Vol 15 No 11, November 2006
1009-1963/2006/15(11)/2489-07
Chinese Physics
© 2006 Chin. Phys. Soc.
and IOP Publishing Ltd

The complexity nature of large-scale software systems*

Yan Dong(闫 栋), Qi Guo-Ning(祁国宁), and Gu Xin-Jian(顾新建)

Key Laboratory of Advanced Manufacture Technology of Zhejiang Province, Zhejiang University, Hangzhou 310027, *China*

(Received 11 December 2005; revised manuscript received 3 July 2006)

In software engineering, class diagrams are often used to describe the system's class structures in Unified Modelling Language (UML). A class diagram, as a graph, is a collection of static declarative model elements, such as classes, interfaces, and the relationships of their connections with each other. In this paper, class graphs are examined within several Java software systems provided by Sun and IBM, and some new features are found. For a large-scale Java software system, its in-degree distribution tends to an exponential distribution, while its out-degree and degree distributions reveal the power-law behaviour. And then a directed preferential-random model is established to describe the corresponding degree distribution features and evolve large-scale Java software systems.

Keywords: software system, degree distribution, power-law behaviour, exponential distribution
PACC: 0250, 0565

1. Introduction

There are many phenomena of 'the rich get richer' in the world, e.g. the World Wide Web (WWW), Internet, citations of the most cited physicists and so on. Watts and Strogatz[1] proposed a novel small-world model in 1998. The WWW is an enormous virtual network of Web pages connected by hyperlinks. Furthermore, Barabási and Albert[2−4] also studied the WWW and proposed a distinguished scale-free network model in 1999, which can perfectly interpret these phenomena of 'the rich get richer'. According to statistics of the WWW by Broder *et al*[5] and Albert *et al*,[6] most Web pages are connected by a small number of hyperlinks, while only a few Web pages are connected by a large number of hyperlinks. The degree distribution $p(k)$ of a network generated by the scale-free model has a k power law decay, $p(k) \sim k^{-\gamma}$, with $\gamma > 0$. The mechanisms leading to the power-law distribution are considered to be growth and preferential attachment,[2−4] which are the fundamental principles of many open systems growing with the continuous addition of new nodes. Since then, more researchers have further studied the scale-free model and more complex systems, and have brought forward many new interesting ideas with unprecedented insights into network evolution and topology. Dai and Guo proposed two models for improving the robustness of networks.[7] Huang *et al* studied correlations in scale-free networks and proposed a correlation measurement in bipartite networks.[8] Recently, more and more realistic systems[1,2,4,9−18] have been studied.

In this paper, the network characteristics of large-scale software systems are studied. In Section 2, the structure of Java software systems is analysed. In Section 3, some real Java software systems provided by Sun and IBM are investigated. In Section 4, a simple directed network model is established. Finally, the new features of Java software systems are summarized in Section 5.

2. Structure of a Java software system

In class-based software systems, a class is the descriptor for a set of objects with similar structures, behaviours, and relationships, which represents a concept within the modelled system.[19] Classes are declared in class diagrams and used in many other diagrams. Note that a class diagram may also contain interfaces, packages, relationships and even instances, such as objects and links. A class diagram is a collection of static declarative model elements, such as classes, interfaces, and the relationships of their connections with each other, and their contents. The relationships between classes include 'association', 'generalization', 'realization' and 'dependency',[19] which can be regarded as the self-organization and collaborations of classes. This is simply illustrated in Fig.1.

*Project supported by the National Natural Science Foundation of China (Grant Nos 60374057 and 50575204).

C^{++} and Java languages generally establish the four relationships above respectively by the stereotype keywords 'include' and 'import'.

