一个多节点声纳系统中
同步时钟机制的可靠性评估和系统优化问题

组号 6　姓名 王山木　学号 519021910418

摘　要：工程系统的设计往往需要考虑使用寿命、可靠性及成本等多方面的因素。本文以一个多节点的声纳系统为例，运用了蒙特卡罗模拟方法研究其同步时钟机制的可靠性及系统寿命并通过大规模的试验得出该系统中工作节点数目在不同优化目标下的最优取值。试验结果表明，随着工作节点数目的增加，系统的可靠性和使用寿命均呈现先上升后缓慢下降的趋势。此外，本文还对系统的可用性进行了理论推导，并与仿真得到的可靠性进行了对比。

关键词：Matlab，蒙特卡罗方法，系统可靠性，系统优化

The Reliability Evaluation and Optimization Problem of A Multi-node Sonar System's Synchronous Clock Mechanism

ABSTRACT: The design of an engineering system often needs to take many factors into consideration, such as service life, reliability and cost. Taking a multi-node sonar system as an example, this paper uses the Monte Carlo simulation method to study the reliability of the synchronization clock mechanism and the system life. Through large-scale experiments, it is concluded that the number of working nodes in the system is under different optimization goals. The simulation results show that with the increase in the number of working nodes, the reliability and the system life both show a trend of increasing first and then decreasing slowly. In addition, this paper also derives the availability of the system theoretically and compares it with the reliability obtained by simulation.

Key words: Matlab, Monte Carlo method, System Reliability, System Optimization

1 引言

对大规模的工程系统而言，系统的可靠性和使用寿命是很重要的两个指标。因此，在进行大规模工程系统设计时，如何在控制成本的前提下进一步提高其可靠性和延长使用寿命至关重要，并且在系统投入使用后能够带来巨大的效益。为了进一步提高可靠性及延长使用寿命，许多大型系统会采用冗余的结构设计从而让系统在一些元件损坏的情况下仍能正常工作，提高系统的容错率。

本文以一个多节点声纳系统为例，运用蒙特卡罗方法以及 Matlab 仿真研究其同步时钟机制的可靠性和系统优化问题，并给出相应的优化结果及理论分析。

2 符号及变量说明

本文用到的主要符号及其含义如表 2-1 所示：

表2-1 本文的主要符号及其含义[1]

符号	含义
$\Pr(X)$	事件 X 的发生概率
$1(X)$	事件 X 为真时取 1，否则取 0
$f_{T_X}(\tau)$	元件 X 使用寿命的概率密度函数
$1/\lambda_X$	元件 X 的期望寿命
P_{EX}	元件发生故障，故障 X 发生的条件概率
$G_{Ai}(t)$，$G_{Bi}(t)$	在时刻 t 节点 i 中切换器 A 和 B 的性能状态
$G_{Ni}(t)$	在时刻 t 节点 i 的性能状态
$G_{sys}(t)$	在 t 时刻声纳系统整体的性能状态

本文用到的主要变量取值如表 2-2 所示：

表2-2 本文的主要变量及其取值[1]

符号	含义	取值
$sample$	实验的样本数量	1.0×10^5
n	节点的个数	5~20 内变化
w	需要判断的时间	2.5×10^4hour

3 多节点声纳系统中同步时钟的模型

3.1 同步时钟机制的物理模型[1]

本文研究的多节点声纳系统中的同步时钟机制的物理模型如附图 1 所示。该声纳系统包含 n 个独立的物理同构的节点，只要有 5 个节点能在同步时钟下正常工作，系统整体就能正常运转。所有节点经由时钟信号总线连接，由其中一个节点担当主节点，其时钟电路工作于主模式。切换器 A 的掷刀接合于触点 2，切换器 B 接通。本地时钟源信号既被用作本节点内部其他电路的工作时钟，又被输出到时钟信号总线上；其余节点均担当从节点，内部电路中切换器 A 的掷刀接合于触点 1，切换器 B 处于断开状态。从节点不向总线输出信号，仅从总线获取时钟信号作为本节点内部其他电路的工作时钟。

切换器可能发生不可修复的故障从而导致节点发生故障。若某节点处于主模式，当它内部的时钟信号检测电路发现总线时钟信号失效，或不同于本地时钟源信号，则会通知控制电路改变切换器 A 和 B 的连接，退出主模式转为从模式。若某节点处于从模式，当它内部的时钟信号检测电路发现总线时钟信号失效，则会通知控制电路进入戒备状态，最终将随机产生一个从节点转入主模式。

3.2 同步时钟机制的数学模型

3.2.1 切换器的数学模型

切换器 A 共有 4 种工作状态，依次记为 A0、A1、A2、A3，其中 A0 为正常工作状态，三种故障状态依次对应为：掷刀无法与触点 1 脱离、掷刀无法与触点 2 脱离以及掷刀保持悬空；切换器 B 共有 3 种工作状态，依次记为 B0、B1、B2，其中 B0 为正常工作状态，两种故障状态依次对应为：掷刀无法与触点 1 脱离、掷刀保持悬空。切换器 A、B 的使用寿命

均服从负指数分布，切换器 A、B 的使用寿命T_A、T_B的概率密度函数为：

$$\begin{cases} f_{T_A}(t) = \lambda_A e^{-\lambda_A t}, \text{其中 } \lambda_A = \frac{1}{3.70\times10^4 hour} \\ f_{T_B}(t) = \lambda_B e^{-\lambda_B t}, \text{其中 } \lambda_B = \frac{1}{4.80\times10^5 hour} \end{cases} \quad (1)$$

