云计算环境下差别矩阵知识约简算法研究

钱 进[1,2,3] 苗夺谦[1,3] 张泽华[1,3]

(同济大学计算机科学与技术系 上海 201804)[1] (江苏技术师范学院计算机工程学院 常州 213001)[2]

(同济大学嵌入式系统与服务计算教育部重点实验室 上海 201804)[3]

摘 要 知识约简是粗糙集理论的重要研究内容之一。经典的差别矩阵知识约简算法只能处理小数据集,而已有的任务并行的知识约简算法是假设所有数据一次性装入内存中,这显然不适合处理海量数据。为此,剖析了差别矩阵元素的特性,根据属性(集)的不可辨识性和云计算技术 MapReduce 设计了适合数据并行的差别矩阵,并首次提出了面向大规模数据的差别矩阵知识约简算法。实验结果表明该知识约简算法是有效可行的,且具有较好的可扩展性。

关键词 云计算,差别矩阵,知识约简,粗糙集

中图法分类号 TP18 **文献标识码** A

Research on Discernibility Matrix Knowledge Reduction Algorithm in Cloud Computing

QIAN Jin[1,2,3] MIAO Duo-qian[1,3] ZHANG Ze-hua[1,3]

(Department of Computer Science and Technology, Tongji University, Shanghai 201804, China)[1]

(School of Computer Engineering, Jiangsu Teachers University of Technology, Changzhou 213001, China)[2]

(Key Laboratory of Embedded System & Service Computing, Ministry of Education of China, Tongji University, Shanghai 201804, China)[3]

Abstract Knowledge reduction is one of the important research issues in rough set theory. Classical knowledge reduction algorithms can only deal with small datasets, while the existing parallel knowledge reduction algorithms assume all the datasets can be loaded into the main memory and only implement reduction tasks concurrently, which is infeasible for handling large-scale data. Massive data with high dimension makes attribute reduction a challenging task. To solve this problem, the characteristics of discernibility matrix cells were analyzed, and discernibility matrix for data parallel was designed in terms of the indiscernibility of the attribute(s) and MapReduce programming model. Thus, large-scale data oriented discernibility matrix knowledge reduction algorithm in cloud computing was proposed. The experimental results demonstrate that our proposed algorithm can scale well and efficiently process large-scale datasets on commodity computers.

Keywords Cloud computing, Discernibility matrix, Knowledge reduction, Rough set

粗糙集理论(Rough Set Theory)[1]正被广泛地应用于机器学习、数据挖掘及模式识别等领域。在粗糙集理论中,知识约简是重要研究内容之一,也是知识获取的关键步骤。所谓知识约简是指在保持知识库的分类能力不变的条件下,删除其中不必要的知识。通过删除冗余属性,可以大大提高信息系统潜在知识的清晰度。因此,研究如何提高知识约简算法效率是十分重要的工作。

目前已提出许多知识约简算法,主要分为基于正区域的属性约简算法[2]、基于信息论的属性约简算法[3,4]和基于差别矩阵的属性约简算法[5-8]等。基于差别矩阵的约简算法引起众多学者关注,他们主要研究如何改进差别矩阵。这些方法在一定程度上提高了小数据集上差别矩阵属性约简算法效率。

当面对海量数据时,目前提出的差别矩阵属性约简算法都无法进行知识约简。因此,并行知识约简是解决海量数据挖掘问题的一个重要途径。已有并行知识约简算法[9,10]利用任务并行来提高知识约简算法效率。然而,任务并行的知识约简算法仍然假设所有数据一次性装入内存中,这显然不适合处理海量数据。云计算(Cloud Computing)[11]是近几年新提出的一种商业计算模型,是分布式计算、并行计算和网格计算的发展。Google 公司提出了分布式文件系统 GFS(Google File System)[12]和并行编程模式 MapReduce[13]。云计算技术已经初步应用于机器学习领域[14,15],但至今还没有应用到粗糙集理论知识约简算法中。

本文深入研究了 MapReduce 编程框架,对现有差别矩阵属性约简算法进行具体剖析,利用属性(集)的不可辨识性和 MapReduce 技术来设计适合大规模数据集的差别矩阵,提出了面向海量数据的云计算环境下差别矩阵知识约简算法,并

到稿日期:2010-09-13 返修日期:2010-12-27 本文受国家自然科学基金(60970061,61075056),上海市重点学科建设项目(B004),江苏省属高校自然科学资金项目(09KJD520004)资助。