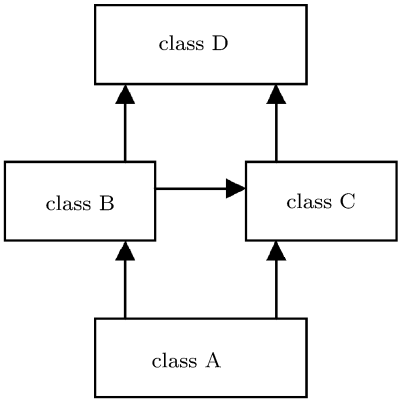


Fig.1. A simple plot of class collaborations.

In this paper, the Java language is chosen that supports the '*Write Once, Run Anywhere*' philosophy[20] claimed by Sun to observe and study relationships of classes as simply illustrated by Fig.2(a). The Java language as a class-based programming language can construct kinds of application systems independent of platforms, as further illustrated by Fig.2(b). In Fig.2(b), the corresponding documents and configuration information in systems are not taken into account. And in Fig.(2b), 'Java Method' (Java API) is mainly Java^{TM} 2 Platform Standard Edition Development Kit (JDK). The scale of JDK continuously increases with its upgrade. For instance, JDK 1.1.6 and 1.5.0 respectively consist of 668 and 6558 classes, which make it easy for programmers to develop a Java system by reusing Java API. Obviously, all versions of JDK form a series of growing networks. The JDK network grows continuously with the addition of new classes. In a software-system network, each class is a node and an edge is formed to connect two classes if they establish at least one of the four class relationships mentioned above. For Java, if class B imports class A, a directed edge from A to B is created. For class A, there are both an in-degree k_{inA}, the number of incoming edges to class A, and an out-degree k_{outA}, the number of outgoing edges from class A. The degree of each class is the sum of its in-degree and out-degree. The degree distribution indicates the probability of finding a node with a specified degree in a given network.

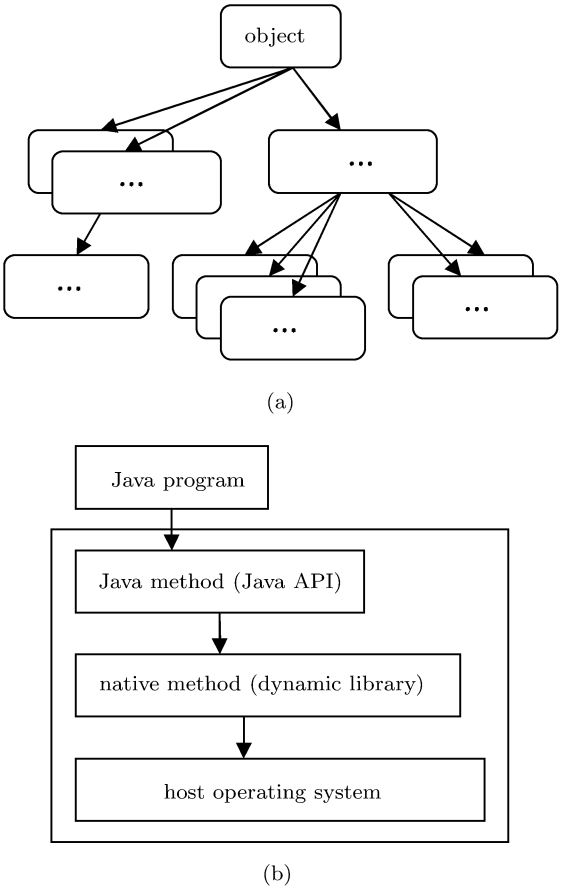


Fig.2. (a) The hierarchy of Sun JDK; (b) Java software systems independent of platforms.

3. Degree distributions of large-scale software systems

3.1. Demonstrations

The networks that are associated with all versions of Sun JDK, IBM netbeans3.6 and netbeans4.0, Globus-Toolkit (GT) core0.4 and core0.9 are examined. Figure 3 shows the in-degree, out-degree and degree distributions of Sun JDK1.5.0 ('Java API' in Fig.2(b), 6558 classes) and IBM netbeans4.0 ('Java Program' in Fig.2(b), 9510 classes). All these distribution plots become fatter with the degree distribution probability decreasing from top to bottom.

