A Research on Software Rejuvenation Policy Based on DNA Computing

Yi-zhe HE¹, Jia-jun XIONG²,* and Zhan WANG²

¹Jiangsu Xinhai Senior Middle School, Nanjing, 210016, China
²Department of Civil Aviation, Nanjing University of Aeronautics and Astronautics, Nanjing, 210016, China

*Corresponding author

Keyword: Software rejuvenation, Degree of restart dependence, DNA computing.

Abstract: In order to advance the biology intelligence and the efficiency of the software rejuvenation, as well as enlarge the software availability and decrease the cost, the rejuvenation granularity of software should be finer than before and the DNA computing should be applied. According to analysis of the coupling relation between processes of software system, this paper researched the principium of DNA computing and put forward the model of DNA computing which determines the degree of the restart dependence. Finally, this paper establishes the elementary policy for actualizing the software Rejuvenation. So that the intelligent software rejuvenation with fine rejuvenation granularity is supported.

Introduction

Software aging means the performance degradation of software due to such reasons as system memory usage and leakage, unreleased file lock, untimely data updates, fragments of storage space and rounding error during the long-term uninterrupted operation of software[1-3]. Software aging will finally lead to failure of software. To confront software aging, the Software Rejuvenation (SR)[2] policy is proposed. SR means to terminate the program and restart the system to clear its internal status (garbage collection, refreshing kernel tables of the operating system and reinitializing its internal data structure, etc.) so as to release the resources of the operating system and restore the performance of the software when the software performance degradation reaches a certain limit [2, 3]. Software rejuvenation, as a positive response to software aging, is a restarting policy taken before the occurrence of software fault[4]. Many known and unknown software faults can be removed be full or partial restart of software. Both the recursive restartability (RR) technology proposed by UC Berkeley and Stanford University during the joint study on the Recovery Oriented Computing (ROC) [5] and the microrestart technology proposed by Candea [6-8] tend to establish a restart tree in advance, and start restart from the module on the bottom layer of the restart tree, and if the performance cannot be restored, restart of larger scale will be conducted from the parent level. These two technologies follow the software loose coupling principle between modules at the beginning of development so as to prevent the restart of one module from impacting normal operation of other modules. They both feature higher rejuvenation granularity, more imitations and weaker intelligence and adaptability. To establish intelligent fine granularity software rejuvenation policy, we can refer to the idea of documents[7,8] to analysis the coupling relationship among system processes, work out the method to determine the restart dependence and degree of dependence of restart of processes so as to finally set up the restart tree of the system and support the process-level fine-granularity software rejuvenation of the system.

To execute fine-granularity SR, enhance the execution efficient of software rejuvenation and further improve the intelligence and automation of Software Rejuvenation technology, through this article, we have analysed the coupling relationship among processes of software system, defined the degree of coupling among processes, provided the methods to determine the dependence and restart dependence degree of process restart, according to the principles and features of DNA computing, we have preliminarily defined the DNA computing model of software rejuvenation to realize...
accurate and efficient determination of the restart dependence degree of system processes, worked out
the implementation policy of system restart so as to truly realize efficient intelligent software
rejuvenation technology with fine granularity at process level.

Process and Process Coupling

Software includes computer programs, data used by such programs and related documents and
materials. A process is just a program being executed that includes program counter, registers and
current values of variables. Although each process is a separate entity, interaction among processes
often happens, and the closeness of their connections is called the degree of coupling. The closer and
more frequent the connections among the processes, the higher their couple degree will be. According
to the data exchanges among processes, coupling is normally classified into: non-direct coupling, data
coupling, control coupling, public coupling and content coupling.