在已知切换器发生故障的前提下，切换器处于各状态的概率如下：

$$\begin{cases} P_{EA1} = Pr(A1 \mid A\ is\ failed) = 0.26 \\ P_{EA2} = Pr(A2 \mid A\ is\ failed) = 0.26 \\ P_{EA3} = Pr(A3 \mid A\ is\ failed) = 0.48 \\ P_{EB1} = Pr(B1 \mid B\ is\ failed) = 0.35 \\ P_{EB2} = Pr(B2 \mid B\ is\ failed) = 0.65 \end{cases} \quad (2)$$

根据贝叶斯公式，可以计算出在 t 时刻切换器处于各状态的概率如下：

$$\begin{cases} P_{Ai} = P_{EAi} \cdot Pr(T_A < t) = P_{EAi} \cdot (1 - e^{-\lambda_A t}), & i \in \{1,2,3\} \\ P_{Bi} = P_{EBi} \cdot Pr(T_B < t) = P_{EBi} \cdot (1 - e^{-\lambda_B t}), & i \in \{1,2\} \end{cases} \quad (3)$$

在本研究中，为进一步简化模型，本文对切换器做出如下假设：

i） 不同元件是否发生故障在统计上相互独立

ii） 元件一旦发生故障，故障类型即刻确定，且其后不会发生变化

iii） 故障均不可修复

3.2.2 节点的状态模型

通过 3.1 的讨论可以发现，节点的状态完全由其内部两个切换器的状态组成，不同切换器的状态组合共有 12 种，对应了 6 种不同的节点状态，其对应关系详见附录中表 1。将六种节点状态记为$g_{Ni},\ i \in \{0,1,2,3,4,5\}$, 则在时刻 t，任一节点的性能状态的概率分布可表示为:

$$p_{Nk}(t) = \sum_{(i,j)\in\{(g_{Ai},g_{Bi})\to g_{Nk}\}} P_{Ai}(t) \cdot P_{Bj}(t) \text{ 其中} k = 0,1,\dots,5\ \ i = 0,1,2,3\ \ j = 0,1,2 \quad (4)$$

3.2.3 系统的状态模型

系统的状态也可以由各个节点的状态推导出来，用$Q_{Ni}(t)$来表示 t 时刻处于g_{Ni}状态的节点个数，即定义：

$$Q_{Ni}(t) = \sum_{j=0}^{n} 1\big(G_{Nj}(t) = g_{Ni}\big),\ i \in \{0,1,2,3,4,5\} \quad (5)$$

根据系统中所有节点的工作状态，可将系统的状态归为 4 类，分别记为G_{sys1}，G_{sys2}，G_{sys3}，G_{sys4}，其中G_{sys1}表示系统确定不能正常工作，G_{sys2}表示系统确定能正常工作，G_{sys3}和G_{sys4}分别表示相同节点状态下系统恰好能正常工作和恰好不能正常工作的状态。

根据理论分析[1]，系统各状态与Q_{Ni}间的逻辑关系为：

i） 当系统满足以下四个条件中任何一个，则系统处于G_{sys1}：

$$\begin{cases} Q_{N5}(t) > 0 \\ Q_{N3}(t) > 1 \\ Q_{N0}(t) + Q_{N2}(t) + Q_{N3}(t) = 0 \\ Q_{N0}(t) + Q_{N1}(t) + 1((Q_{N2}(t) + Q_{N3}(t)) > 0) < \mathrm{k} \end{cases} \quad (6)$$

ii） 若系统满足 $Q_{N5}(t) = 0$，并且以下三个条件任何一个，则系统处于 G_{sys2}：

$$\begin{cases} (Q_{N3}(t) = 1) \wedge (Q_{N0}(t) + Q_{N1}(t) \geq k - 1) \\ (Q_{N3}(t) = 0) \wedge (Q_{N0}(t) \geq 1) \wedge (Q_{N1}(t) \geq k - 1) \\ (Q_{N0}(t) = Q_{N3}(t) = 0) \wedge (Q_{N1}(t) \geq k - 1) \wedge (Q_{N2}(t) \geq 1) \end{cases} \quad (7)$$

iii） 若系统满足以下两个条件，则系统或者处于 G_{sys3}或者 G_{sys4}：

$$\begin{cases} Q_{N3}(t)+Q_{N5}(t)=0 \\ (Q_{N0}(t)\geq 1)\wedge(Q_{N0}(t)+Q_{N1}(t)=k-1)\wedge(Q_{N2}(t)\geq 1) \end{cases} \quad (8)$$

其中处于 G_{sys3} 的概率为 $p=\frac{Q_{N2}(t)}{Q_{N0}(t)+Q_{N2}(t)}$，处于$G_{sys4}$ 的概率为 $1-p=\frac{Q_{N0}(t)}{Q_{N0}(t)+Q_{N2}(t)}$.

