \mathcal{X}' and \mathcal{Y}' is used as an example to illustrate how their metrics ARI and NMI are computed using the improved algorithm, which is depicted in Fig. 1.

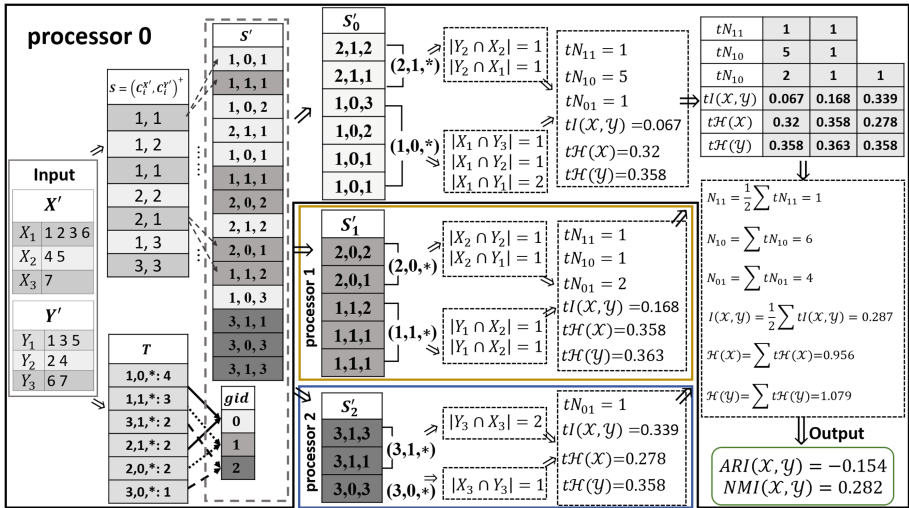


Fig. 1. Computing ARI and NMI with the improved MPI-based algorithm.

The metrics computation exploiting the pair-cid counting based all pair-community intersection size computation includes the following three steps: (1) Construct the pair-cid sequence S based on \mathcal{X} and \mathcal{Y} ; (2) Compute the all pair-community intersection size; (3) Complete the rest computation of metrics. The parallelism of the first two steps are straightforward, dataset are split into multiple smaller groups, each one will be distributed to a process for independently obtaining the cid and occurrence frequency of pair-cid based on the local data respectively, and then the local results will be accumulated to obtain the all pair-community intersection size. We actually construct S sequentially since its computation complexity is low and parallelism is needed. But for the third step, appropriate data grouping strategy is needed.

In order to reduce the communication among MPI processes, the tuples in S with the same cid in \mathcal{X} should be categorized into the same group and sent

to the same process for computing intersection size and the rest of the metrics, which is to ensure that for each X_r , its intersection size with each $Y_i \in \mathcal{Y}$ (for PC-based), its intersection size with its best matched $Y_i \in \mathcal{Y}$ (for NVD), its best harmonic average (for F-Measure), and its entropy and mutual information with each $Y_i \in \mathcal{Y}$ (for MI-based) will be calculated. Meanwhile, the tuples in S with the same cid in \mathcal{Y} should be categorized into the same group and sent to the same process to ensure that each Y_i , its intersection size and mutual information with each $X_i \in \mathcal{X}$ will also have to be calculated for PC-based and MI-based metrics computation.