钱 进(1975—),男,博士生,讲师,CCF 会员,主要研究方向为粗糙集理论、云计算,E-mail:qjqjlqyf@163.com;苗夺谦(1964—),男,博士,教授,博士生导师,主要研究方向为粗糙集理论、粒计算、云计算等;张泽华(1981—),男,博士生,主要研究方向为粗糙集理论、不确定推理。

利用 Hadoop 开源平台[16]实现了所提出的约简算法。实验结果表明，该算法不仅具有高效性，而且能够处理海量数据。

1 相关理论

1.1 粗糙集理论

下面简要介绍主要用到的一些 Rough 集的基本概念，详细情况参考有关文献[1,2,8]。

定义 1[1] 五元组 $S=\langle U,C,D,V,f\rangle$ 是一个决策表，每一个属性子集 $R\subseteq C\cup D$ 决定了一个二元不可区分关系 IND(R)：

$$\mathrm{IND}(R)=\{(x,y)\in U\times U\mid \forall a\in R, f(x,a)=f(y,a)\}$$

关系 IND(R)构成了 U 的一个划分，用 U/R 表示。U/R 中的任何元素 $[x]_R=\{y\mid \forall a\in R, f(x,a)=f(y,a)\}$ 称为等价类。

定义 2[1] 在决策表 S 中，对于每个子集 $X\subseteq U$ 和不可区分关系 $R\subseteq C\cup D$，X 的下近似集与上近似集分别可以由 R 的基本集定义如下：

$$R_-(X)=\bigcup\{Y\in U/R: Y\subseteq X\}$$

$$R^-(X)=\bigcup\{Y\in U/R: Y\cap X\neq\emptyset\}$$

定义 3[1] 在决策表 S 中，$\forall R\subseteq C, X\subseteq U$，用 $\underline{RX}$ 表示 X 的 R_- 的下近似集，决策属性 D 的 R_- 正区域 $POS_R(D)$ 定义为

$$POS_R(D)=\bigcup_{X\in U/D}\underline{RX}$$

定义 4[2] 在决策表 S 中，记 $U/C=\{[x_1']_C,[x_2']_C,\cdots,[x_s']_C\}$，$U'=\{x_1',x_2',\cdots,x_s'\}$，$U'_{POS}=\{x'_{i_1},x'_{i_2},\cdots,x'_{i_t}\}$，其中，$U'_{POS}$ 中的对象为相容对象，U'_{BND} 为 $U'-U'_{POS}$，则 $S'=(U'=U'_{POS}\cup U'_{BND},C,D,V,f)$ 为简化决策表。

不失一般性，假设决策表 S 仅有一个决策属性 D，其不同决策属性值映射为 $1,\cdots,k$，记为 $U/D=\{D_1,D_2,\cdots,D_k\}$，其中，$D_i=\{x\in U\mid f(x,D)=i\}$，$i=1,2,\cdots,k$。

定义 5[8] 在简化决策表 S' 中，U'_{BND} 中的对象为矛盾对象，将所有矛盾对象集记为 D_{k+1}，其决策属性值标记为'?'，映射为 $k+1$。若 $D_{k+1}=\emptyset$，则称决策表 S' 是相容(一致)决策表；否则是不相容(不一致)决策表。

定义 6[8] 在简化决策表 S' 中，其简化差别矩阵 $M'=(m_{ij})$，其任一元素定义为

$$m_{ij}=\begin{cases}\{a\in C: f(x,a)\neq f(y,a)\}, & x\in D_i, y\in D_j\\ \emptyset, & \text{其它}\end{cases}$$

式中，$1\leqslant i<j\leqslant k+1$。

1.2 云计算技术

MapReduce 是一种处理海量数据的并行编程模式。用户将实际应用问题分解成若干可并行操作的子问题，设计相应的 Map 和 Reduce 两个函数就能将自己的应用程序运行在云计算环境中。Map 函数是接收一组输入键值对〈in_key,in_value〉，然后通过某种计算，产生中间结果键值对；而 Reduce 函数接收一个中间结果 key 和对应该 key 的一组 value 值，然后通过归并处理，最终形成〈key,final_value〉。

2 云计算环境下差别矩阵属性约简算法

2.1 适用于 MapReduce 编程模式的差别矩阵

假设决策表 S 可以看成由 $k+1$ 个子决策表组成，且每个子决策表包含相同决策的对象。因此，决策表 S 是"相容决策表"(所有不相容的对象当成一类对象处理)。假设属性 c 有 r 个不同属性值，将其映射为 $1,\cdots,r$。记决策属性映射值为 i 的子决策表中 c 的映射值为 p 的对象个数为 n_p^i。若两个对象决策属性值和 c 对应的属性值也不同，则 c 能够辨识这两个对象。

