

# 异构 GPU 集群的任务调度方法研究及实现

黄锦增, 陈 虎, 赖路双

(华南理工大学 软件学院, 广东 广州 510006)

**摘 要:** GPU 集群已经成为高性能计算的重要方式, 特别对于计算密集型应用, 具有成本低、性能高、功耗小的优势。为了解决 GPU 集群系统运行中的任务负载均衡问题, 文中提出了一种面向计算密集型应用的异构 GPU 集群调度方法, 该方法可以自动发现计算节点, 并动态估计计算节点的计算能力, 并根据计算能力、任务的计算强度和优先级在异构 GPU 集群上合理分配计算资源。同时, 该系统还具有容错能力, 能够处理计算节点的意外退出, 可恢复意外退出计算节点的计算任务, 并动态适应系统的计算规模。通过实验表明, 文中采用的策略达到了预期目的。

**关键词:** 负载均衡; 异构 GPU 集群; 任务调度; 动态适应

**中图分类号:** TP311

**文献标识号:** A

**文章编号:** 1673-629X(2012)05-0032-05

## Research and Implementation of Task Schedule Method on Heterogeneous GPU Cluster

HUANG Jin-zeng, CHEN Hu, LAI Lu-shuang

(School of Software Engineering, South China University of Technology, Guangzhou 510006, China)

**Abstract:** GPU cluster has become an important method for high performance computing, especially for compute-intensive applications. It has many advantages, such as low cost, high performance and low power consumption. To solve the load balancing problem of GPU cluster system, propose an algorithm for heterogeneous GPU cluster, it can automatically identify computation nodes, dynamically estimate the computing capability of these nodes and allocate resources in heterogeneous GPU cluster based on computation nodes' capability, tasks' computing strength and priority. At the same time, the system is also fault tolerant, which is able to handle unexpected exit of computation nodes, recover the computing task of calculation nodes out of an unexpected exit and dynamically adapt to the calculation size of the system. The experiment result shows this strategy achieves desired purpose.

**Key words:** load balance; heterogeneous GPU cluster; task schedule; dynamical adaptation

### 0 引 言

由于 GPU 出色的浮点处理计算能力, 使用 GPU 集群来处理计算密集型任务已开始成为高性能计算发展的趋势之一。

但是在 GPU 集群<sup>[1]</sup>的任务调度中有四个重要问题需要解决:

1、GPU 集群的异构特征。由于 GPU 的发展非常迅速, 不同时期、不同价格的 GPU 计算节点的计算能力差异很大。在实际应用系统中, 往往都是由多种 GPU 节点构成的异构集群, 如何在异构 GPU 集群上合理分配计算任务是发挥集群效率的重要因素。

2、GPU 集群的可伸缩能力。在实际应用中, 往往需要动态扩展或减少系统的计算规模, 需要动态调度系统能在持续运行的情况下适应计算集群规模的变

化。

3、GPU 集群的容错能力。由于计算任务的计算时间长, 系统的计算节点数目多, 整个系统的无故障工作时间短, 需要任务调度机制能及时发现故障节点, 并将故障节点的任务重新发送给其他正常节点。

4、任务的优先级调度策略。用户提交的任务往往具有不同的优先级别, 如何根据预先设定的优先级策略合理划分计算资源也是集群调度中的重要问题。

如何实现任务负载均衡<sup>[2,3]</sup>是整个 GPU 集群系统的核心问题, 因为任务的不恰当调度会使得系统真正的潜在计算能力不能被开发出来, 并且可能会抵消并行化带来的收益。调度的目标是通过将任务正确分配到各个 GPU 处理单元, 并使得任务按照一定的顺序执行, 以最小化并行应用程序的完成时间。

从广义上来讲, 负载均衡算法主要可分为两大类, 即静态算法和动态算法<sup>[4-6]</sup>。静态算法仅适用于在分发任务前就知道各个节点的相关信息这种情况下。但这种方法比较刻板, 没有适应节点信息改变的能力, 所

收稿日期: 2011-09-21; 修回日期: 2011-12-17

作者简介: 黄锦增(1987-), 男, 广东人, 硕士研究生, 研究方向为高性能计算; 陈 虎, 副教授, 博士, 研究方向为计算机系统结构。

以实用性较小。对于现在实际应用的大型集群系统来说,都会采用动态负载均衡算法,即能通过相应系统状态来适应负载的动态均衡。它能很好地利用系统资源,提高系统的性能和利用率,但它的设计实现远比静态负载均衡算法复杂,它需要收集系统各节点的相关信息,关注它们的当前状态,然后再根据这些收集到的信息来分派任务。