3.3 系统可靠性指标

1. 系统工作寿命T_f（Time to Failure）：又称工作寿命，指系统从初始时间到首次发生失效的时间，即系统首次向G_{sys1}或G_{sys4}发生转移的时间。由于模型存在系统生命无限大的漏洞[1]，案例中我们限定最大值为$9.0\times10^4\ hour$。

2. 平均首次失效时间(Mean Time to Failure)：指平均工作寿命，即T_f的统计平均$\mathrm{E}(T_f)$。

3. 系统可靠性：系统工作寿命超过某一定值w的概率，即系统在$0<\mathrm{t}<\mathrm{w}$期间一直正常工作的概率：$R(w)=\Pr(T_f\geq w)$。接下来的研究中，我们取w为$2.5\times10^4\ hour$。

4 蒙特卡罗模拟方法及其 Matlab 实现

4.1 蒙特卡罗模拟方法简介[2]

蒙特卡罗（Monte Carlo）方法，又称随机抽样或统计试验方法，属于计算数学的一个分支，它是在上世纪四十年代中期为了适应当时原子能事业的发展而发展起来的。传统的经验方法由于不能逼近真实的物理过程，很难得到满意的结果，而蒙特卡罗方法由于能够真实地模拟实际物理过程，故解决问题与实际非常符合，可以得到很圆满的结果。这也是以概率和统计理论方法为基础的一种计算方法，是使用随机数（或更常见的伪随机数）来解决很多计算问题的方法。将所求解的问题同一定的概率模型相联系，用电子计算机实现统计模拟或抽样，以获得问题的近似解。为象征性地表明这一方法的概率统计特征，故借用赌城蒙特卡罗命名。蒙特卡罗方法通过抓住事物运动的几何数量和几何特征，利用数学方法来加以模拟，即进行一种数字模拟实验。它是以一个概率模型为基础，按照这个模型所描绘的过程，通过模拟实验的结果，作为问题的近似解。

蒙特卡罗模拟方法以概率论中的大数定理作为其数学理论基础：从大量随机试验中得到的样本均值应当与待求量的期望非常接近，并且随着样本容量趋向于无穷大，样本均值和期望的残差将趋向于零。

4.2 蒙特卡罗模拟方法在本案例中的 Matlab 实现

4.2.1 切换器的寿命演变的仿真实现

1）仿真思路：切换器寿命的演变主要有两种仿真思路，思路一是将“时间”按照固定步长推进，每达到一个新的时间点，根据相关概率推断切换器是否会发生故障以及发生故障后转化为哪一种状态，其中选取的固定步长也称为仿真颗粒度。显然，时间离散间隔越小，仿真越精细，但同时运算量也越大，运行速率越低。思路二则是将“时间”按变化步长推进，即我们首先按照一定的规则随机生成切换器 A 和切换器 B 发生故障的时间，认为其在这个时间点发生了故障，之后再按照概率随机生成故障后转化的状态，由于变步长的方式只需检验在生成的时间点系统是否故障，故其仿真效率更高。在本案例中，由于我们需要对至少1.0×10^5套系统进行模拟，同时每一套系统具有 5 至 20 个节点，每个节点各具有两个切换器，如果采用固定步长的方式进行仿真，其运算量会相当巨大，运算时间会相当长；另一方面，当仿真次数足够多时，变化步长的方式也能取得精度较高的仿真结果。因此，在本案例

的研究中，我们采取变步长的方式进行仿真。

2）Matlab 实现：Matlab 中提供了 exprnd 函数，用以生成服从指数分布的随机数，本案例中，可以先用 exprnd(mu_A, samples, n)按参数生成一个$sample \times n$ 的矩阵 t_A，其中下标为(i, j)的元素值表示 “第 i 个系统中第 j 个节点的 A 切换器发生故障的时间”。同理可生成矩阵 t_B。

随后我们可以利用函数 rand()生成一个$n_{sample} \times n$ 的矩阵 state_A，并按照 3.2.1 一节中的式(2)的概率将其中的元素分别设为 1、2、3，其中下标为(i, j)的元素值表示 “第 i 个系统中第 j 个 A 切换器发生故障后切换器的状态”。同理可生成矩阵 state_B。

一个系统内某节点内部切换器每发生一次变化需要同时记录下 “节点的序号、切换器的类型、故障发生时间、切换器转换后的状态” ，因此我们用上述矩阵生成一个新的矩阵 change_AB，具体实现过程见附录中 variable.m 文件。记矩阵 change_AB 中任意一行的数据依次为 x，y，z，w，则每一行表示的信息如下：

$$\begin{cases} \text{当 y} = 0 \text{ 时，第 x 个节点的 A 转换器在 z 时刻变为了状态 w} \\ \text{当 y} = 1 \text{ 时，第 x 个节点的 B 转换器在 z 时刻变为了状态 w} \end{cases}$$

再利用 sortrows 函数，将矩阵 change_AB 按照第三列（即按照时间先后顺序）排序得到新的矩阵，记为 change_sort。

4.2.2 节点的状态变化的仿真实现

附录的表格 1 中列举了切换器 A 和 B 的所有组合对应的节点状态，因此我们只需要知道每发生一次变化后节点内部切换器 A 和切换器 B 的状态即可推知该节点在这一时刻至下一次变化发生前的状态如何。因此，我们需要两个矩阵来记录下各个节点中切换器 A 和切换器 B 的状态，并在每一次变化发生后进行更新。同时，我们需要一个矩阵存储各个节点的状态，由于初始时节点都正常工作，即工作于状态 0，可用 zeros(samples, n)生成一个$sample \times n$的全零矩阵 state_nodes。