It is hard to group the tuples in S to satisfy the above criteria. We re-represent the tuples in S and construct a new sequence S' , targeting at achieving parallelism for the following computations. Specifically, each tuple $(C_i^{\mathcal{X}}, C_i^{\mathcal{Y}})$ in S can be re-represented using two new tuples $(C_i^{\mathcal{X}}, f, C_i^{\mathcal{Y}})$ and $(C_i^{\mathcal{Y}}, f, C_i^{\mathcal{X}})$ in S' , where f is flag. $f = 0$ indicates that the first cid belongs to \mathcal{X} and $f = 1$ indicates that it belongs to \mathcal{Y} . As illustrated in Fig. 1, tuple $(1, 1)$ in S is re-represented as two tuples $(1, 0, 1)$ and $(1, 1, 1)$ in S' . For a particular X_r , there will be $|X_r|$ tuples with the prefix $(r, 0, *)$ in S' . That is, the larger a community X_r is, the more tuples with the same prefix $(r, 0, *)$ appear in S' . Similarly, the larger a community Y_s is, the more tuples with the same prefix $(s, 1, *)$ appear in S' . Since the tuple with the same prefix $(r, 0, *)$ or $(s, 1, *)$ will be categorized into the same group, in order to balance the number of tuples partitioned to each processors, the size of each community is collected in sequence $T = \{t_1, \dots, t_{K+K'}\}$, where t_i is $(r, 0, |X_r|)$ or $(s, 1, |Y_s|)$ and T is sorted in descending order of the community size. A tuple t_i can be uniquely identified by its prefix (cid, f) in T , let $seq(cid, f)$ denote the index of its corresponding t_i with prefix $(cid, f, *)$ in the sequence T , $g(cid, f)$ denote the group id (gid for short) of t_i , then $g(cid, f) = seq(cid, f) \bmod p$. Therefore the tuples in S' are categorized to the group using the same strategy $g(cid, f) = seq(cid, f) \bmod p$. That is, for a tuple $(cid, f, *)$ in S' , it will be sent to process with number $g(cid, f) = seq(cid, f) \bmod p$ for processing. This hash-mapping strategy scatters tuples derived from large communities to each process so as to balance the workload, i.e., the number of tuples to be processed. As illustrated in Fig. 1, tuples $(2, 0, *)$ will be processed by process 1 since $(seq(2, 0) \bmod 3 = 4 \bmod 3 = 1)$. In addition, we can also see that the tuples derived from the 3 largest communities X_1, Y_1 and Y_3 are scattered to be processed by process 0, 1, and 2 respectively.

Process i ($0 \leq i \leq (p - 1)$) gets a group of tuples of S' , denoted as S'_i , and orders the tuples in dictionary descending order of their prefix (cid, f) . As described in Sect. 2, the occurrence frequency of tuples can be calculated as the corresponding two communities's intersection size. As illustrated in Fig. 1, tuple $(3, 0, 3)$ appears only once in S'_2 in processor 2, meaning that $|X_3 \cap Y_3| = 1$. Then each processor calculates tN_{11} , tN_{10} , tN_{01} , $tF\text{-measure}(\mathcal{X}, \mathcal{Y})$, $tNVD(\mathcal{X}, \mathcal{Y})$, $tI(\mathcal{X}, \mathcal{Y})$, $tH(\mathcal{X})$, and $tH(\mathcal{Y})$ accordingly (refer to Eqs. 1–8). Finally, process 0 accumulates all local results of the other processors, together with its own local results and outputs the metrics accordingly.

Algorithm 1: The improved MPI-based algorithm.

Input: $\mathcal{X} = \{X_1, X_2, \dots, X_K\}$, $\mathcal{Y} = \{Y_1, Y_2, \dots, Y_{K'}\}$
Output: A full set of CQA metrics for \mathcal{X} and \mathcal{Y} , including RI, ARI, JI, RMI, F-measure, NVD, NMI, VI.

- 1 processor 0:
- 2 Construct sequence S and S' for \mathcal{X} and \mathcal{Y} ;
- 3 Partition S' into S'_0, \dots, S'_{p-1} ;
- 4 Sent S'_1, \dots, S'_{p-1} to processor $1, \dots, p-1$;
- 5 **forall the processor i in $\{0, \dots, (p-1)\}$ do**
- 6 Calculate all pair-community intersection size for S'_i
- 7 Calculate local results $tN_{11}, tN_{10}, tN_{01}, tNVD(\mathcal{X}, \mathcal{Y}), tF\text{-measure}(\mathcal{X}, \mathcal{Y}), tI(\mathcal{X}, \mathcal{Y}), tH(\mathcal{X})$, and $tH(\mathcal{Y})$.
- 8 Send local results to processor 0
- 9 processor 0:
- 10 Calculate and output all metrics based on Eqs. 1–8.