定义 7 在决策表 S 中，$c\in C$，属性 c 能够辨识的对象对(Object Pair)总数为 $\mathrm{DIS}_c^D=\sum_{1\leqslant i<j\leqslant k+1}\sum_{1\leqslant p<q\leqslant r}n_p^i n_q^j$。

在云计算环境下，由于不同等价类可能分布在多个节点上，而 DIS_c^D 计算涉及多个不同的等价类，因此 DIS_c^D 无法进行计算。故转而求属性 c 不能辨识的对象对个数，即当任意两个对象决策属性值不同，而条件属性 c 对应的值相同时，则 c 不能辨识这两个对象。

定义 8 在决策表 S 中，$c\in C$，属性 c 不能辨识的对象对总数为：$\widetilde{DIS}_c^D=\sum_{1\leqslant p<r}\sum_{1\leqslant i<j\leqslant k+1}n_p^i n_p^j$。

定理 1 在决策表 S 中，$c\in C$，则 $DIS_c^D+\widetilde{DIS}_c^D=\sum_{1\leqslant i<j\leqslant k+1}|D_i||D_j|$。

定义 9 在决策表 S 中，$A\subseteq C$，属性集 A 不能辨识的对象对总数为：$\widetilde{DIS}_A^D=\sum_{1\leqslant p<r}\sum_{1\leqslant i<j\leqslant k+1}n_p^i n_p^j$。

定理 2 在决策表 S 中，$c\in C-A$，则 $\widetilde{DIS}_{A\cup c}^D\leqslant\widetilde{DIS}_A^D$。

性质 1 在决策表 S 中，$P\subseteq Q$，则 $\widetilde{DIS}_Q^D\leqslant\widetilde{DIS}_P^D$。

定理 3 在决策表 S 中，$A\subseteq C$，A 是 C 相对于决策属性 D 的一个约简的充分必要条件为

(1) $\widetilde{DIS}_A^D=\widetilde{DIS}_C^D$；

(2) $\forall a\in A, \widetilde{DIS}_{A-\{a\}}^D>\widetilde{DIS}_A^D$。

由定义 9 可以看出，属性集 A 不能辨识的对象对总数由不同等价类所不能辨识的对象对个数累加得到，而不同等价类是可以并行计算的。因此，原有的差别矩阵知识约简算法经过上面处理后就能够在 MapReduce 编程框架下并行实现，从而可以处理大规模数据。本文虽不生成差别矩阵元素，但基本思想与之对应，故仍称为差别矩阵知识约简算法。

2.2 云计算环境下差别矩阵知识约简算法

在云计算环境下差别矩阵知识约简算法中，Map 函数主要完成不同数据块中的等价类计算，而 Reduce 函数主要计算同一个等价类所不能辨识的对象对个数。为了减少不必要的存储和计算，给出下面定理。

定理 4 对于一个等价类 A_p，$A_p\subseteq POS_A(D)$，则 $\sum_{1\leqslant i<j\leqslant k+1}n_p^i n_p^j=0$。

性质 2 对于一个等价类 A_p，若 $A_p\subseteq BND_A(D)$，则 $\sum_{1\leqslant i<j\leqslant k+1}n_p^i n_p^j>0$。

在 Reduce 过程中，当一个等价类辨识的对象对个数为 0 时，不输出信息，从而节省存储空间，降低网络传输时间。

下面，给出 Pawlak 粗糙集模型下差别矩阵知识约简算法，分别设计了 Map(算法 1 用于计算等价类)、两个 Reduce(算法 2 用于计算简化决策表，算法 3 用于计算同一等价类不能辨识的对象对个数)和主程序 ParallelDis(算法 4 用于选择最优候选属性，计算约简)。

算法 1 Map (string EquivalenceClass, string value)

输入：属性集合 A，对象 x

输出：〈EquivalenceClass,〈$d(x)$,1〉

//EquivalenceClass 是由属性 A 在对象 x 上导出的等价类，〈$d(x)$,1〉

是指该等价类决策值为 $d(x)$ 出现 1 次

1. EquivalenceClass=“”;
2. 对任意 $a\in A$ 进行如下操作:

   EquivalenceClass+=$a(x)$+“ ”;
3. 输出〈EquivalenceClass,〈$d(x)$,1〉〉

算法 2 Reduce(string EquivalenceClass, pairs[〈$d1$, $n1$〉,〈$d2$,$n2$〉…])