由于初始时，所有节点内部的切换器都是正常工作，同样可以利用 zeros(samples, n)，生成一个$sample \times n$的全零矩阵 A，用于记录 samples 个系统中所有节点内部切换器 A 的初始状态。同理可生成矩阵 B。此后，我们依次读取变化信息，即矩阵 change_sort 的每一行，并根据第二列和第四列的数据对 A/B 矩阵进行更新。

在每次更新后，我们需要重新判断该节点的状态，由 3.2.2 我们得知节点共有 6 种工作状态，可用 0，1，2，3，4，5 依次对应。而当切换器 A 和切换器 B 取不同值时，各对应一种节点工作状态。因此我们可以根据附表 1，生成如下的节点状态矩阵：

$$\text{state\_matrix} = \begin{bmatrix} 0 & 3 & 1 \\ 1 & 5 & 1 \\ 2 & 3 & 4 \\ 4 & 4 & 4 \end{bmatrix}$$

若矩阵的行、列均从零开始计数，即左上角的第一个元素对应下标为（0，0），则该矩阵的行数对应了切换器 A 的状态，列数对应了切换器 B 的状态，而矩阵的元素对应了节点的工作状态。如此，我们只需在每次变化发生后，更新变化节点的切换器的状态，再以更新后的状态做下标，便可读取到变化发生后节点的工作状态。

4.2.3 系统的状态变化的仿真实现

由 3.2.3 的讨论可知，系统可分为 4 种运行状态，其中G_{sys1}及G_{sys4}无法正常工作。需要判断系统能否正常工作，需要先获得系统中各种类型的节点的个数，再计算附表 2 给出的 C1~C9共 9 个逻辑表达式的取值。根据 3.2.3 的讨论，判断系统工作状态的伪代码如下：

Algorithm 1: Determine the state of a system

Input: Number of nodes in each state：$(Q_{PF}, Q_{SO}, Q_{DM}, Q_{MO}, Q_{DN}, Q_{FB})$

Output: The current state of the system: $\begin{cases} state = 0: work\ properly \\ state = 1: cannot\ work \end{cases}$

1. Calculation logic expression C1 to C9 ;
2. **If** (C1 || C2 || C3 || C4) **then**
3. state = 0;
4. **Else if** (C5 && (C6 || C7)) **then**
5. state = 1;
6. **Else if** (C8 && C9) **then**
7. Randomly generate a number between 0 and 1: q
8. **If** ($q \le \frac{Q_{DM}}{Q_{DM}+Q_{PF}}$) **then**
9. state = 1;
10. **Else**
11. state = 0;
12. **End**

4.3 系统可靠性指标的获取

1）本案例中，我们取每次仿真的系统的个数为$\text{samples} = 1.0 \times 10^5$，分别在节点个数为5~20时对各个系统进行遍历，按照 4.2 中的思路得到系统由正常工作转为无法工作时的时间，作为该系统的工作寿命T_f。再取这些样本的平均寿命:

$$\mathrm{E}(T_f) = \left(\sum_{i=1}^{1.0\times10^5} T_{fi} \right) \Big/ (1.0 \times 10^5)$$

即可得系统的平均寿命$\mathrm{E}(T_f)$的仿真结果。

2）同样的，可以计算出系统寿命大于w的频率 $f(T \ge w)$，根据大数定理即可得到系统在 w 时刻可靠性的估计值 $R(w) = P(T \ge w) \approx f(T \ge w)$

5 Matlab 仿真结果

5.1 系统的平均寿命仿真结果

按照第 4 节所述思路进行 Matlab 编程及仿真实验，得到不同节点个数情况下系统的平均寿命如表 5-1 所示：

表 5-1 不同节点个数对应的系统平均寿命

节点数目	5	6	7	8	9	10	11	12
系统平均寿命（h）	11870	22770	32800	41810	49650	56180	61430	65500
节点数目	13	14	15	16	17	18	19	20
系统平均寿命（h）	68770	71170	72810	73740	74450	74820	75030	74910

5.2 系统可靠性仿真结果

同理可通过 Matlab 得到不同节点个数情况下系统的可靠性，结果如表 5-2 所示：

表 5-2 不同节点个数对应的系统可靠性

节点数目	5	6	7	8	9	10	11	12
系统可靠性（%）	11.85	32.67	53.95	70.92	82.19	88.81	92.24	94.18
节点数目	13	14	15	16	17	18	19	20
系统可靠性（%）	95.06	95.16	94.98	94.65	94.36	94.00	93.64	93.10