Paul Werstien<sup>[7]</sup>等提出了一种利用分散的特性来避免瓶颈和单节点失效的问题,除了把 CPU 队列长度作为度量因素之外,还考虑了 CPU 和内存的利用率。他的实验结果表明他提出的这种算法比传统的只考虑 CPU 队列长度的算法能得到更好的结果。

Min Choi<sup>[8]</sup>等提出一种新的度量因素,称为一组有效任务,是为了解决由于任务资源需求的不确定带来的问题。这种算法主要针对一个任务指定一个节点的情况。相似的实验表明这种算法比只利用历史记录

的算法有更好的性能。到目前为止,有许多用于解决动态负载均衡的办法被提出来了,但大多数都是假设系统是稳定的,主要考虑 CPU 利用率<sup>[9]</sup>、CPU 队列长度<sup>[10]</sup>、资源队列长度<sup>[11]</sup>,以及预测资源需求<sup>[12]</sup>等,而缺少对异构集群系统调度策略的综合优化考虑,以及对于节点故障情况的处理。

文中拟提出一种针对异构 GPU 集群的自适应任务调度方法。该方法可以动态发现计算节点,并自动估计计算节点的计算能力。同时,可以根据节点计算能力、任务的计算强度和优先级在异构 GPU 集群上合理分配计算资源,并支持系统的可伸缩和计算节点的容错。

## 1 异构 GPU 集群任务调度策略的设计及其实现

对于现有大型集群系统主要分为两类:第一种就是节点之间可以直接通信的;第二种就是各节点是独立的,不能直接通信,节点的管理完全依靠调度器。对于这两种方式各有优缺点。第一种方式主要问题在于难协同各节点的同步,同时也不便于统一管理;而第二种方式主要瓶颈在于网络架构上。文中主要针对的是第二种,因此不必考虑通信问题。

系统中包含了两类节点:一个任务调度节点和多个计算节点。由于需要支持多任务和异构 GPU 平台,每个计算节点的任务处理能力和处理速度会有所不同,需要系统能根据计算资源的实时情况动态调度计算任务,以实现系统利用率的最大化。其中任务调度节点负责任务的分解和计算节点的管理,计算节点利用 GPU 卡完成计算任务。为了实现整个系统的动态

负载均衡,任务调度器的设计尤其重要。

### 1.1 任务调度模型

基于所调度的任务在相同机器上所需要的完成时间不同,给每个任务定义了“计算强度”的属性,来表示此任务在特定机器上需要的完成时间;同理,每台计算节点的单位时间内能完成的相同计算强度的任务数也是有所差异的,所以,给计算节点定义了“计算能力”属性,来表示此计算节点在单位时间内能完成的特定强度的任务数。

任务调度模型如图 1 所示,其中包含了  $M$  个计算任务和  $N$  个计算节点。其中每个计算任务都可以分解为很多可独立运行的规模较小的节点任务,具体分解出多少节点任务是由总计算能力决定的。任务调度器从计算任务中获得很多较小的节点任务,并将这些节点任务统一分配到不同的计算节点上。

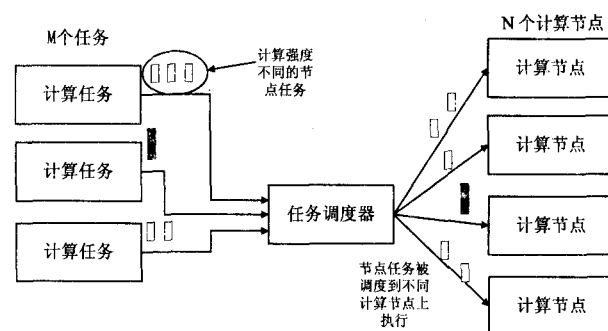


图1 任务调度模型

对于节点任务  $i (1 \leq i \leq M)$ , 定义  $S_i$  为此任务的计算强度(正比于任务的计算量),  $W_i$  为该任务的优先级权重,  $\eta_i = \frac{W_i}{\sum_{i=1}^M W_i}$  为此任务的资源占用比例。

系统中定义了四个优先级(紧急、高、中、低),分别对应为 5、3、2 和 1,表示在任务数目相等的情况下,这四种优先级任务占用计算资源的比例分别为 50%、30%、20% 和 10%。