One might be tempted to fit the scattered points in each of plots in Fig.3. An alternative way to obtain a proper fit is to make a plot of the cumulative distribution. It loses any difference between the values

of data points. If $p(k)$ is defined as a degree distribution, then the cumulative distribution can be obtained as follows:

$$P_c(k) = \int_k^{+\infty} p(k)\mathrm{d}k\,. \tag{1}$$

Figure 4 shows the cumulative distributions for the Java networks mentioned above.

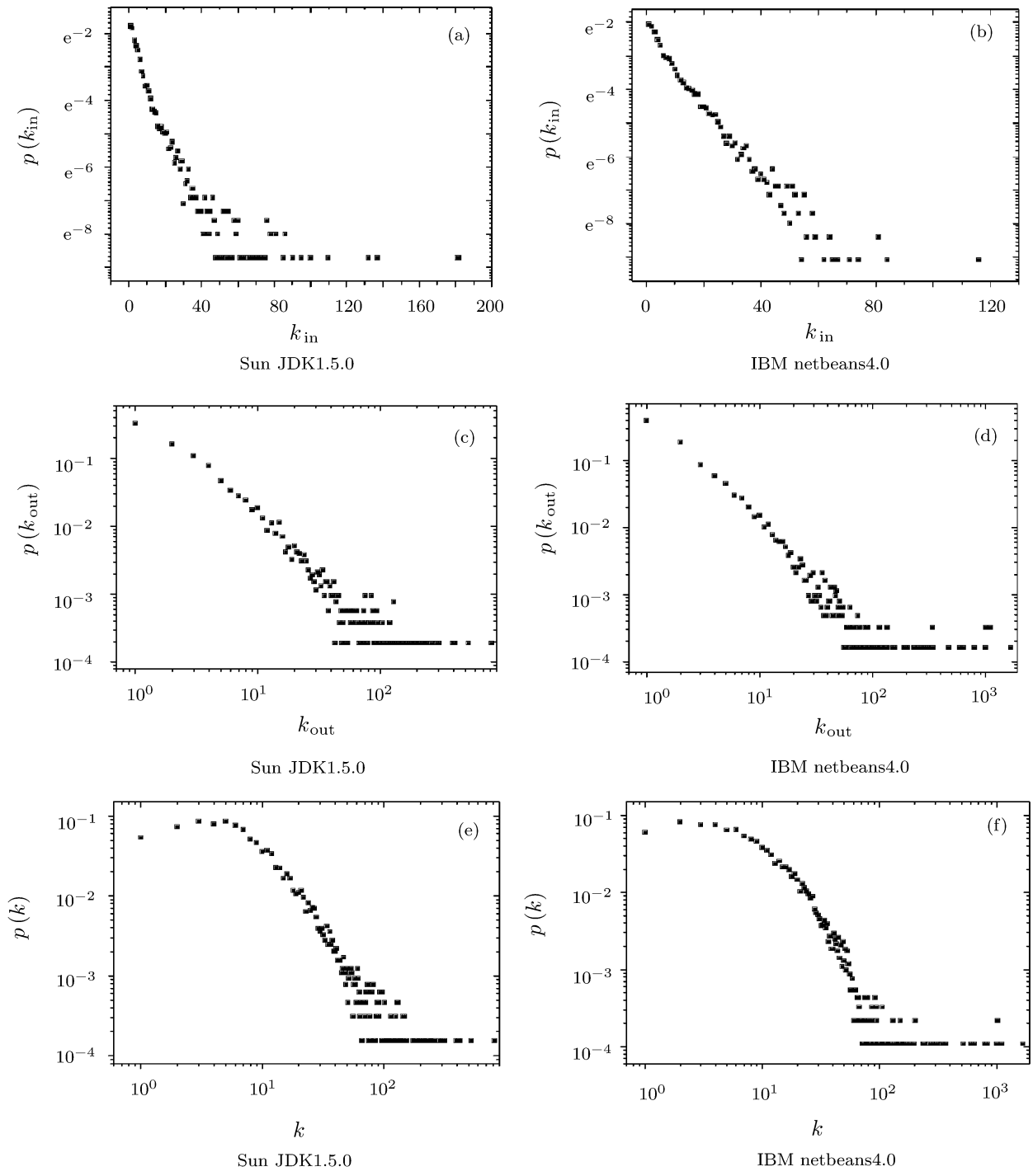


Fig.3. In-degree (a, b), out-degree (c, d)) and degree (e, f) distributions of Sun JDK1.5.0 and IBM netbeans4.0.

