

基于无共享架构的海量感知数据实时处理系统

赵卓峰¹, 魏文飞², 马 强³

(1 北方工业大学 云计算研究中心, 北京 100041; 2 福州市勘测院, 福建 福州 350003;

3 中国科学院 计算技术研究所, 北京 100190)

摘 要: 为了满足具有海量性、连续性及不确定性的感知数据实时处理需求, 采用云计算典型的分布式无共享集群架构, 文中提出一种并行化的海量感知数据实时处理模型, 并给出了相应的编程接口. 在此基础上设计了一种去中心化的分布式感知数据实时处理系统架构以及基于 ZooKeeper 的集群伸缩管理方案, 从而保证了感知数据处理系统的实时性及扩展性. 通过一个结合城市车辆监管实际应用的实验, 验证了该系统在负载均衡的情况下, 其处理性能能够随着计算节点的增加而接近线性增长.

关键词: 感知数据; 实时处理; 云计算; 智能交通

中图分类号: TP319

文献标识码: A

文章编号: 1000-7180(2012)09-0009-06

A Real-Time Processing System for Massive Sensing Data

ZHAO Zhuo-feng¹, WEI Wen-fei², MA Qiang³

(1 Research Center for Cloud Computing, North China University of Technology, Beijing 100041, China;

2 Fuzhou Investigation and Surveying Institute, Fuzhou 350003, China;

3 Institute of Computing Technology, CAS, Beijing 100190, China)

Abstract: With the development of Internet of Things, the sensing data gathered by large amounts of sensors shows the massive, continuous and probabilistic characteristics. In order to satisfy the requirements of real-time processing of such sensing data, the share-nothing architecture of cloud computing is adopted to support the parallel sensing data processing with extendibility, and a parallel computing model and corresponding programming interface for real-time sensing data processing is proposed by extending the MapReduce. Furthermore, a decentralized distributed architecture and cluster management method are designed to implement the parallel computing model. The experiment shows that the system has good scalability and the processing performance increases in linear progression as the number of server increases.

Key words: large data; real-time processing; cloud computing; intelligent transportation

1 引言

随着物联网技术及相关建设的不断发展, 通过各种传感设备实时采集并集中汇交的感知数据逐渐成为信息化系统中一类新的、关键的数据形式, 同时基于感知数据提供更加精确、全面、智能的信息服务也成为当前以智慧城市为代表的数字城管、智能交通等信息化应用中的研发热点^[1].

近年来, 很多数据流系统被设计来满足持续获得数据的处理需求, 如 STREAM^[2]、Borealis^[3] 等. 由于数据流的持续性和无限性, 有限的处理机不可能处理数据流的完整信息, 因而采用窗口机制(时间、数据量)来划定处理边界, 窗口边界范围内已积累的数据称之为历史数据, 现有的数据流处理系统受数据采集速度、传输带宽和内存容量等因素的限制, 侧重于针对一个相对小规模的历史数据进行, 它

们不得不面对可靠性、扩展性和伸缩性方面的问题。S4^[4]是基于分布式可伸缩架构实现的两个典型数据流系统,它们分别采用流水线和分布处理单元技术提高数据流系统的处理能力,但 S4 需要在节点间传输大量中间结果,HOP 需要完成多个节点结果的同步合并,都没有充分利用多核计算资源的本地化能力。以 MapReduce^[5]为代表的云计算技术为基于可伸缩架构的大规模数据处理带来了新的思路,其强调在大量低端通用服务器(Commodity hardware)集群的无共享架构(Shared-nothing architecture)上建立面向海量数据处理的并行计算模型和可伸缩环境,并在以搜索为主的大规模互联网数据处理等应用中得到了良好的验证。但现有的典型 MapReduce 实现(如 Hadoop^[6])属于对持久化数据的批处理方式,在每次处理时,都需要初始化运行环境,同步的在 Map 和 Reduce 阶段载入、处理大规模数据,并在节点间传输大量数据。按照这种批处理方式处理具有前文所述特征的感知数据,若每次处理小规模批的数据,则系统开销太大,实时性受到限制,若等待批达到一定规模则又增加了处理延迟,因此也无法直接满足感知数据实时处理需求^[7]。