对于计算节点  $j (1 \leq j \leq N)$ , 定义  $C_j$  为该节点的计算能力。参数  $C = \sum_{j=1}^N C_j$  表示  $N$  个计算节点的总计算能力。

### 1.2 任务调度器结构

任务调度器的主要功能包括四个方面:

- 1) 根据计算任务的计算强度和优先级,从计算任务中分解合适数目的节点任务加入到任务调度器中;
- 2) 根据计算节点的计算能力将节点任务分派到不同的计算节点上;
- 3) 计算节点计算能力的自动估计;
- 4) 计算节点的容错处理。

为了达到这样的设计目标,任务调度器采用了如图 2 所示的结构。

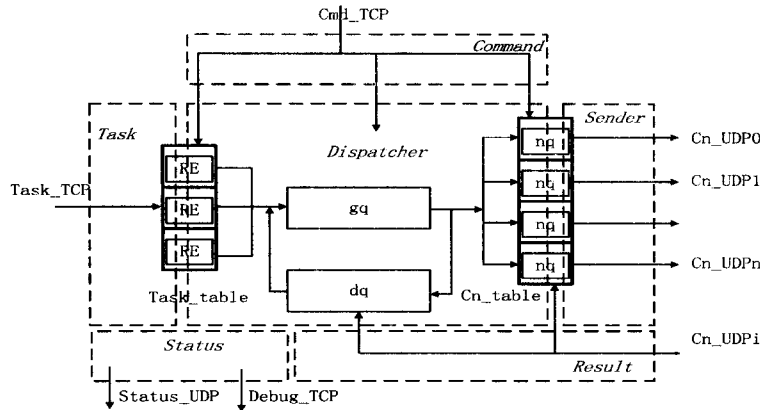


图2 任务调度器总体结构

任务节点中有两个核心数据结构:任务表 Task\_table 和计算节点表 cn\_table。其中任务表中记录了计算强度  $S$ 、优先级、完成情况等任务状态,计算节点表记录了各个计算节点的计算能力  $C$ 、节点任务队列  $nq$  等信息。

系统中包含了两个节点任务队列  $gq$  和  $dq$ 。其中  $gq$  为从计算任务中分解得到的节点任务队列; $dq$  为发送给计算节点但还没收到计算结果的节点任务队列。每个计算节点都有一个队列  $nq$ ,保存分配给该节点的节点任务。

系统中包含了 Task、Dispatcher、Sender、Result 和 Commander 等五个线程。其中 Task 线程监听 TCP 端口,等待用户端发送来的任务命令;Dispatcher 线程周期性地分解计算任务,将分解得到的节点任务放置到全局队列  $gq$  中,并同时放置到  $dq$  中,然后根据不同计算节点的速度将计算任务分配到不同计算节点的任务队列  $nq$  中;Sender 线程周期性从每个  $nq$  中读取计算任务,通过 UDP 端口轮转向各个端口发送节点任务;Result 线程周期性检查各个计算节点回送的结果报文,如果发现计算节点已经完成某个节点任务,则将此节点任务从  $dq$  中删除;Command 线程等待系统管理员发出的命令。

如果  $dq$  中的节点任务在预定时间内没有返回结果,则将它从  $dq$  重新放回  $gq$ ,重新进行调度。这样即使计算节点意外停机而没有完成分配的节点任务,通过这种重发机制可以将未完成的节点任务重新交由其他计算节点完成,从而实现了系统的容错性。

### 1.3 任务的分解与分派

调度线程 dispatcher 的主要任务是按照每个计算任务的计算强度和优先级分解若干节点任务到全局队列  $gq$ ,并按照各个计算节点的计算能力和节点任务的计算强度将  $gq$  中的节点任务分派到不同的计算节点队列  $nq$  上。

对于计算任务  $i$ ,应分解的节点任务数为:  $T_i + t_i =$

$\frac{\eta_i}{S_i}C$ 。其中  $\eta_i, S_i$  为该计算任务的资源占用比例和节点任务计算强度,  $C$  为总的计算能力。 $T_i$  为右式的整数部分,  $t_i$  为右式的小数部分。当  $t_i$  不等于零时,产生一个  $0 \sim 1$  的随机数  $\delta$ ,当  $\delta$  大于  $t_i$  时,  $T_i$  加一,否则  $T_i$  不变。之所以对右式的小数部分采用随机分配的方法,是因为在整体计算能力较低,且计算强度较大的情况下,右式可能始终小于 1,从而造成某些计算任务总是得不到分解。采用了随机方法之后,