5.3 数据分析

根据表 5-1 及表 5-2 的数据，可知：

1）当系统的节点数目为 19 时，系统的平均寿命最长，为 75065.7 小时。

2）当系统的节点数目为 14 时，系统的可靠性最高，为 95.16%。

更进一步，对数据进行可视化，得到系统节点个数与系统平均寿命和可靠性的关系如图 5.1 所示：

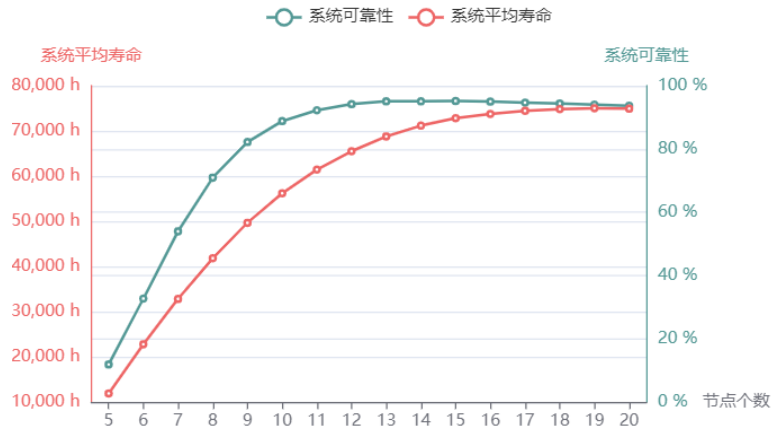

图 5.1 系统节点个数与系统平均寿命和可靠性的关系

可以看出可靠性和平均寿命随着系统中节点数目的增加，都呈现先快速上升，随后上升减缓、最后缓慢下降的变化趋势，说明在本案例研究的系统中，节点个数并非越多越好，节点数目过多将增加系统在较短时间内发生总线阻塞现象的概率，从而削弱系统的可靠性、减短系统的寿命。

6 拓展部分：系统的可用性的理论分析及系统模型优化

6.1 系统可用性的理论推导

根据 3.2.1 中的讨论可以得知，切换器 A 和切换器 B 在 t 时刻处于各种故障状态的概率如式（3）所示，由[1]中分析可知，节点所处的六种状态所对应的概率，并可进一步推至系统在 t 时刻处于各状态的概率，由于我们只需要计算$\Pr(G_{sys2}\mathrm{V}(valid \mid G_{sys3}))$即可得系统可用性，故我们需要计算事件 C5~C9[1]的概率。

针对不同节点数目的系统，我们穷举所有的可能的节点状态组合，并判断每一种组合对应的系统状态、计算每一种组合出现的概率，将使得系统有效的组合出现的概率进行累加，即可得系统在$\mathrm{t}=\mathrm{w}$时的可用性$\mathrm{A(w)}$。该部分可借助 Matlab 进行运算，其伪代码如下：

Algorithm 2: Evaluate the availiability of the system

Input: Time w, the number of nodes n

Output: System availability at this time: $A(w,n)$

1. Exhaustive list of all combinations;
2. **For** all combinations
3. Calculation logic expression C5 to C9 ;
4. **If** (C1 || C2 || C3 || C4) **then**
5. continue;
6. **Else if** (C5 && (C6 || C7)) **then**
7. Calculate the probability p of the current combination;
8. $A(w,n) = p$;
9. **Else if** (C8 && C9) **then**
10. Calculate the probability p1 of the current combination;
11. $A(w,n) = p1 \times \frac{Q_{DM}}{Q_{DM}+Q_{PF}}$;
12. **End**

完整的 Matlab 代码详见附录中 evaluate.m 文件。

使用 Matlab 仿真计算出 5~20 个节点时，系统在$t = 2.5 \times 10^4\ hour$时的系统可用性及可靠性的汇总如表 6-1 所示。

表 6-1 不同节点个数对应的系统可用性和可靠性

节点数目	5	6	7	8	9	10	11	12
系统可用性（%）	12.81	34.01	55.16	71.62	82.60	89.15	92.72	94.46
系统可靠性（%）	11.85	32.67	53.95	70.92	82.19	88.81	92.24	94.18
节点数目	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
系统可用性（%）	95.19	95.36	95.25	94.99	94.67	94.30	93.92	93.52
系统可靠性（%）	95.06	95.16	94.98	94.65	94.36	94.00	93.64	93.10

可以发现，系统的可用性也是在节点个数为 14 个时取得最大值，但总是略微高于系统的可靠性，这是由于系统的可用性只考虑了系统在$t = 2.5 \times 10^4\ hour$时的状态，这里面会包含先失效后“复活”的系统[1]，但在分析系统可靠性时，这部分系统未参与计算，因而可靠性总是略小于可用性。由于“复活”现象的发生概率比较小，所以本案例中也可用系统可用性计算结果作为可靠性近似解。

6.2 系统数学模型优化

根据[1]中的分析，我们知道，当系统中所有节点的切换器 A、B 都已损坏，且一个节点处在 Master Only 的状态，其余节点处在 Slave Only 的状态时，系统的寿命会变得无限长。为避免这种情况对研究造成影响，在案例的前期研究中，我们将系统的最大寿命限制为$9.0 \times 10^4\ hour$。通过仿真我们发现，许多并未达到“寿命无限长”的状态的系统的寿命本身就可能超过$9.0 \times 10^4\ hour$却也被模型限制了寿命。

改进后的模型只对会使得系统寿命为无限长的系统做寿命上的限制。具体实现中只需先检测系统中是否所有节点内部的切换器都已损坏，再检测是否满足一个节点处于 Master Only 的状态、其余节点处于 Slave Only 的状态。可以通过增加一个函数 check_infinite 和逻辑关系式实现。详细实现见附录中 variable_length.m 及 check_infinite.m。

优化后的模型的仿真结果显示：

1）当系统的节点数目为 19 时，系统的平均寿命最长，为 207598.03 小时。

2）当系统的节点数目为 14 时，系统的可靠性最高，为 94.96%。

优化后系统的可靠性和平均寿命的最优解与优化前，均在 14 个和 19 个节点时取得最大值；同时，相比优化前的模型，系统的可靠性基本一致，但系统的平均寿命大大提升。图 6.1 给出了优化前后系统平均寿命的对比：