针对现有技术的上述不足,本文针对海量感知数据的实时处理需求,采取基于并行化提高感知数据处理性能以及基于无共享集群架构保障系统伸缩的技术思路,提出一种并行化的海量感知数据实时处理模型并给出了相应的编程接口,并在此基础上设计了一种去中心化的分布式感知数据实时处理系统架构以及基于 ZooKeeper^[8]的集群伸缩管理方案,下面章节将分别对它们进行详细介绍。

2 并行化的海量感知数据实时处理模型与编程接口

2.1 并行处理模型

由于感知数据主要表现为一组时间序列下针对感知对象相关状态属性的数据,感知数据间往往具有无相互依赖性的特征。因此,为了提高感知数据的实时处理性能,我们采用数据并行的思路,设计了如图 1 所示面向感知数据实时处理的单控制流多数据流(single process and multiple data, SPMD)并行模式^[9]。

在图 1 中,感知数据可以根据感知对象的属性特征及处理需求进行并行化划分,如道路车辆实时感知数据可以根据车牌号或车辆位置所属区域等进行并行化划分,经并行化划分的数据被分配到不同

的并行处理节点,并行处理节点按照预定义的计算任务对感知数据进行处理,各并行处理节点的中间处理结果汇交到相应节点进行合并,形成最终的处理结果。

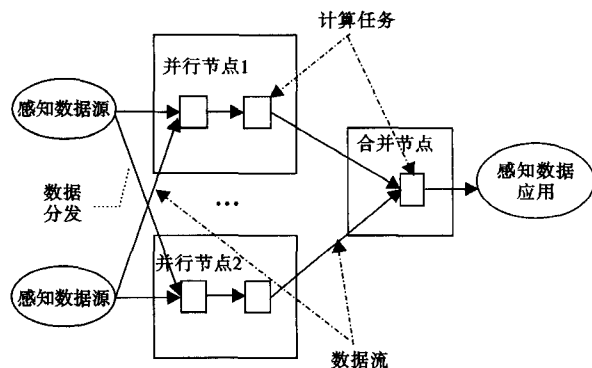


图 1 并行化的海量感知数据实时处理模型示意图

下面给出上述并行化的海量感知数据实时处理模型中的几个核心概念的定义:

定义 1 感知数据对象 DO(Data Object)是表示特定感知对象在特定时刻状态属性的数据对象。

定义 2 感知数据流 DS(Data Stream)表示持续不断获得的感知数据 DO 序列。

定义 3 感知历史数据 HD(History Data)表示历史上积累的感知数据 DO 序列。

这里,可以指定 DO 中的某个属性为 Key,以作为对 DS 及 HD 中感知数据进行并行化划分的依据。在收到属于某个感知数据流的数据及需要提取某些感知历史数据时,将对 DO 中作为 Key 的属性的值进行哈希,然后根据 Key 的哈希值进行数据的分发、缓存和后续的并行处理。关于感知数据流及感知历史数据的缓存结构及模式可参考文献^[10]

定义 4 计算任务 CT(Computing Task)是感知数据并行处理中最基本的数据处理单元,每个 CT 可以按照预定规则接收 DO 中 Key 为特定哈希值的 DS 及 HD 数据,然后进行处理,最后可能将处理结果发送给其它计算任务或者将结果输出到结果合并节点。

因为感知数据会根据 Key 的哈希值被划分成多个部分,所以每个计算任务都有多个实例,每个实例处理其中的部分感知数据。

定义 5 感知数据实时处理应用 RA(Real-time Application)是由一系列计算任务 CT 组成的序列集合,每个计算任务都会有多个实例。

如图 1 所示,计算任务 CT 可以放在不同的节点上,按照感知数据流向构成一个有向无环图。

2.2 编程接口

为了支持上述并行化的海量感知数据实时处理模型下的应用编程,需要提供简单、有效的编程接口,一方面支持用户根据业务处理需求自定义算子,另一方面并且只需编写数据处理的串行逻辑,有系统来负责并行计算的调度。

针对上述目标,在并行化的海量感知数据实时处理中主要编程工作集中于计算任务 CT 方面。在这里,我们允许开发人员结合业务需求编写自己的计算任务 CT,并且只需要在 CT 中实现串行的数据处理逻辑。每个 CT 在系统中都是一个可以动态加载的模块,可以将多个 CT 放入到一个计算节点当中,进行统一管理。