可以保证大计算强度的任务仍然可以按照预定比例提交节点任务。

Dispatcher 线程将根据每个计算节点  $j$  的计算能力  $C_j$ ,从  $gq$  队列中选择若干个节点任务提交给对应的计算节点,对应的算法如下所示:

1. 设置  $C$  为节点的计算能力  $C_j$ ;
2. 如果  $gq$  为空,则返回;
3. 取  $gq$  中头节点的节点任务  $CT$ ,并获得其对应的计算强度  $S$ ;
4. 若  $C < S$ ,则
  - 4.1 将  $CT$  加入  $nq$ ;  $C = C - S$ ;
  - 4.2 转至 2;
5.  $p = (C/S)$  的小数部分,并取  $(0,1)$  区间中的随机数  $r$ ;
6. 若  $p > r$ ,则将  $CT$  加入  $nq$ ,否则保持  $CT$  在  $gq$  队列中。

该算法也使用了对于  $C/S$  的小数部分进行随机取任务的方法,这也是为了解决某些类型的节点任务计算强度大于计算节点的计算能力,而始终得不到分配的问题。

### 1.4 自适应调整计算能力及节点任务发送速率

由计算能力的定义可知,计算能力值是由计算节点返回到调度节点的结果报文数决定的,其真实值必须通过多次动态调整得到,这样得到的值更能真实地反映当前系统的计算能力。系统开始运行时将为每个计算节点  $j$  设置初始值  $C_j(0)$ 。在系统运行时刻,由 result 线程记录每个周期从特定计算节点中回送的计算结果,并由此统计出此周期  $T$  内计算节点  $j$  返回的节点任务的计算强度之和  $S_j(T)$ 。节点  $j$  在  $T$  周期内计算能力  $C_j(T)$  的调整方法为:

$$C_j(T) = \alpha C_j(T-1) + (1-\alpha) \frac{\sum_{t=T-L}^T S_j(t)}{L}$$

其中  $\alpha$  为  $(0,1)$  区间的小数,  $L$  为回溯周期数(一般设置为 64 至 256)。之所以采用一定的回溯周期,

是为了平滑网络等方面的不稳定性,可能在较长时间内节点返回的计算报文数很少,但是在短时间内突然返回大量计算结果,而造成计算能力估计的较大变动。

由于每个计算节点的计算能力有所不同,因此采用了一种自适应的方式来处理向节点的任务发送速度。

在计算节点初始化时将提供一个初始的 Speed 值。该参数值可以通过估算显卡的计算能力得到。

在系统运行时刻,通过两种方法来调整 Speed 的值。

1、result 线程记录了在每个周期中从特定计算节点中回送回来的计算节点报文数 Result\_num。根据这个值可以调整 speed 值。 $Speed = \alpha * speed + (1 - \alpha) * result\_num$ 。alpha 为[0,1]区间的某个数。

2、当计算节点的缓冲 dq 容量超过 3/4 时,将发送 Slowdown 报文,减少向该计算节点发送的任务;当该计算节点的缓冲容量低于 1/4 时,将发送 speedup 报文,提高该计算节点的发送速度;当计算节点的缓冲容量超过 7/8 时,将发送 Stop 报文,暂停向该节点发送报文;如果从在暂停状态下,报文恢复到 1/4 时,将发送 Resume 报文,恢复发送。

## 2 实验结果

测试用的异构 GPU 集群中包含了一台调度节点,4 台计算节点,配置如表 1 所示。

表 1 异构 GPU 集群的基本配置环境

	CPU	主存容量	GPU
调度节点	Intel i7 920	3GB	
计算节点 1	Intel i7 920	3GB	GTX 295
计算节点 2	Intel Q6600	3GB	GTX 295
计算节点 3	Intel Q6600	8GB	9800 GTX
计算节点 4	Intel i7 920	3GB	GTX 295

### (1) 计算效率。

针对同一计算任务,分别在单个计算节点和集群上计算,分别测试其运行时间,如表 2 所示。

表 2 单个节点与集群系统的执行时间(单位:秒)