If the data along processes are exchanged via the parameter tables of interfaces, it is called data
coupling. If an upper process calls a lower process to complete designated functions and the call is
realized via the delivery of parameters, it is called control coupling. If a public environment is used
for data exchange among processes, it is called public coupling. Such public environment can be
global variables, global data structure, communication buffers and data(base) files, etc. If a process
directly enters another process for data access or service usage or there is a two-way calling
relationship between both, it is called content coupling. The difficulties of the four types of coupling
above are in ascending order, with content coupling having the poorest adaptability. If the
connections among processes are implemented only by control and calling by the upper processes,
this case is regarded as non-direct coupling. For the purpose of this article, processes with data,
control, public and content couplings occurring by direct interaction are called direct coupling
processes.

Determination of Restart Dependence and Restart Dependence Degree of Processes

The restart of a process may cause faults or errors of other processes, which is called the restart
dependence among processes, which is dependent on their degree of coupling. Restart dependence is
classified into four categories in response to the types of couplings among processes: mutually
independent, function-dependent, status-dependent and mutually function-dependent, whose
definitions are provided below:

**Definition 1:** If process \( A \) calls process \( B \), \( A \) is function-dependent on \( B \), represented as \( A \rightarrow B \).

**Property 1:** Transitivity of function dependence: If \( A \rightarrow B \), \( B \rightarrow C \), then \( A \rightarrow C \).

**Theorem 1:** If process \( A \) is function-dependent on \( B \), then \( B \) will be restarted when \( A \) is restarted,
when \( A \) will not necessarily be restarted when \( B \) is restarted.

**Definition 2:** If \( A \rightarrow B \) and \( B \rightarrow A \), then \( A \) and \( B \) are mutually function-dependent, represented as \( A \leftrightarrow B \).

**Property 2:** Mutual function dependence features exchangeability and deliverability. If \( A \leftrightarrow B \),
then \( B \leftrightarrow A \); If \( A \leftrightarrow B \), \( B \rightarrow C \), then \( A \leftrightarrow C \).

**Theorem 2:** If process \( A \) and \( B \) are mutually function-dependent, then when one process is
restarted, the other will also be restarted, namely, \( A \) and \( B \) are always restarted at the same time.

**Definition 3:** If process \( A \) shares data or status with \( B \), then \( A \) and \( B \) are status-dependent, represented as \( A \wedge B \).

**Property 3:** Status dependence features exchangeability and deliverability. If \( A \wedge B \), then \( B \wedge A \); If
\( A \wedge B \), \( B \wedge C \), then \( A \wedge C \).

**Theorem 3:** If process \( A \) is status-dependent on \( B \), then when one process needs to be restarted,
the other process will also be restarted, namely, \( A \) and \( B \) are always restarted at the same time.

**Definition 4:** If process \( A \) and \( B \) are neither function-dependent nor status-dependent, then \( A \) and
\( B \) are mutually independent, represented as \( A \square B \).

**Property 4:** Exchangeability of mutual independence: If \( A \square B \), then \( B \square A \).

In case of data coupling and non-direct coupling, corresponding processes are regarded as
mutually independent. Mutual independence does not have deliverability.

When two processes are not directly connected, it is required to consider the restart dependence of all processes passed during the period to determine their restart dependence.

Definition 5: The degree of dependence of the restart of a process \( A \) on that of another process \( B \) is called the restart dependence degree \( D[A \cdot B] \), whose value range is \( \{0, 1, -1, 2, 3, 4\} \). More specifically, \( D[A \cdot B] = 0 \), \( D[A \rightarrow B] = 1 \), \( D[A \land B] = 2 \) and \( D[A \leftrightarrow B] = 3 \). Besides, Definition \( D[A \cdot A] \) is the restart dependence degree of process \( A \) itself, and \( D[A \cdot A] = 4 \). In particular, if \( A \rightarrow B \), then the restart dependence degree of \( B \) and \( A \) is \( D[B \cdot A] = -1 \).

Thus we can get that \( D[A \cdot B] = 0/1/2/3 \Leftrightarrow A \) is independent from \( B / A \) and \( B \) are function-dependent/\( A \) and \( B \) are status-dependent/\( A \) and \( B \) are mutually function-dependent.