开发人员在编写 CT 时,需要继承一个抽象类 AbstractCT,并且必须要实现其中的 execute 方法,execute 方法中就是该 CT 具体的数据处理逻辑。不仅如此,开发者还可以在 CT 中添加必要的成员变量。如果开发人员需要在 CT 实例化时来加载数据或者做其它工作,可以在 CT 的无参数构造函数中进行编写。

表 1 给出了计算任务 CT 编程需要实现的主要编程接口,其中 execute 是开发人员必须要实现的方法,当 CT 接受到数据以后,将会调用该方法来进行数据处理;load 是加载业务计算所需的感知历史数据及其它相关数据的方法;checkpoint 和 restoreStates 方法是当感知数据处理应用为状态敏感时,用来进行数据备份和恢复的方法。

表 1 感知数据并行处理计算任务主要编程接口

方法名称	功能
execute	执行 CT 的数据处理逻辑
load	加载感知历史数据及其它数据
checkpoint	对 CT 中的指定的内存数据进行 checkpoint
restoreStates	加载 CT 最近一次 checkpoint 后的内存状态

3 去中心化的分布式感知数据实时处理系统架构

如前所述,如何提供一个可灵活扩展、伸缩的分布式处理架构是支撑上述并行处理模型的关键。为解决该问题,我们设计了一种去中心化的分布式感知数据实时处理系统架构。

如图 2 所示,在该架构中共有两类节点,路由节点 RN(Routing Node)和计算节点 CN(Computing

Node)。其中路由节点负责感知数据流向的路由工作,而计算节点是计算任务 CT 的运行容器,负责管理运行在该节点上的计算任务,为计算任务提供运行环境以及接收和发送数据。上述节点间采用一种基于 ZooKeeper 的分布式集群协调服务进行管理(第 4 节将对此进行详细介绍),从而可以去除分布式架构中的中心化特征。

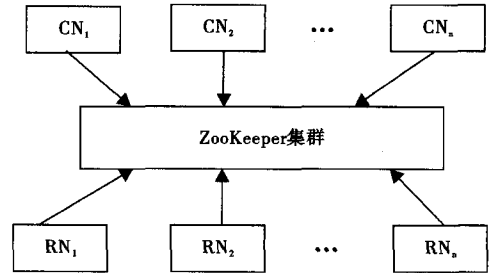


图 2 分布式感知数据实时处理系统架构

3.1 路由节点

路由节点 RN 的作用是决定计算节点该执行哪些计算任务以及计算任务应该接受感知数据中的哪部分数据。如上节所描述的那样,开发人员在编写感知数据处理应用时,会预先在配置文件中定义感知数据流、感知历史数据、计算任务以及它们之间的关系。当数据源或者计算节点在发送数据时,首先会根据数据所属的流和它作为 Key 的属性的 hash 值在自己的内部路由表查询,如果曾经发送过相同类型的数据,路由表中肯定存在一条记录,该记录中记载着接收该类型数据的计算任务所在计算节点的地址信息,数据源或计算节点就直接向目标计算节点发送数据。如果在本地路由表中没有查到该类型数据对应的计算任务,那么数据源或计算节点将会在路由节点集群中任意选择一个路由节点发送路由查询命令。路由节点收到路由查询命令后,会查询内部的路由表,如果存在路由记录,直接将路由信息返回给数据源或者计算节点,如果不存在,则说明该数据类型是第一次出现在系统当中,路由节点会根据数据类型以及集群节点信息分配一个计算节点 CN,并将该计算节点 CN 的地址信息返回给数据源或者计算节点。

所有数据源和计算节点 CN 只有在第一次发送某个流的数据时才会向路由节点发送路由命令,所以路由节点的负载非常小。

路由节点在启动后,会将自己加入到路由节点集群,并且会获取计算节点集群中所有计算节点的信息,当有计算节点加入或退出时,会立即更新自己

内部的信息. 路由节点集群中的所有路由节点的路由信息是一致的, 所以数据源或客户端在发送路由查询命令时, 任意选择一个路由节点即可.

3.2 计算节点

计算节点 CN 的功能是管理它所在节点上的所有计算任务, 作为这些计算任务的容器, 为它们提供运行环境, 以及接受和发送数据.

CN 在启动时, 首先读取感知实时数据处理应用的配置文件, 然后加载感知数据流和感知历史数据的定义、计算任务定义以及它们之间关系, 然后将自己加入到计算节点集群当中, 并且获取路由集群中所有路由节点的信息.