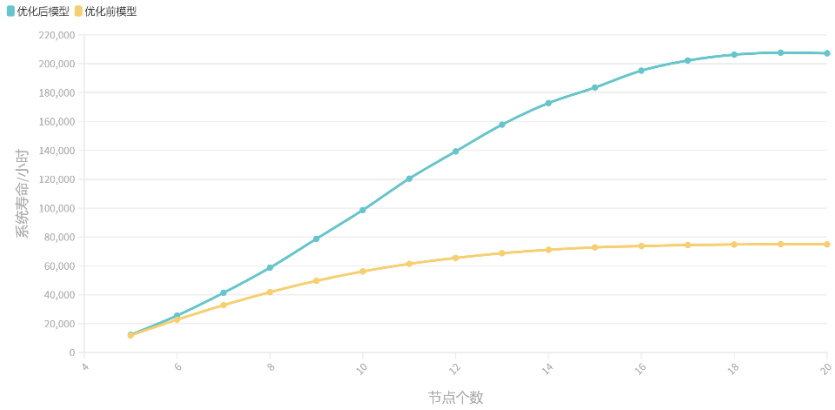

图 6.1 系统节点个数与系统平均寿命的关系

7 结论

本次研究中，我们基于蒙特卡罗方法和 Matlab 仿真分析了一个多节点声纳系统同步时钟机制的可靠性和平均寿命。通过对系统进行建模分析，并编写 Matlab 程序仿真得到当系统节点个数在 5~20 个时，有以下两个优化结论：

i） 系统的节点数目为 14 时，系统的可靠性达到最高，约为 95.16%；

ii）系统的节点数目为 19 时，系统的平均寿命达到最长，约为 75065.7 小时。

此外，本文还从概率论的角度出发，理论推到了系统的可用性，并与仿真得出的可靠性进行了对比分析。同时，针对仿真中观察到的现象，对原有的模型进行了一定的优化，并发现优化后系统的平均寿命可以得到更准确的预测。

8 致谢

在本次问题的研究中本人得到了老师和同学们的帮助，我对此表示衷心的感谢：

蒋乐天老师在授课过程中讲解了问题的研究思路、程序的设计思路以及论文的写作技巧，为本人的研究提供了很大的指导作用。同时，蒋乐天老师也在我进行工作进度汇报后向我提出了进一步优化研究的建议，这为我后续的研究和最终报告的完成提供了极大的帮助。此外，在初稿工作汇报中，许多同学对自己的探究思路的阐述，为我在程序编写过程提供了思路启发。此外，在本案例的模型、算法设计以及论文写作过程中，我也与其他同学就一些问题进行了交流和探讨，限于篇幅这里不逐一列出，本人同样要向他们表达诚挚的谢意。

9 参考文献

[1] 上海交通大学电子工程系. 工程问题建模与仿真课程讲义[EB/OL]. ftp://202.120.39.248.

[2] 司守奎.数学建模算法与应用[M]. 北京:国防工业出版社, 2015. 14-15.

附 录

一、正文引用的图表：

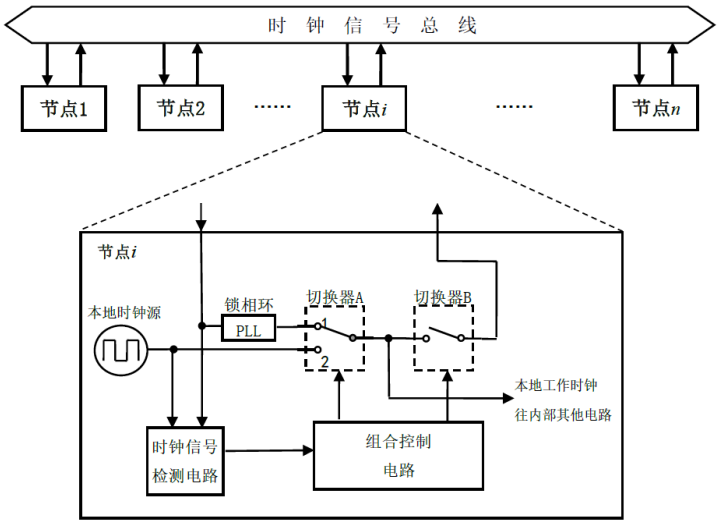

图 1 多节点声纳系统同步时钟机制的物理模型示意图[1]

表 1 切换器-节点状态关系[1]

切换器 A 状态	切换器 B 状态	节点状态	符号含义
A0	B0	N0	性能完好
	B1	N3	只能作为主节点，否则就会阻塞总线
	B2	N1	只能作为从节点
A1	B0	N1	只能作为从节点
	B1	N5	节点总是阻塞总线
	B2	N1	只能作为从节点
A2	B0	N2	只能作为主节点或不阻塞总线的失效节点
	B1	N3	只能作为主节点，否则就会阻塞总线
	B2	N4	只能作为不阻塞总线的失效节点
A3	B0	N4	只能作为不阻塞总线的失效节点
	B1	N4	只能作为不阻塞总线的失效节点
	B2	N4	只能作为不阻塞总线的失效节点

表 2 C1~C9 逻辑表达式内容[1]