In all, the improved MPI-based algorithm is described in Algorithm 1. Specifically, in steps 2–4, process 0 construct the consequence S and S' sequentially since the computation cost here is low, which takes only 1–2 s. In steps 6–8, p processes execute in parallel for all the intermediate pair counting, harmonic average with best matched community, mutual information and entropy calculation. In step 10, process 0 accumulates all the local results and output all the CQA metrics. Here, we can see that our proposed algorithm has two main advantages: (1) Combine the advantages of *pair-cid counting based* and MPI-based metric computation; (2) Systematically organize the metrics' computation, remove the redundant computation and output all metrics together.

4 A RESTful Web Service for Non-overlapping CQA Metrics Computation

4.1 System Model

Figure 2 depicts the system model of the *non-overlapping CommuMetrics* (*CommuMetrics* for short), a RESTful non-overlapping CQA Web service. *CommuMetrics* provides a web based metrics computation services for *data scientists* and *decision makers*, who can use browsers (e.g., *IE*, *Chrome*, etc.) and *HTTP* protocol to upload their communities and benchmark, then send request for desired CQA metrics, then *CommuMetrics* backend will process the computation and return the calculated metrics. The interaction provided by the user interface is quite simple, only around four buttons *Add* for communities dataset specification, *Upload* for transfer the dataset to backend for processing, and *Run* for request submissions. The data scientists and decision makers can obtain high performance metrics computation via a few click operations.

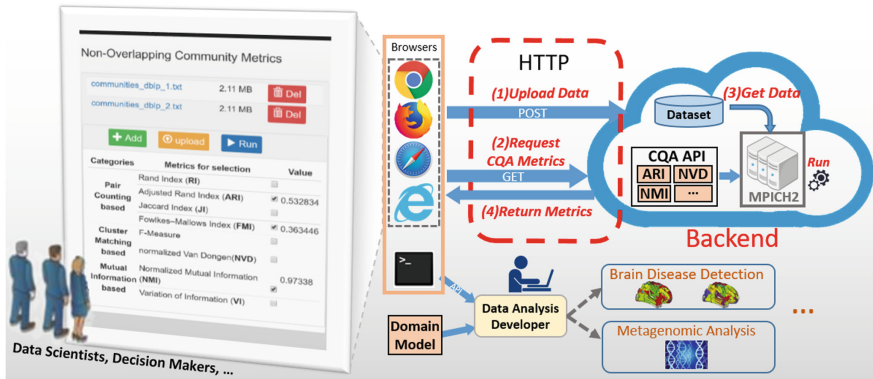


Fig. 2. The system model of the RESTful non-overlapping CQA Web service.

Meanwhile, CommuMetrics also provide metrics computation API for *data analysis developer*. Here, all the uploaded communities datasets and corresponding CQA metrics are regarded as *Resource*, each resource has a specific and unique name, represented as *Universal Resource Identifier* (URI). CommuMetrics provides RESTful API using URIs for accessing any services. Combing domain knowledge and the APIs, developers can focus on domain specific data analysis, e.g., brain disease detection and microbial metagenomic sequence data analysis.

4.2 Prototype Implementation

As a proof of concept, we provide an implementation of CommuMetrics Web service prototype. First, the backend of CommuMetrics is developed upon Flask², a lightweight web application framework based on Python. Flask is used to match the URI of a HTTP request to its corresponding business logic function. For example, when *Upload* is clicked for uploading a local dataset for metrics computation, Flask will invoke the corresponding function to accept and store it.