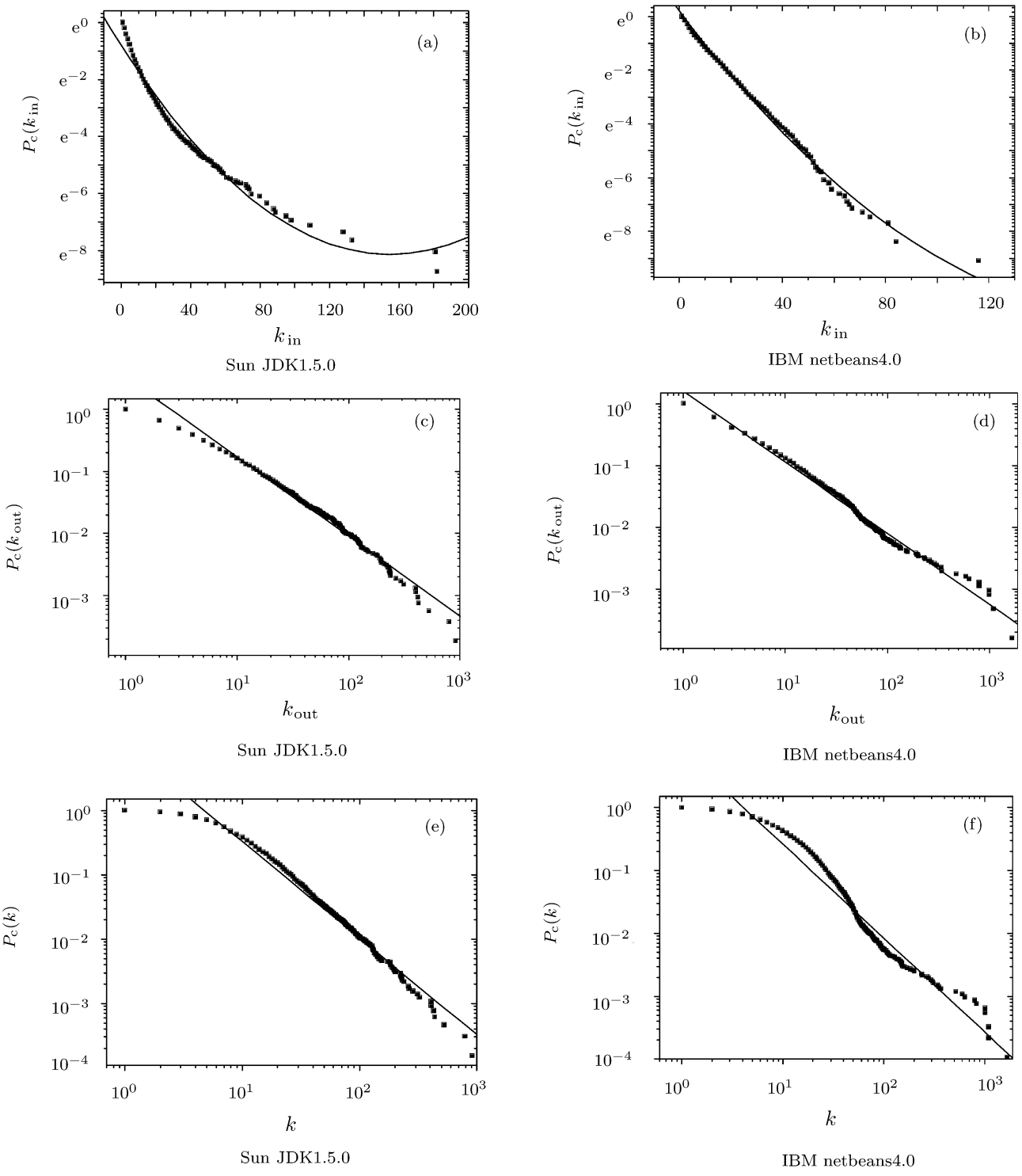


Fig.4. In-degree (a, b), out-degree (c, d) and degree (e, f) cumulative distributions of Sun JDK1.5.0 and IBM netbeans4.0.

3.2. In-degree distributions

In Figs.3 and 4, plots in (a) and (b) show the in-degree (cumulative) distributions on linear-logarithm scales, which are different from those of other plots in (c, d, e, f) on log–log scales. It indicates that they follow different distributions. Plots in (c, d, e, f) are fitted linearly. The in-degree cumulative distribution plots in (a, b) are fitted in Fig.4 by a parabola, then

$$Y = \exp(a + bx + cx^2).$$

The values of parameters a, b and c obtained by fitting are shown in Table 1. The in-degree cumulative distributions of some other software systems are also

analysed in the same way, and the following relationship between the absolute values $|b|$ and $|c|$ ($|c| \ll |b|$) is found. For simplicity,

$$\begin{aligned} &Y = \exp(a + bx), \\ &P_c(k_{\text{in}}) = \exp(a + bk_{\text{in}}), \\ &p(k_{\text{in}}) \sim \exp(bk_{\text{in}}). \end{aligned} \tag{2}$$

The in-degree distributions of the C^{++} systems examined by Myers[21] reveal the power-law behaviour. But the in-degree distributions of the Java systems examined by us tend to the exponential distribution.

Table 1. Parameter values of in-degree distributions

Parameters	a	b	c
Sun JDK1.5.0	−0.75627	−0.0934	2.98059×10^{-4}
IBM netbeans4.0	0.20705	−0.12748	3.58718×10^{-4}

3.3. Out-degree and degree distributions