符号	逻辑表达式
C1	$Q_{N5}(t) > 0$
C2	$Q_{N3}(t) > 1$
C3	$Q_{N0}(t) + Q_{N2}(t) + Q_{N3}(t) = 0$
C4	$Q_{N0}(t) + Q_{N1}(t) + 1((Q_{N2}(t) + Q_{N3}(t)) > 0) < \mathrm{k}$
C5	$Q_{N5}(t) = 0$
C6	$(Q_{N3}(t) = 1) \wedge (Q_{N0}(t) + Q_{N1}(t) \geq k - 1)$
C7	$(Q_{N3}(t) = 0) \wedge (Q_{N0}(t) \geq 1) \wedge (Q_{N1}(t) \geq k - 1)$*或* $(Q_{N0}(t) = Q_{N3}(t) = 0) \wedge (Q_{N1}(t) \geq k - 1) \wedge \&(Q_{N2}(t) \geq 1)$
C8	$Q_{N3}(t) + Q_{N5}(t) = 0$
C9	$(Q_{N0}(t) \geq 1) \wedge (Q_{N0}(t) + Q_{N1}(t) = k - 1) \wedge (Q_{N2}(t) \geq 1)$

二、代码清单：

main.m:主函数

```
clear;
tic;

mu_A = 3.70e4;
mu_B = 4.80e5;
n_samples = 1e5;  %%样本数目
n_max = 20; %%最多取点数字
life = zeros(1, n_samples);
max_life = 9e4;
mean_life = zeros(1, 20);
reliability = zeros(1, 20);
%节点状态矩阵，用以记录不同切换器状态下节点的状态
state_matrix = [0, 3, 1; 1, 5, 1; 2, 3, 4; 4, 4, 4]; %在函数
get_node_state 中使用


for n = 5 : n_max
    variable_length; %变字长仿真得到系统寿命
    mean_life(n) = mean(life);

    life_n = life;
    life_n(life_n > 2.5e4) = 1;
    life_n(life_n > 1) = 0;
    reliability(n) = sum(life_n) / n_samples; %计算可靠性

    disp(['当前系统节点个数：',num2str(n)]);
    disp(['当前系统平均寿命：',num2str(mean_life(n)),'h']);
    disp(['当前系统可靠性：',num2str(reliability(n) * 100),'%']);
    fprintf('\n');
end

[a, b] = max(mean_life);
disp(['系统平均寿命最长为:',num2str(a),'小时，','包含了',num2str(b),'个
节点']);
[c, d] = max(reliability);
disp(['系统可靠性最大为:',num2str(c * 100),'%，','包含了
',num2str(d),'个节点']);

%plot_life_figure;
plot_reli_figure;
```

variable_length.m：变字长仿真得到系统的平均寿命

```
%%变步长随机生成切换器由好变坏的时间
t_A = exprnd(mu_A, n_samples, n);
t_B = exprnd(mu_B, n_samples, n);

%%按概率随机生成切换器发生故障后会转换到哪个状态
state_A = rand(n_samples, n);
state_B = rand(n_samples, n);

state_A(state_A > 0.52) = 3;
state_A(state_A <= 0.26) = 1;
state_A(state_A < 1) = 2;

state_B(state_B <= 0.35) = 1;
state_B(state_B < 1) = 2;



A = zeros(n_samples, n);  %记录当前每个节点内部 A 切换器的状态
B = zeros(n_samples, n);  %记录当前每个节点内部 B 切换器的状态
state_nodes = zeros(n_samples, n);    %记录系统内节点的状态

%遍历所有生成的系统，并求得其寿命
for s = 1 : n_samples
    %随机生成转换事件的节点序号、发生时间、转换后状态
    change_A = [1 : n; zeros(1, n); t_A(s, :); state_A(s, :)];
    %第二行数据用 0 表示是 A 发生了变换
    %记某一列数据依次为 x,y,z,w：第 x 个节点的 A(y = 0 时）转换器在 z 时刻变为
了状态 w
    change_B = [1 : n; ones(1, n); t_B(s, :); state_B(s, :)];
    %第二行数据用 1 表示是 B 发生了变换
    change_AB = [change_A, change_B]';

    change_sort = sortrows(change_AB, 3); %按照时间先后排序
    %记某一行数据依次为 x,y,z,w：第 x 个节点的 A(y = 0 时）转换器在 z 时刻变为
了状态 w

    for i = 1 : 2 * n %遍历所有事件
        change = change_sort(i, :);

        %PDF 中处理方式：如果超过最大限度生命，则设为封顶
        if change(3) >= max_life
            life(s) = max_life;
            break;
        end
```

```
%改进处理方式：只有系统达到“寿命无限”时才设为封顶
%         if (check_infinite(state_nodes(s, :), n))
          &&(sum(A(s,:)~=0) + sum(B(s,:)~=0) == 2 * n)
          %判断节点中切换器是否都损坏，且1个MO、n-1个SO
%            life(s) = max_life;
%            break;
%         end

        if (change(2) == 0)
            A(s, change(1)) = change(4);
        else
            B(s, change(1)) = change(4);
        end

%由切换器的状态推断节点的状态
%节点状态：0-Perfect Fuctioning | 1-Slave Only   | 2-Disable/Master
%节点状态：3-Master Only        | 4-Disable Only | 5-Fail Bus
        state_nodes(s, change(1)) =
state_matrix( A(s,change(1))+1 , B(s,change(1))+1 );

        state_sys = get_system_state(state_nodes(s, :));%判断系统状态

        if (state_sys == 0) %当系统失效时，系统寿命设为当前变化发生的时间
            life(s) = change(3);
            break;
        end
    end
end
```