The web client of CommuMetrics is developed using HTML, CSS and JavaScript. Jinja2³, a website template engine, is used to re-render the response web pages from Flask. The web client attaches the RESTful APIs for uploading community datasets and requesting CQA metrics computation into buttons, i.e., *Upload* and *Run*, as illustrated in Fig. 2. When a buttons is clicked, e.g., button *Run* for submitting a metrics computation request, it will communicate with the backend services via HTTP protocol, and invoke the remote corresponding python functions using jQuery AJAX, then the backend provisions the metrics computation services using the computation algorithms, e.g., the improved MPI-based metrics computations, or the existing igraph, scikit-learn, or MPI-based algorithms.

² <http://flask.pocoo.org/>.

³ <http://jinja.pocoo.org/>.

5 Performance Evaluation

The CQA service provisioning efficiency is measured by the absolute *execution time* and relative *speedup* for metrics computation. First, execution time evaluates how much time the CQA metrics computation takes, the shorter the execution time, the more efficient the underlying algorithm is. Second, speedup includes the *scale-out speedup* and *scale-up speedup*. The scale-out speedup is the ratio of the execution time by using one machine against the one by using m machines, which measures the improvement in execution speed when the number of computers increases. When the number of machines is fixed, the scale-up speedup is the ratio of the execution time by using one core per computer against the one by using k cores, which measures the improvement in execution speed when available hardware resources for a computer increases.

5.1 Dataset and Testbed

Communities detected from artificial networks and the benchmark communities are used as datasets. First, LFR proposed algorithms is applied to generate networks with specified parameters for mimicking real network properties [26], e.g., the vertex number ranging from 3.1×10^5 to 1.06×10^8 , average vertex degree ranging from 30 to 2×10^4 , exponent for the vertex degree distribution $\gamma = -2$, community size distribution $\beta = -1$, and the mixing parameter $\mu = 0.35$. *GossipMap* [27], an efficient community detection algorithm, is applied on the generated networks for community detection. The features of the benchmark and detected communities are summarized in Table 1, including the number of vertices (n), the maximum number of vertices in communities ($M(\mathcal{X})$ and $M(\mathcal{Y})$), the number of communities (K and K'), the number of unique tuples in sequence S ($U(S)$) and the percentage of unique tuples in S ($P(S) = \frac{U(S)}{n}$).

Table 1. The features of the benchmark and detected communities

Network name	n	Ground truth		GossipMap		$U(S)$	$P(S)(\%)$
		$M(\mathcal{Y})$	K'	$M(\mathcal{X})$	K		
dblp	317080	6596	61483	9267	61251	153141	48.3%
amazon	334863	371	53245	477	53074	138845	41.5%
youtube	1134890	81267	172326	84763	172177	372361	32.8%
wiki	1791489	151052	5027	153417	5190	24489	1.4%
orkut	3072441	734132	4115	633283	4542	34053	1.1%
livejournal	3997962	286711	241220	444612	234066	779619	29.5%
friendster	65608366	967366	7884397	950259	8269426	30798609	46.9%
uk200705	105896555	272433	6445260	277820	6437726	35051759	33.1%

Our experimental testbed is a cluster consisting of 16 DOWN TC4600 blades, each of which has 2 10-core Intel(R) Xeon(R) E5-2680 v2 2.8 GHz CPUs. Meanwhile, each blade has 2×1 TB hard disk, 64 GB RAM and 1 Gbit Ethernet.

The blades are installed with CentOS 2.6.32, GCC 5.4 and MPICH3 3.2.1⁴, a high performance and widely portable MPI implementation. We will use igraph, scikit-learn, MPI-based and our improved MPI-based algorithms to compute the CQA metrics.

5.2 Execution Time

Figure 3 depicts the execution time of our improved MPI-based algorithm against that of *scikit-learn*, *igraph* and existing *MPI-based* over the 8 detected communities, where the y-axis is set as log scale, and the cross indicates that the algorithm cannot finish its computation within 10,000s over the corresponding dataset. From the experimental results, we can have the following findings.