In Fig.4, the plots in (c, d, e, f) reflect $Y \sim x^{-\gamma}$, then

$$p(k_{\text{out}}) \sim k_{\text{out}}^{-\gamma}, \qquad p(k) \sim k^{-\gamma}. \tag{3}$$

The degree and out-degree distributions follow the scale-free distribution without an exponential cut off[22,23] according to plots of Figs.4(c), 4(d), 4(e) and 4(f). However, there is a little difference between the out-degree and degree distributions due to a little fluctuation in the latter. According to the plots of Figs.4(e) and 4(f), only 10 percent of all classes have a higher degree than 27 in Sun JDK1.5.0 and IBM netbeans4.0, while the maximal degrees are respectively 818 and 1678. Thus, only a few classes have a high degree, while much more classes have a low degree. This again explains the phenomenon of polarization, namely 'the rich get richer'. After fitting linearly plots of Figs.4(e) and 4(f), the scaling exponents of large-scale software systems can be obtained as shown in Table 2, which in addition includes all other systems mentioned above. We find out the mean scaling exponent $\langle\gamma\rangle$ of degree distributions to be about 2.5. In a software system, the out-degree of a class can become very high, although its in-degree is perhaps low. For a large-scale software system, its maximal in-degree is always much less than its maximal out-degree. For example, the maximal in-degrees of Sun JDK1.5.0 and IBM netbeans4.0 are respectively 182 and 116, while their maximal out-degrees are 817 and 1672.