Convention 1: When there are multiple restart dependences between two processes, we take the one with the highest restart dependence degree. When there are multiple couplings between two processes, we take the one with the highest coupling degree.

Definition 6: If processes \( A \) and \( B \) are function-dependent, we consider that \( A \) is reachable to \( B \) while \( B \) is unreachable to \( A \); If processes \( A \) and \( B \) are status-dependent or mutually function-dependent, we consider that \( A \) and \( B \) are mutually reachable; and If process \( A \) and \( B \) are mutually independent, we consider that \( A \) and \( B \) are mutually unreachable, and vice versa.

Definition 7: If process \( A \) and \( K \) are not directly connected, while \( A \) can reach \( K \) through several processes, we consider that there is an reachable path \( R[A \sim K] \) from \( A \) to \( K \) with processes in-between connected with “-“.

Convention 2: If there is an reachable path \( R[A \sim K] = A - K_1 - K_2 - \ldots - K_n - K \) from process \( A \) to process \( K \), then the restart dependence degree of \( A \) and \( K \) should be \( D[A \cdot K] = \text{Min} [D[A \cdot K_1], D[K_n \cdot K]] \).

Deduction 1: If process there is an reachable path \( R[A \sim K] \) between \( A \) and \( K \) but no such reachable path as \( R[A \sim K] \), then \( A \) is function-dependent on \( K \).

Deduction 2: If there is neither an reachable path \( R[A \sim K] \) nor an reachable path \( R[A \sim K] \) between process \( A \) and process \( K \), then \( A \) and \( K \) are mutually independent.

Deduction 3: If there are both an reachable path \( R[A \sim K] \) and an reachable path \( R[A \sim K] \) between process \( A \) and \( K \), then \( A \) and \( K \) are either status-dependent or mutually function-dependent.

Detailed demonstrations for theorems 1-3 and deductions 1-3 are given in document [9].

**Application of DNA Computing in Software Restart**

The fundamental idea of DNA computing is to substitute digital switching components with the information processing ability of organic molecules in computers. The applicability of DNA computing in determining the process restart dependence degree to build system restart tree can be determined by analysing the determination methods for restart dependence and restart dependence degree of processes in software rejuvenation in combination with the principles of DNA computing.

**Determining Restart Dependence Degree of Processes with DNA Computing**

Suppose in an interaction diagram of direct coupling processes, each edge is represented by, where, is the initial process, is the terminal process and is the weight of the directed edge; to facilitate determining the restart dependence degree between any processes with deductions 1, 2 and 3, we convert the interaction diagram of direct coupling processes as follows:

\[
\begin{align*}
(s, d, 0) \quad \rightarrow \quad & (s, d, 0) \\
(s, d, 1) \quad \rightarrow \quad & (s, d, 1) \\
(s, d, -1) \quad \rightarrow \quad & (s, d, 0) \\
(s, d, 2) \quad \rightarrow \quad & (s, d, 1)
\end{align*}
\]
Where, each vector with three factors represents a directed edge in the diagram, with the first and second factors respectively as the initial vertex and terminal vertex of the directed edge, and the third as the weight of the directed edge. Thus we can get the directed graph \( G = (V, l) \), where, \( V = \{ V_i | V_j \} \), representing the limited vertex set of processes in the figure, \( E = \{ | l_i = (V_i, V_j), W_i > 0, V_i, V_j \} \), the number of processes in the figure is \( n \). Set \( s \) and \( d \) as process numbers, and the solution to establish DNA computing is based on the following non-deterministic algorithm:

Input: Directed graph \( G \) of a direct coupling process with \( n \) vertexes.

Step 1: Generate random paths between any vertexes in the graph.

Step 2: Keep only the shortest path between any vertexes \( V_i \) and \( V_j \), and discard the other paths.

Step 3: If there is still any path kept between the two vertexes, we get HP, otherwise, there’s no HP.