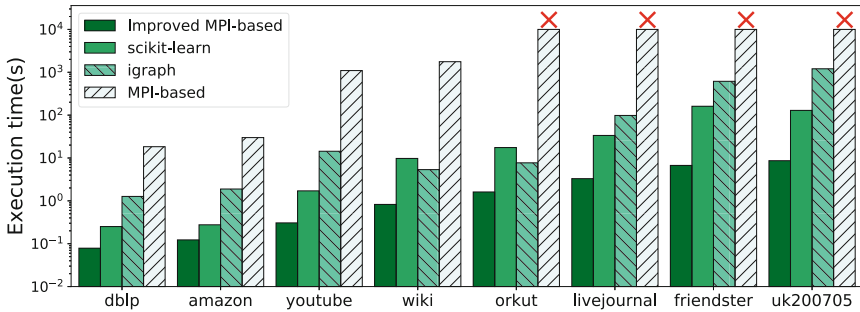


Fig. 3. The execution time of metrics computation.

- The execution time of the improved MPI-based algorithm is always shorter than those of other 3 algorithms. Meanwhile, we can also see that the increase of the execution time of the improved MPI-based algorithm is less than those of the other 3 algorithms as the number of vertices increases. As mentioned in Sect. 2, MPI-based methods requires $K_L \times K'$ times comparison to compute the all pair-community intersection size. Since all the datasets for our experiment are with large number of communities and big size community, though it uses MPI for parallel computation, its execution efficiency is actually not good. Combining the efficiency of pair-cid counting based all pair-community intersection size computation and the parallel computation of MPI, the improved MPI-based algorithm outperforms the other 3 algorithms in terms of execution time.
- The execution time is affected by not only the number of vertices of the network (i.e., n), but also the number of pair-communities between \mathcal{X} and \mathcal{Y} (i.e., $K \times K'$). For example, scikit-learn completes its execution faster on dataset uk200705 than friendster, though the number of vertices of

⁴ <https://www.mpich.org/>.

uk200705 is bigger than that of friendster. The number of pair-communities of uk200705 is 6445260×6437726 , which is less than that of friendster, i.e., 7884397×8269426 .

From the execution results, we can also see that the execution time of the igraph is longer than that of scikit-learn for most datasets except for wiki and orkut datasets. The reason is that igraph and scikit-learn use different data structure to store the computed all pair-community intersection size, where igraph uses *adjacent list* and scikit-learn uses *contingency tables*. The computation cost for constructing the adjacent list depends on the datasets. As described in Table 1, the percentage of unique tuples in S , i.e., $P(S)$, for wiki and orkut is 1.4% and 1.1% respectively, which is far less than that of the other datasets. Small $P(S)$ means small actual size of the adjacent list, and thus less construct time and less pair-community intersection size query time.

5.3 Speedup

The speedup performance of the improved MPI-based and the existing MPI-based algorithms are compared using the wiki dataset. First, we compare that by increasing the number of blades in the cluster from 1 to 16, where only core of each blade will be used for process execution. From Fig. 4(a), we can see that the improved MPI-based algorithm achieves better speedup. The speedup starts to decrease for the existng MPI-based algorithm takes place when the number of blades is larger than 10 while there is a slowdown in speedup improvement for the improved MPI-based algorithm when the number of blades is larger than 14. Second, we use only one blade and compare the speedup by increasing the number of cores in the blade from 1 to 20. From Fig. 4(b), we can see that the improved MPI-based algorithm achieves much better speedup.

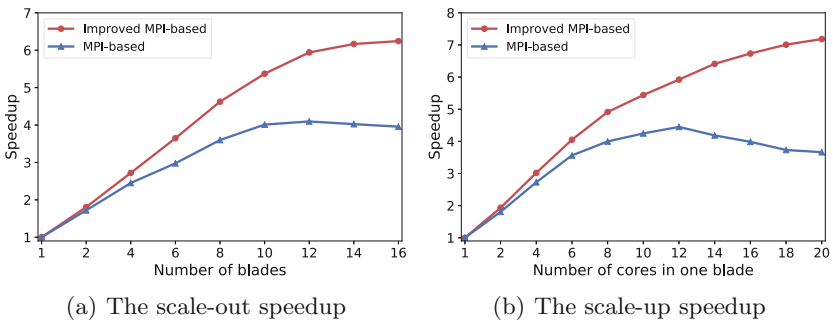


Fig. 4. The speedup of the MPI-based metrics computation over wiki dataset.