get_system_state.m：根据节点状态推断系统当前工作状态

```
%返回 0 说明系统失效，1 正常工作
function state = get_system_state(state_nodes)
    %统计系统中处于各状态的节点个数
    PF = sum(state_nodes == 0);
    SO = sum(state_nodes == 1);
    DM = sum(state_nodes == 2);
    MO = sum(state_nodes == 3);
    DN = sum(state_nodes == 4);
    FB = sum(state_nodes == 5);
    C1 = (FB >= 1);
    C2 = (MO >= 2);
    C3 = (PF + MO + DM == 0);
    C4 = ((PF + SO + ((MO + DM) > 0)) < 5);
    C5 = (FB == 0);
```

```
    C6 = (MO == 1 && PF + SO >= 4);
    C7 = ((MO == 0 && PF >= 1 && PF + SO >= 5)||(MO == 0 && PF ==
0 && DM >= 1 && SO >= 4));
    C8_C9 = ((FB + MO == 0)&&((PF >= 1)&&(PF + SO == 4)&&(DM >=
1)));

    if (C1||C2||C3||C4)
        state = 0;
    elseif (C5&&(C6||C7))
        state = 1;
    elseif (C8_C9)
        state = (rand() <= (DM / (DM + PF)));
    end
end
```

plot_life_figure：绘制系统平均寿命与节点个数的关系图

```
x = 5 : n_max;
R = mean_life(5 : n_max);
[~, R_mpos] = max(R);
R_mpos = R_mpos + 4;
figure(1),plot(x, R, '.-','markersize', 20);
xlabel('系统节点个数','fontsize', 20)
ylabel('系统平均运行寿命/小时','fontsize', 20)
title('系统平均寿命与节点个数的关系','fontsize', 20)
grid on
hold on
plot(R_mpos,R(R_mpos-4),'r.','markersize',30);
hold off
```

plot_reli_figure：绘制系统可靠性与节点个数的关系图

```
x = 5 : n_max;
R = reliability(5 : n_max);
[~, R_mpos] = max(R);
R_mpos = R_mpos + 4;
figure(1),plot(x, R, '.-','markersize', 20);
xlabel('系统节点个数','fontsize', 20)
ylabel('t=25000h 时系统的可靠性','fontsize', 20)
title('t=25000h 时系统的可靠性与节点个数的关系','fontsize', 20)
grid on
hold on
plot(R_mpos,R(R_mpos-4),'r.','markersize',30);
hold off
```

check_infinite：检查系统是否满足 1 个 MO，其余 SO

```
function state = check_infinite(state_nodes, n)
    SO = sum(state_nodes == 1);
    MO = sum(state_nodes == 3);
    if ((MO == 1)&&(SO == n - 1))
        state = 1;
    else
        state = 0;
    end
end
```

evaluate.m：理论推导系统可用性

```
w = 2.5e4;
p_A0 = exp(-w / mu_A);
p_B0 = exp(-w / mu_B);
p_A1 = 0.26 * (1 - p_A0);
p_A2 = 0.26 * (1 - p_A0);
p_A3 = 0.48 * (1 - p_A0);
p_B1 = 0.35 * (1 - p_B0);
p_B2 = 0.65 * (1 - p_B0);
p_PF = p_A0 * p_B0;
p_MO = (p_A0 + p_A2) * p_B1;
p_SO = p_A0 * p_B2 + p_A1 * p_B0 + p_A1 * p_B2;
p_FB = p_A1 * p_B1;
p_DM = p_A2 * p_B0;
p_DN = p_A2 * p_B2 + p_A3;
availability = zeros(1, 20);
for n = 5 : 20 %遍历5~20个点的情况
    nodes = zeros(1, n);
    for PF = 0 : n
        for MO = 0 : (n - PF)
            for SO = 0 : (n - PF - MO)
                for FB = 0 : (n - PF - MO - SO)
                    for DM = 0 : (n - PF - MO - SO - FB) %遍历所有组合
                        DN = n - PF - MO - SO - FB - DM;
                        if (FB >= 1)||(MO >= 2)||(PF + MO + DM ==
0)||(PF + SO + sum(MO + DM > 0) < 5)
                            continue;
                        end
                        if (MO == 1 && PF + SO >= 4)||(MO == 0 &&
PF >= 1 && PF + SO >= 5)||(MO == 0 && PF == 0 && DM >= 1 && SO >=
4)
                            availability(n) = availability(n) +
nchoosek(n, PF) * nchoosek(n-PF, MO) * nchoosek(n-PF-MO, SO)
```

```
* ...
                                        nchoosek(n-PF-MO-SO, FB) *
nchoosek(n-PF-MO-SO-FB, DM) * ...
                                        nchoosek(n-PF-MO-SO-FB-DM,
DN) * p_PF^(PF) * p_MO^(MO) * ...
                                        p_SO^(SO) * p_FB^(FB) *
p_DM^(DM) * p_DN^(DN);
                    else
                        availability(n) = availability(n) +
nchoosek(n, PF) * nchoosek(n-PF, MO) * nchoosek(n-PF-MO, SO)
* ...
                                        nchoosek(n-PF-MO-SO, FB) *
nchoosek(n-PF-MO-SO-FB, DM) * ...
                                        nchoosek(n-PF-MO-SO-FB-DM,
DN) * p_PF^(PF) * p_MO^(MO) * ...
                                        p_SO^(SO) * p_FB^(FB) *
p_DM^(DM) * p_DN^(DN) * DM / (DM + PF);
                    end
                end
            end
        end
    end
end
```