Table 2. Scaling exponents of degree distributions

Systems	γ	Systems	γ
JDK1.1.6	2.45	JDK1.4.2.06	2.49
JDK1.1.8	2.44	JDK1.5.0	2.52
JDK1.2.1	2.48	Netbeans3.6	2.51
JDK1.2.2	2.49	Netbeans4.0	2.49
JDK1.3	2.46	GT core0.4	2.55
JDK1.3.1.14	2.96	GT core0.9	2.40
JDK1.4.0	2.49		

4. Directed preferential-random model

On the basis of the distribution features of Java systems examined above, the directed preferential-random model is established as follows.

The model starts with m_0 isolated nodes, and at each time step the following operations are performed.

(i) A new node is added.

(ii) With the probability α ($0 < \alpha < 1$), add a directed edge from an existing target node i to a newly coming node, where the probability with which i is chosen is governed by the probability

$$\prod_{\text{out}}(k_{\text{out_}i}) = \frac{k_{\text{out_}i}}{\sum_j k_{\text{out_}j}}; \tag{4}$$

otherwise add a directed edge from the newly coming node to the existing node i, where i is chosen randomly and the probability with which the directed edge will point to i is written as

$$\prod_{\text{in}}(k_{\text{in_}i}) = \frac{1}{N-1}. \tag{5}$$

Repeat this process m ($m \le m_0$) times.

For a large network generated by this model, the network size $N = m_0 + t \approx t$. According to (ii), at each time step, the number $m_{\text{in_}i}$ of incoming edges to node i is $m_{\text{in_}i} = m(1-\alpha)$, and the number $m_{\text{out_}i}$ of outgoing edges from node i is $m_{\text{out_}i} = m\alpha$.

4.1. In-degree distribution

Under the approximation that $k_{\text{in_}i}$ is continuous, the continuous rate of change of $k_{\text{in_}i}$ can be written as

$$\begin{aligned} \frac{\partial k_{\text{in_}i}}{\partial t} &= m_{\text{in_}i}\prod_{\text{in}}(k_{\text{in_}i}) \\ &= \frac{m(1-\alpha)}{N-1} \approx \frac{m(1-\alpha)}{t}. \end{aligned} \tag{6}$$

The solution of this equation, with the initial condition that every node i at its introduction has $k_{\text{in_}i}(t_i) = m\alpha$, is

$$k_{\text{in_}i} = m(1-\alpha)\ln\Big(\frac{t}{t_i}\Big) + m\alpha. \tag{7}$$

According to the Barabási–Albert continuum theory,[3,4] the in-degree distribution can be obtained as follows:

$$\begin{aligned} p(k_{\text{in}}) &= \frac{t}{(m_0+t)}\frac{1}{m(1-\alpha)}\mathrm{e}^{-(k_{\text{in}}-m\alpha)/(m(1-\alpha))} \\ &\approx \frac{1}{m(1-\alpha)}\mathrm{e}^{-(k_{\text{in}}-m\alpha)/(m(1-\alpha))}, \ (t\to\infty). \end{aligned} \tag{8}$$

4.2. Out-degree distribution

Summing $\sum_j k_{\text{out_}j}$ over all nodes in the network except the newly introduced one yields

$$\sum_j k_{\text{out_}j} = mt - (1-\alpha)m \approx mt, \quad (t\to\infty). \tag{9}$$

The probability $\prod_{\text{out}}(k_{\text{out_}i})$ is the continuous rate of change of $k_{\text{out_}i}$,

$$\begin{aligned} \frac{\partial k_{\text{out_}i}}{\partial t} &= m_{\text{out_}i}\prod_{\text{out}}(k_{\text{out_}i}) \\ &= m\alpha\frac{k_{\text{out_i}}}{\sum_j k_{\text{out_}j}} = \alpha\frac{k_{\text{out_}i}}{t}. \end{aligned} \tag{10}$$