The reason is that the p processes in existing MPI-based algorithm involve massive data transmission between processes, which degrades the overall performance. Specifically, existing MPI based algorithm partition the communities in

\mathcal{X} and \mathcal{Y} into p parts respectively. For example, $\mathcal{X} = \{X'_1, X'_2, \dots, X'_p\}$, where $p \leq K$ and $p \leq K'$. For any X'_i , $X'_i = \{X_{i_1}, \dots, X_{i_l}\}$, where $X_{i_j} \in \mathcal{X}$ and $|X'_i| \geq 1$. Similarly, $\mathcal{Y} = \{Y'_1, Y'_2, \dots, Y'_p\}$. For any X'_i, Y'_j , they will be sent to a process for the intersection size computation of pair communities between X'_i and Y'_j . After each process completes its execution, it will send the corresponding Y'_j to the other processes, then each process will continue to compute the intersection size of pair communities between its corresponding X'_i and the newly received Y'_k . However, as illustrated in Fig. 1, with our data grouping strategy, the p processes in the improved MPI-based algorithm are actually executed independently, no data transmission is needed in between.

6 Conclusions

A thorough comparison of different community detection methods is critical for the analysis of networked systems, where community quality assessment (CQA) is one of the key performance metrics.

Popular used non-overlapping CQA metrics include RI, ARI, JI, F-measure, RMI, NVD, NMI and VI, whose computation complexities are high. MPI has been exploited to expedite the computation, yet, the efficiency needs further improvement. Meanwhile, MPI based implementation usually calls for computational domain knowledge like load balancing, effective resource utilization, etc., such complicated details handicap data scientists from adopting it.

In this paper, we first propose an improved MPI-based algorithm to improve the CQA metrics computation: (1) It computes all the aforementioned popular used metrics together, where the results of common sub-routine computation are shared. Existing algorithms only output part of the metrics, e.g., scikit-learn output only ARI, FMI, and NMI. (2) It combines the advantages of existing algorithms, using igraph and scikit-learn's fast pair-cid counting based all pair-community intersection size computation, and MPI-based parallel and distributed computation. Then we design a RESTful framework to wrap the CQA metrics computation as a Web service for non-overlapping communities, i.e., CommuMetrics, which make CQA metrics computation can be accessed in an easy way. The results of our empirical comparison study demonstrate its efficiency.

Acknowledgment. We appreciate the support by National Key R&D Program of China (2018YFB1003201), Guangdong Natural Science Foundation (2016A030313036), 2017 Guangdong Undergraduate Teaching Quality and Teaching Reform Project (839-0000026812), Shenzhen Science and Technology Foundation (JCYJ2015032414003 6842, JCYJ20170302153955969, JCYJ20150529164656096, JCYJ20160520162743717), National Natural Science Foundation of China (61103001, 61672358, 61703281, 11547040), Shenzhen science and technology project (JCYJ20150529164656096, JCYJ20170302153955969), Guangdong Province Key Laboratory of Popular High Performance Computers (2017B030314073), Shenzhen Peacock Talent Grant (827-000175) and Guangdong Pre-national project (2014GKXM054), the PhD Start-up Fund of Natural Science Foundation of Guangdong Province, China (2017A030310374) The Young Teachers Start-up Fund of Natural Science Foundation of Shenzhen University.