输入:等价类 EquivalenceClass 及对应的不同决策值列表 pairs

输出:简化决策表

1. 对 pairs[〈$d1$,$n1$〉,〈$d2$,$n2$〉….]进行如下操作:
2. 如果决策值都相同,输出〈EquivalenceClass,$d1$〉;
3. 否则,输出〈EquivalenceClass,‘?’〉

算法 3 Reduce(string EquivalenceClass, pairs[〈$d1$,$n1$〉, 〈$d2$,$n2$〉…])

输入:等价类 EquivalenceClass 及对应的决策值列表 pairs

输出:〈EquivalenceClass,IndisObjectPairSum〉

//EquivalenceClass 是等价类,IndisObjectPairSum 是该等价类所不能辨识的对象对个数

1. 对任意 pair 〈d,n〉 ∈ pairs[〈$d1$,$n1$〉,〈$d2$,$n2$〉….]进行如下操作:
2. {统计不同决策值出现的频率(n_p^1,n_p^2,…,n_p^{k+1});}
3. IndisObjectPairSum=$\sum_{1\leqslant i<j\leqslant k+1} n_p^i n_p^j$;
4. 如果 IndisObjectPairSum 不为 0

   输出〈EquivalenceClass,IndisObjectPairSum〉

算法 4 主程序 ParallelDis

输入:一个决策表 S

输出:一个约简 Red

1. 启动一个 Job,执行 Map 和算法 2 的 Reduce 函数,得到“相容决策表”;
2. Red=∅;
3. 启动一个 Job,执行 Map 和算法 3 的 Reduce 函数,计算$\widetilde{DIS}^D_{Red\cup a}$($a\in C-Red$),选择 $a_l=\min_{a\in C-Red}\{\widetilde{DIS}^D_{Red\cup a}\}$(若这样的 a_l 不唯一,则任选其一);Red=Red∪$\{a_l\}$;
4. 重复步骤 3,直到$\widetilde{DIS}^D_{Red}=\widetilde{DIS}^D_C$;
5. 启动一个 Job,执行 Map 和算法 3 的 Reduce 函数,从 Red 的尾部开始从后往前对每个属性 a 进行判断是否可省,若$\widetilde{DIS}^D_{Red-\{a\}}=\widetilde{DIS}^D_{Red}$,则说明 a 是可省的,Red=Red−{a}。
6. 输出 Red。

3 实验

本节主要从运行时间、加速比(Speedup)和可扩展性(Scaleup)[15] 3 个方面对所提出的 ParallelDis 算法性能进行评价。选用 UCI 机器学习数据库 tic-tac-toe-endgame 和 mushroom 测试算法正确性,用两个人工数据集(Dataset1 和 Dataset2)测试性能。两个人工数据集分别包含 500 万和 1000 万条记录,30 个属性。利用 hadoop 和 Java 在 4 台普通计算机(3.0GHz CPU,1Gb 内存)构建的集群上进行实验。

(1)运行时间

运行结果如表 1 所列。表中“—”表示传统方法无法运行或 ParallelDis 运行时间太长。从表 1 可以看出,小数据集不适宜使用 MapReduce 技术,同时任务并行方式比任务串行运行时间更短。分别对 500 万、1000 万条记录的 Dataset1 和 Dataset2 在不同数量的节点上采用任务并行方式进行测试。从图 1 可以看出,随着计算机节点增多,运行时间减少。

表 1 单机上运行时间(s)

数据集	传统方法	ParallelDis	
		任务串行	任务并行
TicTac	0.016	3442.469	710.078
mushroom	0.625	6357.156	348.266
Dataset1	—	—	22,974.374
Dataset2	—	—	48,085.047

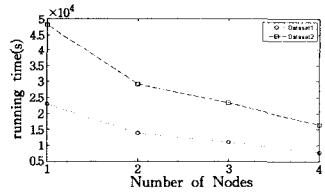

图 1 运行时间

(2)加速比

加速比是指将数据集规模固定,不断增大计算机节点数时并行算法的性能。为了测定加速比,保持不同大小的 500 万、1000 万记录的两个数据集不变,增加计算机节点数至 4 台。一个理想的并行算法加速比是线性的,即当计算机节点数增加至 m 时,其加速比为 m。然而,由于计算机间互相通信,并行算法存在通信开销,因此其实际加速比低于理想的加速比。图 2 显示了不同数据集的加速比大小。实验结果表明 ParallelDis 算法具有较好加速比性能。