Therefore, in the graph, each vertex \( V \) can be represented by a DNA string \( s_i \) with the random length of \( 20^i \) DNA, \( 0 \leq i \leq n \), cases of \( i = 1,2,3 \), should be represented with the following oligonucleotide fragments with the length of 20:

\[
\begin{align*}
  s_1 &= \text{GCTATTCGAGCTTAAAGCTA} \\
  s_2 &= \text{TATCGGATCGGTATATCCGA} \\
  s_3 &= \text{GGCTAGGTACCAGCATGCTT}
\end{align*}
\]

Define Watson-Crick morphism function \( h \) represents a mapping of each base in the DNA to corresponding complementary base Watson-Crick:

\[
\begin{align*}
  h(A) &= T, h(T) &= A, h(c) &= G, h(G) &= C;
\end{align*}
\]

thus we can get:

\[
\begin{align*}
  h(s_1) &= \text{CGATAAGCTCGAATTTCGAT} \\
  h(s_2) &= \text{ATAGCCTAGCCATATAGGCT} \\
  h(s_3) &= \text{CCGATCCATGGTCGTACGAA}
\end{align*}
\]

Where, (1) when there is a directed edge with the weight of 1 between \( V_i \) and \( V_j \) pointing from \( V_i \) to \( V_j \), then \( s_i \) and \( s_j \) are respectively divided into two substrings with the unit length of 10, \( s_i = s_i^0 s_i^1 \), \( s_j = s_j^0 s_j^1 \), the corresponding directed edges can be encoded with base sequenced with the available length of 20: composed of the second half of the oligonucleotide fragment corresponding to its initial vertex and the Watson-Crick complementary base of first half of the oligonucleotide fragments corresponding to the terminal vertex, which is:

\[
\begin{align*}
  e_{i,s_1} &= \text{CATATAGGCTCGAATAGCTC} \\
  e_{3+s_2} &= \text{GAATTTCGATATAGCCTAGC}
\end{align*}
\]

(2) When there are 2 directed edges, each with the weight of 3, between \( V_i \) and \( V_j \), and \( V_i \) and \( V_j \) are exchangeable initial vertex and terminal vertex, then we divide \( s_i \) and \( s_j \) each into two substrings, \( s_i = s_i^0 s_i^1, s_j = s_j^0 s_j^1 \); \( s_i^0 \) is 13 long, \( s_i^1 \) is 7 long, \( s_j^0 \) is 7 long and \( s_j^1 \) is 13 long; then directed edges between \( V_i \) and \( V_j \) can be encoded with base sequenced with the available length of 14: composed of the second half of the oligonucleotide fragment corresponding to its initial vertex and the Watson-Crick complementary base of first half of the oligonucleotide fragments corresponding to the terminal vertex, then:

\[
\begin{align*}
  e_{i,s_1} &= \text{CATATAGGCTCGAATAGCTC} \\
  e_{3+s_2} &= \text{GAATTTCGATATAGCCTAGC}
\end{align*}
\]
For each vertex $V_i$ and each directed edge $V_i \rightarrow V_j$ in the graph, in a single coupled reaction, a number of oligonucleotide fragments $s_i$ and $e_{i\rightarrow j}$ are mixed together. Here, oligonucleotide fragment $s_i$ is used as a template to generate the oligonucleotides of related edges connected in the coupled reaction. Finally, various DNA molecules will be generated through coupled reaction which can be regarded as codes corresponding to random paths in the directed graph. Therefore, we can conclude:

1. If there are two paths between vertexes $V_i$ and $V_j$ to allow both of them to reach each other, then:
   a. where the two paths are with the same length and the number of bases in the paths are multiples of 14 instead of 20, then processes $i$ and $j$ are mutually function-dependent; when the number of bases in the paths are multiples of both 14 and 20, we check the vertexes in the paths as well as the length of each edge, if the length is 14, processes $i$ and $j$ are mutually function-dependent, otherwise, they are status-dependent;
   b. where the two paths are not with the same length, processes $i$ and $j$ are status-dependent;
2. If there is the shortest path between vertexes $V_i$ and $V_j$ from $V_i$ to $V_j$ and not such shortest path from $V_j$ to $V_i$, the processes $i$ and $j$ are function-dependent.