	计算节点 1	计算节点 2	计算节点 3	计算节点 4	集群系统
运行时间	874.81	987.74	2922.2	952.5	286.15

系统达到理论峰值时的计算时间为  $\frac{1}{\sum \frac{1}{T_j}}$ , 其中

$T_j$  为每个节点的计算时间。由上表可以推算出达到理论峰值时的计算时间为 282 秒,可见整个集群系统的效率达到了 98.5%,充分地发挥了 GPU 集群的计算能力(补充一点,这里计算节点 1 和计算节点 2 两者采用

的 GPU 型号一样,但运行时间却相差很多,笔者做了多次实验,结果还是一样,可能是机器显卡配置问题,但这不会影响集群系统计算到的性能效率)。

### (2) 多个不同优先级同种类型任务的资源划分。

在集群环境下同时运行 4 个不同优先级(紧急,高,中,低)且计算强度相同的任务,得到的实验数据是:这 4 个不同优先级任务平均每周期内完成的节点任务数分别为 20,12,8,4,计算资源比例分别为 5:3:2:1。由此可以分析出,优先级越高的任务在单位时间内完成的节点任务越多,即获得的计算能力值高,且其计算能力分配比例基本与预定义的优先级比例一致。

### (3) 多个相同优先级不同类型任务。

在上述集群中同时运行相同优先级但计算强度分别为 1、1.3、28.2 的三个任务,得到的实验数据是:其在平均每周期内完成的平均节点任务数分别为 14.4, 10.8, 0.5,平均每周期内的计算资源分别为 14.4, 14.04, 14.1,计算资源比例分别为 1:0.975:0.979(在一个周期内,每个任务平均获得的计算资源等于单个节点任务的计算强度×完成的节点任务数)。由此可以看出,对于各种不同任务强度的任务分派都是均衡的。

### (4) 自适应计算能力。

计算节点 1 和计算节点 2 的计算能力基本相近。在实验中,初始设置两个计算节点的计算能力分别为 20 和 0。每十二个周期取样一次,其计算能力的自适应调节过程如图 3 所示。

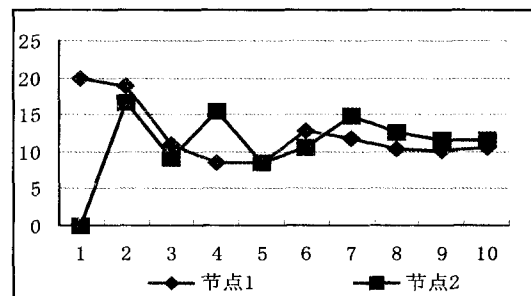


图 3 自适应调节计算能力情况

从上图可以看出,经过 200 秒后,两个计算节点的计算能力会收敛稳定,趋向一个稳定的值。

### (5) 可扩展能力。

在天河 1 号 A 超级计算机系统上使用 256 个节点进行了四种类型任务的计算,以测试系统的可扩展能力,其测试结果如表 3。

表 3 系统可扩展能力

任务类型	任务 1	任务 2	任务 3	任务 4
在单个节点上的计算速度	0.818	1.123	0.00078	0.56
在 256 个节点上的计算速度	193	252	0.19	136
加速比	236	225	245	243
计算效率	92%	88%	96%	95%

从表 3 中可以看出,系统在计算节点数量增加时,计算性能也能随之同步增加,系统整体运行效率处于 88% 到 96% 之间。

### 3 结束语

由上面的分析可知,运行效果达到了预期的目的,解决了异构 GPU 集群系统的任务调度问题,实现了整个系统的动态负载均衡,提高了整个系统的计算效率,并初步形成了一套支持多优先级任务调度、支持异构 GPU 节点、调度效率高、具有节点容错能力、可扩展能力强的动态调度机制,为将来大规模计算密集型任务的调度提供了解决方案。

#### 参考文献:

- [1] Buyya R. 高性能集群计算:结构与系统[M]. 郑伟民,石威,译. 北京:电子工业出版社,2001.
- [2] 陈华平,黄刘生,安虹,等. 并行分布计算中的任务调度及其分类[J]. 计算机科学,2001,28(1):45-48.
- [3] 李丙锋,祝永志,魏榕晖. 异构 Beowulf 系统负载均衡技术的研究与实现[J]. 计算机技术与发展,2008,18(7):60-65.
- [4] 徐群,祝永志. 集群系统中的负载均衡问题的研究[J]. 计算机技术与发展,2009,19(8):129-132.
- [5] 陈志刚,曾志文. 中间应用服务器动态负载均衡的物理模