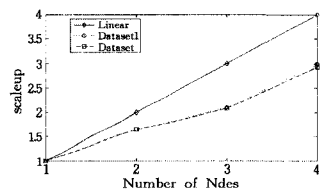

图 2 加速比

(3)可扩展性

可扩展性是指按与计算机节点数成比例地增大数据集规模时并行算法的性能。为了测定可扩展性,实验将数据集复制 2、3 和 4 次,分别在 2 台、3 台和 4 台计算机节点上运行。图 3 显示了可扩展性结果。当数据集规模按计算机节点数同比例增长时,运行时间稍微变长,其原因是 ParallelDis 生成的等价类太多而造成串行部分运行时间过长。

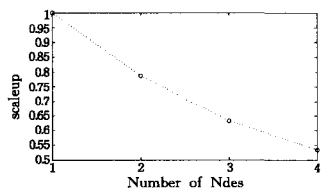

图 3 可扩展性

结束语 传统的知识约简算法通过基数或计数排序算法来快速计算等价类和改进差别矩阵以提高算法效率,但只适合处理小数据集。为了进行面向大规模数据集约简,深入分析原有差别矩阵约简算法,利用属性(集)的不可辨识性来设计适合处理海量数据的差别矩阵,提出了云计算环境下差别矩阵知识约简算法,并利用 Hadoop 在普通计算机的集群上进行实验。实验结果表明该知识约简算法是正确有效的,能够处理大规模数据集。

参 考 文 献

[1] Pawlak Z. Rough Sets [J]. International Journal of Computer and Information Science, 1982, 11(5): 341-356

[2] 徐章艳，刘作鹏，杨炳儒，等. 一个复杂度为 max(O(|C||U|), O(|C|²|U/C|))的快速属性约简算法[J]. 计算机学报，2006，29(3)：391-399

[3] 苗夺谦，胡桂荣. 知识约简的一种启发式算法[J]. 计算机研究与发展，1999，36(6)：681-684

[4] 王国胤，于洪，杨大春. 基于条件信息熵的决策表约简[J]. 计算机学报，2002，25(7)：759-766

[5] Skowron A, Rauszer C. The discernibility matrices and functions in information systems[C]// Słowiński R, ed. Intelligent Decision Support Handbook of Applications and Advances of the Rough set Theory. Kluwer Academic Publishers, Dordrecht, 1992, 311-362

[6] 杨明. 一种基于改进差别矩阵的属性约简增量式更新算法[J]. 计算机学报，2007，30(5)：815-822

[7] Miao D Q, Zhao Y, Yao Y Y, et al. Relative reducts in consistent and inconsistent decision tables of the Pawlak rough set model[J]. Information Sciences, 2009, 179: 4140-4150

[8] Qian J, Miao D Q, Zhang Z H, et. al. Hybrid approaches to attribute reduction based on indiscernibility and discernibility relation[J]. Int. J. Approx. Reason, 2010, doi: 10. 1016 /. ijar. 2010. 07. 011

[9] 王立宏，吴耿锋. 基于并行协同进化的属性约简[J]. 计算机学报，2003，26(5)：630-635

[10] 肖大伟，王国胤，胡峰. 一种基于粗糙集理论的快速并行属性约简算法[J]. 计算机科学，2009，36(3)：208-211

[11] 刘鹏. 云计算[M]. 北京：电子工业出版社，2010

[12] Ghemawat S, Gobioff H, Leung S T. The Google file system[J]. SIGOPS-Operating Systems Review, 2003, 37(5): 29-43

[13] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113

[14] Chu C T, Kim S, Lin Y A, et al. MapReduce for machine learning on multicore[C]// Proceedings of NIPS. vol. 19, 2006

[15] Zhao Wei-zhong, Ma Hui-fang, He Qing. Parallel k-Means clustering based on MapReduce[C]// Jaatun M G, Zhao G, Rong C, eds. CloudCom 2009, LNCS 5931. 2009: 674-679

[16] Hadoop[EB/OL]. http://lucene. apache. org/hadoop

(上接第 164 页)

S_Cruisecontrollaws，图 7 是其软构件结构图。子系统 S_Hci 中有两个进程：一个进程有 4 个线程，另一个进程包含 1 个线程；子系统 S_Cruisecontrollaws 有一个进程，该进程包含 2 个线程。