Now we will briefly describe the operation of such mathematical model with an example. Fig. 1 is the flowchart of the UML cash drawing cases from a bank’s ATM system. In the figure, Object a: Bankcard Reader, Object b: Customer Panel, Object c: Cash Dispenser, Object d: Cash Container, Object e: Receipt Printer, Object f: Card Transaction, Object g: Cash Withdrawal and Object h: Bank System, Actor A: Bank customer, Actor B: Bank System.

![Figure 1. Sequence for objects of cash drawing cases](image)

From the data model of DNA computing we can get the restart dependence degree between any processes in the figure as shown in Table 1, and finally we can get the reachable set of each process as shown below:

$S[a] = \{a, b, c, d, e, f, g, h\}$; $S[b] = \{b, c, d, e, f, g, h\}$;
$S[c] = \{c, d\}$; $S[d] = \{d\}$;
$S[e] = \{c, d, e, g, h\}$; $S[f] = \{b, c, d, e, f, g, h\}$;
$S[g] = \{c, d, e, g, h\}$; $S[h] = \{c, d, e, g, h\}$;

Then we suppose that through analysis on data from long-term monitoring of the system or...
experience that the leakage of memory and CPU occupancy are main factors to cause system performance degradation, then we order the impacts of each processes on physical memory as $c,e,g,f$, then a corresponding branch of the restart tree is as shown in Fig. 2; if we order the impacts of each processes on CPU wastage as $d,b,h,a$, we can get another branch of the restart tree as shown in Fig. 3; and thus we can finally get the system restart tree as shown in Fig. 4.

Table 1. Degree of restart dependence between any processes

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>-1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>-1</td>
<td>-1</td>
<td>4</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>d</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>4</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>e</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>-1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>f</td>
<td>-1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>g</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>-1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>h</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>-1</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

![Figure 2.restart tree(a)](image1)

![Figure 3.restart tree(b)](image2)

![Figure 4.restart tree(c)](image3)

**Implementation of Restart**

If the leakage of memory is the major factor of software degradation, and CPU wastage is far lower, then according to case 1, we can get the restart chain $S[c] \# S[e] \# S[f] \# U$; if the leakage of memory is the major factor of software degradation, CPU wastage is a minor factor but must be considers, then we can get the restart chain according to case 2: $S[d] \# S[b] \# S[a] \# U$. For case 3, for instance, when the software start running, the leakage of memory is the major factor of software degradation, then $S[c]$ is the head of the restart chain, after a period of time, we detect that CPU wastage has become the major factor, then we should restart another branch, therefore, $S[c] \# S[d]$. Note that even if $S[d] \subseteq S[c]$ we should not ignore such restart group because CPU wastage is not necessarily caused by the incapability of restarting $S[c]$ to release sufficient CPU, but possibly caused by the current running environment. If it is confirmed that restarting $S[d]$ fail to release sufficient CPU, we can then restart the next leaf on this branch. Following the same way, we can determine the restart group and get the restart chain according to the actual monitored values during the runtime.

**Conclusions**

This paper analyzes the characteristics and principles of DNA computing, constructed preliminary mathematical model for process restart of software rejuvenation based on DNA computing along with its functions and working principles so as to rapidly and efficiently determine the restart dependence degree of any processes in a system, proposed preliminary implementation policy for system restart, and finally laid a solid foundation for executing software rejuvenation policy with fine granularity at process level for truly realizing intelligent and automatic software rejuvenation, greatly improving the implementation efficiency of rejuvenation technology, minimizing the rejuvenation cost, thus
enhancing the applicability and flexibility of rejuvenation technology and improving the reliability of software.

Acknowledgement

This research was financially supported by the Project of the National Natural Science Foundation of China (61671237).

References