The solution of this equation, with the initial condition that every node i at its introduction has $k_{\text{out_}i}(t_i) = m(1-\alpha)$, is

$$k_{\text{out_}i}(t) = m(1-\alpha)\Big(\frac{t}{t_i}\Big)^{\alpha}. \tag{11}$$

Finally, the out-degree distribution can be obtained as follows:

$$\begin{aligned} p(k_{\text{out}}) &= \frac{t}{(m_0+t)}\frac{1}{\alpha}(m(1-\alpha))^{1/\alpha}\,k_{\text{out}}^{-(1+1/\alpha)} \\ &\approx \frac{1}{\alpha}(m(1-\alpha))^{1/\alpha}\,k_{\text{out}}^{-(1+1/\alpha)}, \quad (t\to\infty). \end{aligned} \tag{12}$$

4.3. Degree distribution

According to Eqs.(6) and (10), the continuous rate of change of k_i is

$$\begin{aligned} \frac{\partial k_i}{\partial t} &= m_{\text{in_}i}\prod_{\text{in}}(k_{\text{in_}i}) + m_{\text{out_}i}\prod_{\text{out}}(k_{\text{out_}i}) \\ &= m(1-\alpha)\frac{1}{N-1} + m\alpha\frac{k_{\text{out_}i}}{\sum_j k_{\text{out_}j}} \\ &= m(1-\alpha)\frac{1}{t} + m\alpha(1-\alpha)t_i^{-\alpha}t^{\alpha-1} \\ &= \frac{A}{t} + Bt_i^{-\alpha}t^{\alpha-1}, \end{aligned} \tag{13}$$

where $A = m(1-\alpha)$, and $B = m\alpha(1-\alpha)$.

The solution of this equation with the initial condition that every node i at its introduction has $k_i(t_i) = m$, is

$$k_i(t) = \frac{1}{\alpha}\Big(A\ln\Big(\frac{t}{t_i}\Big)^{\alpha} + B\Big(\frac{t}{t_i}\Big)^{\alpha}\Big) + m - A. \tag{14}$$

By Taylor's theorem,

$$k_i(t) = (A\xi^{-1}+B)\frac{1}{\alpha}\Big(\Big(\frac{t}{t_i}\Big)^{\alpha} - 1\Big) + m\,, \tag{15}$$

where $1 \le t_i \le t$ and $1 < \xi < (t/t_i)^{\alpha} \le t^{\alpha}$.

Using Eq.(15), one can write down the probability, with which a node has a degree $k_i(t)$ smaller than k, $P[k_i(t) < k]$ as follows:

$$\begin{aligned} &P[k_i(t) < k] \\ &\quad = P[(A\xi^{-1}+B)\frac{1}{\alpha}\Big(\Big(\frac{t}{t_i}\Big)^{\alpha} - 1\Big) + m < k]. \end{aligned} \tag{16}$$

Assuming that the nodes are added to the network at equal time intervals, the t_i values have a constant probability density $p(t_i) = 1/(m_0+t)$.

Substituting this into Eq.(16), one can obtain

$$\begin{aligned} &P[k_i(t) < k] \\ &\quad = 1 - \frac{t}{m_0+t}\Big(\alpha\frac{k + \frac{(A\xi^{-1}+B)}{\alpha} - m}{A\xi^{-1}+B}\Big)^{-1/\alpha}. \end{aligned} \tag{17}$$

Then the degree distribution $p(k)$ can be obtained

$$\begin{aligned} p(k) &= \frac{\partial P[k_i(t) < k]}{\partial k} \\ &= \frac{t}{m_0+t}\alpha^{-(1+1/\alpha)}(A\xi^{-1}+B)^{1/\alpha} \\ &\quad \times \Big(k + \frac{(A\xi^{-1}+B)}{\alpha} - m\Big)^{-(1+1/\alpha)}, \end{aligned} \tag{18}$$

where $0 < \xi^{-1} < 1$. For a large network, there may be $\xi^{-1} \to 0$ as $t \to \infty$.