(上接第 31 页)

中用于界面控制的代码,在这里是指 Aspx 后台代码,这部分代码主要是通过 IDE 工具自动生成的。可以看到,在代码量上,最终代码约为三万八千行,而生成代码两万七千行,生成的代码占大部分,有效减少了手工编写的代码的数量,提高了系统的开发效率。

### 4 结束语

文中提出一种 MIS 系统快速开发策略,即通用管理信息系统框架+自动代码生成器的架构,适于中小企业信息化管理系统的开发,可以大幅度地减少手工代码编写数量,有效缩短系统的开发周期,降低系统的开发成本。系统架构未来的改进,需要扩充的功能包括:表示层支持、开发过程支持和多表关联支持。

#### 参考文献:

- [1] 宋翔宇,曾雅琳. 一种新的代码生成器的设计与实现[J]. 计算机科学,2011,38(7A):67-69.
- [2] 陆远,胡莹. .NET 平台下敏捷开发架构及代码生成技术[J]. 微计算机信息,2009,25(11-3):11-12.
- [3] Turner M S V. Microsoft Solutions Framework Essentials Building Successful Technology Solutions[R/OL]. 2009. <http://www.docin.com/p-132599666.html>.

型[J]. 计算机工程,2001(1):44-46.

- [6] 王霜,修保新,肖卫东. Web 服务器集群的负载均衡算法研究[J]. 计算机工程与应用,2004(25):78-80.
- [7] Werstein P, Situ H, Huang Zhiyi. Load Balancing in a Cluster Computer[C]//Proceedings of the Seventh International Conference on Parallel and Distributed Computing Applications and Technologies. [s. l.]:[s. n.], 2006.
- [8] Chi M, Yu Jung-Lok, Kim Ho-Joong, et al. Improving Performance of a Dynamic Load Balancing System by Using Number of Effective Tasks[C]//IEEE International Conference on Cluster Computing Proceedings 2003. [s. l.]:[s. n.], 2003: 436-441.
- [9] Tanenbaum A S. Distributed Operating Systems[M]. Englewood Cliffs, New Jersey: Prentice-Hall, 1995.
- [10] Kunz T. The influence of different workload descriptions on a heuristic load balancing scheme[J]. IEEE Transactions on Software Engineering, 1991, 17(7):725-730.
- [11] Ferrari D, Zhou S. An empirical investigation of load indices for load balancing applications[M]//Scheduling and Load Balancing in Parallel and Distributed Systems. Los Alamitos, California: IEEE Computer Society Press, 1995:487-496.
- [12] Devarakonda M V, Iyer R K. Predictability of process resource usage: a measurement-based study on UNIX[J]. IEEE Transactions on Software Engineering, 1989, 15(12):1579-1586.
- [4] 张晨. .NET 环境下基本业务系统生成平台的设计与实现[D]. 西安:西安电子科技大学,2008.
- [5] 宋明,唐虹. 面向中小型 MIS 的数据库操作池的研究[J]. 计算机工程与设计,2005,26(9):2519-2521.
- [6] 张静,孔芳. 一个数据模型驱动的代码生成工具的设计与实现[J]. 计算机应用与软件,2010,27(11):151-153.
- [7] 杨学增,王满敬. 基于 B/S 三层架构的 ASP.NET 系统的代码生成研究[J]. 软件导刊,2009,8(7):11-13.
- [8] 张志杰. 基于分层结构的管理信息系统架构设计[J]. 计算机技术与发展,2010,20(10):146-153.
- [9] 白尚旺. PowerDesigner 软件分析设计技术[M]. 北京:电子工业出版社,2002:79-132.
- [10] Yao Leiyue, Chen Yong. An Optimizing Strategy for Massive Data Management System Based on SQLSERVER 2000[C]//Proceedings of 2009 Asia-Pacific Conference on Information Processing( Volume 1). [s. l.]:[s. n.], 2009.
- [11] 李娟娟,李龙澍. 基于 Web 的 MIS 开发中动态报表的实现[J]. 计算机技术与发展,2008,18(8):179-181.
- [12] Weigert T, Weil F, van den Berg A, et al. Automated Code Generation for Industrial-strength Systems[C]//32nd Annual IEEE International Conference on Computer Software and Applications, COMPSAC '08. [s. l.]:[s. n.], 2008:464-472.