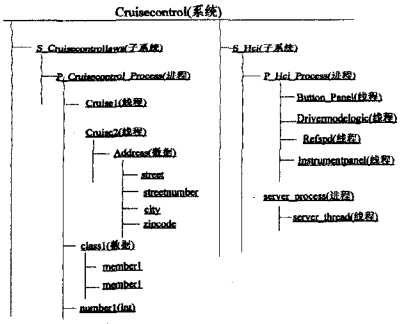

图 7 AADL 模型航行控制系统结构

4.1 航行控制系统的转换

将文件 cruise. aaxl 和 cruise_Instance. aaxl[3] 作为工具 AADLToSimCode 的输入，该工具自动生成的仿真代码文件为 Curisecontrol _ Process. cpp、Hci _ Process. cpp、Hci _ Process. h、S _ tb. cpp、Server _ Process. cpp、Curisecontrol _ Process. h、Cruisecontrol. h、Cruisecontrollaws. h、Hci. h、S_tb. h 和 Server _ Process. h，其中 Cruisecontrol. h、Hci _ Process. cpp 和 Hci_Process. h 的部分代码已在第 2 节展示。

4.2 调度仿真

```
//File:cruise_main.cpp
#include "systemc.h"
#include "S_Cruisecontrol.h"
#include "S_tb.h"
int sc_main(int argc, char *argv[]){
    sc_signal<short> Breakpedalpressed;
    sc_signal<short> Clutchpedalpressed;
    sc_signal<float> Autospd_Mph;
    //调用，对仲裁器
    sc_clock clk("clk", 20, SC_MS);
    cout << "Initialing stb..." << endl;
    S_tb stb("stb");
    stb.clk(clk);
    stb.breakpedalpressed(Breakpedalpressed);
    stb.Clutchpedalpressed(Clutchpedalpressed);
    stb.autospd_Mph(Autospd_Mph);
    cout << "successfully create instance stb!" << endl;
    cout << "Initialing scruise..." << endl;
    S_Cruisecontrol scruise("scruise");
    scruise.clk(clk);
    scruise.breakpedalpressed(Breakpedalpressed);
    scruise.clutchpedalpressed(Clutchpedalpressed);
    scruise.Autospd_Mph(Autospd_Mph);
    cout << "successfully create instance scruise!" << endl <<endl;
    sc_start(200000, SC_MS);
    cout << "finished at time " << sc_time_stamp() << endl;
    return 0;
```

图 8 仿真驱动程序

在 4.1 节生成的仿真代码的基础上，项目组撰写了仿真启动和激励产生文件 Curise_main. cpp，如图 8 所示。SystemC 仿真内核支持仿真进程的调度执行，直接控制并发进程的调度策略。图 9 是基于 SystemC 仿真内核对 AADL 线程调度仿真结果的部分截图，显示了线程 Button_Panel 调度执行的起始时间、终止时间以及线程与其它线程的数据和事件的通讯过程。

图 9 基于 SystemC 仿真内核的调度仿真

结束语 通过本文的研究成果，用户可以实现基于 SystemC AADL 软构件的仿真，包括软构件之间交互、执行时间和以及线程调度的仿真等。用户也可以将本文的研究成果与基于 SystemC 的 AADL 执行平台构件仿真相结合，对软硬件进行协同仿真，根据仿真结果迭代构造和精化设计模型，以尽早发现设计模型中存在的问题，保障设计模型的质量，进而保证任务和安全关键领域软件系统的质量。

目前由于 SystemC 仿真内核不支持线程的占先调度策略，项目组正在研制基于 SystemC 的占先调度器。另外，项目组在本文基础上，已实现了基于 SystemC 的实时性仿真和流延迟仿真，正在撰写基于 SystemC 的实时性仿真和流延迟仿真相关论文。

参 考 文 献

[1] 杨志斌，皮磊，等. 复杂嵌入式实时系统体系结构设计与分析语言：AADL[J]. Journal of Software, 2010, 21(5): 899-915

[2] de las Heras E, Villar E. Specification for SystemC-AADL Interoperability, Intelligent Solutions in Embedded Systems[C]// 2007 Fifth Workshop. 2007: 76-86

[3] http://www. aadl. info/aadl/currentsite/examplemodel. html

[4] The SEI AADL Team. An extensible open source AADL tool environment (OSATE)[EB/OL]. http://www. aadl. info/aadl/downloads/osate13, 2006