Thus the in-degree distribution of the network generated by this simple model behaves exponentially, while its out-degree and degree distributions reveal the power-law behaviour. Although the degree distribution has not an accurate result, one can obtain the value range of its degree distribution for a specified degree k. For JDK1.5.0, the scaling exponent of the out-degree distribution obtained is 2.3, so $\alpha \approx 0.77$.

5. Conclusions

In this paper, network features of some realistic Java software systems provided by Sun and IBM are studied. After the analyses of the degree distributions of these systems, some new features are found in large-scale software systems written in Java. The mean scaling exponent of degree distributions obtained is about 2.5 according to Table 2. In addition, the phenomenon of 'the rich get richer' also exists in large-scale software systems. Myers[21] found that the degree distributions of the systems written in C^{++} also followed the power-law distribution and $\gamma \approx 2.5$, so did the degree distributions of the systems written in Java. Albert and Barabási[4] noted that many actual evolving networks could develop both power-law and exponential degree distributions, which is validated by the above conclusions (in Sections 3 and 4). Therefore the directed preferential-random network model is established to evolve software systems. All the above features can be arbitrarily applied to the large-scale software systems constructed in UML in software engineering. However, realistic software systems are much more complex. There may be some other evolving mechanisms in realistic systems,[24] e.g. the addition of edges and rewiring edges,[10] which are out of the range of discussion in this paper.

References

[1] Watts D J and Strogatz S H 1998 *Nature* **393** 440
[2] Barabási A L and Albert R 1999 *Science* **286** 509
[3] Barabási A L, Albert R and Jeong H 1999 *Physica* A **272** 173
[4] Albert R and Barabási A L 2002 *Rev. Mod. Phys.* **74** 47
[5] Broder A, Kumar R, Maghoul F, Raghavan P, Rajalopagan S, Stata R, Tomkins A and Wiener J 2000 *Comput. Netw.* **33** 309
[6] Albert R, Jeong H and Barabási A L 1999 cond-mat/9907038
[7] Dai S and Guo Y J 2004 *Chin. Phys.* **13** 423
[8] Huang Z X, Wang X R and Zhu H 2004 *Chin. Phys.* **13** 273
[9] Li Y, Shan X M and Ren Y 2004 *Acta Phys. Sin.* **53** 11 (in Chinese)
[10] Albert R and Barabási A L 2000 *Phys. Rev. Lett.* **85** 5234
[11] Barabási A L, Jeong H, Ravasz R, Néda Z, Vicsek T and Schubert A 2002 *Physica* A **311** 590
[12] He Y, Zhang P P, Xu T, Jiang Y M and He D R 2004 *Acta Phys. Sin.* **53** 1710 (in Chinese)
[13] Liljeros F, Edling C R, Amaral L A N, Stanley H E and Aberg Y 2001 *Nature* **411** 907
[14] Jeong H, Tombor B, Albert R, Oltvai Z N and Barabási A L 2000 *Nature* **407** 651
[15] Montoya J M and Solé R V 2000 cond-mat/0011195
[16] Scala A, Amaral L A N and Barthélémy M 2000 *Europhys. Lett.* **55** 594
[17] Chen Z Y and Wang X F 2005 *J. Systems Engineering* **20** 132 (in Chinese)
[18] Wu J and Tan Y J 2005 *J. Systems Engineering* **20** 128 (in Chinese)
[19] OMG *Unified Modeling Language Specification*, http://www.omg.org/technology/documents/formal/uml. htm
[20] Flanagan D 1997 *Java in a Nutshell* 2nd edition (Sebastopol, CA: O'Reilly)
[21] Myers C R 2003 *Phys. Rev.* E **68** 046116
[22] Amaral L A N, Scala A, Barthélémy M and Stanley H E 2000 *PNAS* **97** 11149
[23] Callaway D S M, Newman M E J, Strogatz S H and Wattz D J 2000 *Phys. Rev. Lett.* **85** 5468
[24] Newman M E J and Park J 2003 *Phys. Rev.* E **68** 036122