References

1. Newman, M.E., Girvan, M.: Finding and evaluating community structure in networks. *Phys. Rev. E* **69**(2), 026113 (2004)
2. Newman, M.E.: Modularity and community structure in networks. *Proc. Natl. Acad. Sci. USA* **103**(23), 8577–8582 (2006)
3. Newman, M.E.: Community detection and graph partitioning. *EPL (Europhys. Lett.)* **103**(2), 28003 (2013)
4. Rohe, K., Chatterjee, S., Yu, B.: Spectral clustering and the high-dimensional stochastic blockmodel. *Ann. Stat.* **39**(4), 1878–1915 (2011)
5. Amini, A.A., Levina, E.: On semidefinite relaxations for the block model. *Ann. Stat.* **46**(1), 149–179 (2018)
6. Xie, J., Szymanski, B.K., Liu, X.: SLPA: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In: 2011 IEEE 11th International Conference on Data Mining Workshops (ICDMW), pp. 344–349 (2011)
7. Girvan, M., Newman, M.E.: Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA* **99**(12), 7821–7826 (2002)
8. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, É.: Scikit-learn: machine learning in python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011)
9. Csardi, G., Nepusz, T.: The igraph software package for complex network research. *InterJ. Complex Syst.* **1695**(5), 1–9 (2006)
10. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Comput.* **22**(6), 789–828 (1996)
11. Chen, M., Liu, S., Szymanski, B.K.: Parallel toolkit for measuring the quality of network community structure. In: 2014 European Network Intelligence Conference (ENIC), pp. 22–29 (2014)
12. Statista (The Statistics Portal): Number of monthly active Facebook users worldwide as of 4th quarter 2017 (in millions) (2017). <http://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/>. Accessed 05 May 2018
13. Fielding, R.T., Taylor, R.N.: Architectural Styles and the Design of Network-Based Software Architectures, vol. 7. University of California, Irvine Doctoral dissertation (2000)
14. Hu, S.C., Chen, I.C., Lin, Y.L.: Designing a restful question bank service in cloud. In: 2011 International Conference on Multimedia Technology, pp. 5803–5806 (2011)
15. Tanna, R.D., Kumar, K.S.V., Karthika, S.: Analytics as a service for beginners. In: 2017 International Conference on Computational Intelligence in Data Science (ICCIDS), pp. 1–6 (2017)
16. Ko, R.K., Kirchberg, M., Lee, B.S., Chew, E.: Overcoming large data transfer bottlenecks in restful service orchestrations. In: 2012 IEEE 19th International Conference on Web Services, pp. 654–656 (2012)
17. Wagner, S., Wagner, D.: Comparing Clusterings: An Overview. Universität Karlsruhe, Fakultät für Informatik Karlsruhe (2007)
18. Rand, W.M.: Objective criteria for the evaluation of clustering methods. *J. Am. Stat. Assoc.* **66**(336), 846–850 (1971)
19. Hubert, L., Arabie, P.: Comparing partitions. *J. Classif.* **2**(1), 193–218 (1985)

20. Ben-Hur, A., Elisseeff, A., Guyon, I.: A stability based method for discovering structure in clustered data. In: Pacific Symposium on Biocomputing, vol. 7, pp. 6–17 (2001)
21. Fowlkes, E.B., Mallows, C.L.: A method for comparing two hierarchical clusterings. *J. Am. Stat. Assoc.* **78**(383), 553–569 (1983)
22. Dongen, S.: Performance criteria for graph clustering and Markov cluster experiments. Technical report, CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands (2000)
23. Strehl, A., Ghosh, J.: Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.* **3**, 583–617 (2003)
24. Meilă, M.: Comparing clusterings by the variation of information. *Learn. Theory Kernel Mach.* **3**, 173–187 (2003)
25. Bille, P., Pagh, A., Pagh, R.: Fast evaluation of union-intersection expressions. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 739–750. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77120-3_64
26. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection, June 2014. <http://snap.stanford.edu/data>
27. Bae, S.H., Howe, B.: GossipMap: a distributed community detection algorithm for billion-edge directed graphs. In: 2015 SC-International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–12 (2015)