计算任务 CT 并不是在计算节点启动以后就被加载, 而是当 CT 第一次接收到数据时才会被实例化. 计算节点为它所管理的所有计算任务接受数据, 当计算节点收到由数据源或其它计算节点发送过来的数据后, 根据数据所属流的名称和 Key 的 hash 值, 将数据分发给相应的计算任务.

计算节点在接收数据的同时也为它所管理的所有计算任务发送数据, 在发送数据时, 如果应该处理该数据的下游计算任务也在该计算节点内, 计算节点将会直接将数据分发给相应的下游计算任务. 否则就如路由节点介绍中所描述的过程发送数据.

如果集群中某个计算节点出现异常, 那么向该计算节点发送数据的数据源或者其它计算节点将会向路由节点发送路由查询命令, 路由节点将会重新分配一个正常的计算节点接替异常节点, 异常节点上的所有计算任务将会在新的计算节点上启动, 保证了系统能够连续不断的正常运行.

4 基于 ZooKeeper 的集群伸缩管理方案

如上节所述, 海量感知数据实时处理系统是一种分布式系统, 并且采用的是去中心化的分布式结构. 去中心化的分布式结构虽然很容易扩展, 并且不会出现单点失效的问题, 但是它所面临的最大的问题是如何正确、高效的进行集群管理. 因为没有控制中心, 所以要实现集群管理相比于有控制中心结构负载很多. 为此, 需要解决以下两个问题:

(1) 如何在运行时向集群中添加和删除节点.

(2) 如何能够让集群中的每个节点快速、准确的了解到一致的集群信息.

为了解决上述问题, 本文采用 ZooKeeper 作为系统的集群管理解决方案. ZooKeeper 是开源项目 Hadoop 下的子项目, 是一款分布式系统协调器. 无

中心的分布式结构最为困难的地方就在于集群管理, 而 ZooKeeper 正为此提供了一套简单、高效、稳定的解决方案. ZooKeeper 服务可以由 1 台或多台服务器提供, ZooKeeper 保证只要集群中有一半以上的服务器正常, ZooKeeper 服务就能正常提供服务. ZooKeeper 使用目录树的型式存储信息. 在目录树中每个节点可以存储 1M 的数据, 目录树的节点分为两种类型: EPHEMERAL 和 PERSISTENT. EPHEMERAL 节点的特点是当创建该节点的客户端失效后, 该节点也会消失, 利用这一性质可以实现计算节点的加入和退出. PERSISTENT 节点与 EPHEMERAL 节点正好相反, 不管创建它的客户端是否失效, 它都会一直存在, 直到被删除, 利用这类节点可以进行集群信息的存储和共享.

系统中的路由节点和计算节点在加入群时, 都向 ZooKeeper 进行注册, 这样已经在集群中的其它节点就能够知道由新的节点加入, 新加入的节点也能够通过 ZooKeeper 中的节点信息, 获取现在集群中的所有节点的信息.

如图 3 所示, 系统中的 ZooKeeper 目录结构中两个 PERSISTENT 节点: /cns 和 /rns, 这两个目录节点分别是计算节点和路由节点的挂靠点. 计算节点在加入集群并向 ZooKeeper 注册自己时, 会在 /cns 节点下创建 EPHEMERAL 目录节点, 该目录节点保存创建该节点的计算节点的 IP 地址和端口号等信息. 路由节点则会在 /rns 节点下创建 EPHEMERAL 目录节点, 该目录节点同样保存的是创建该节点的路由节点的 IP 地址和端口号等信息.

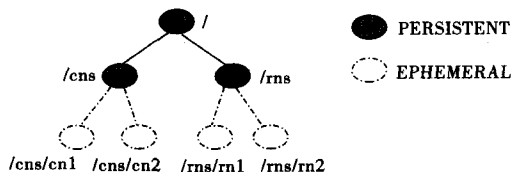


图 3 感知数据处理系统中 ZooKeeper 的目录结构

每个计算节点都向 ZooKeeper 注册监听 /rns 节点下是否有节点变化的事件, 而每个路由节点则向 ZooKeeper 注册监听 /cns 节点下是否有节点变化的事件. 当路由节点或计算节点加入或退出集群中时, 其它节点就会收到相应的通知.

5 实验与评价

为了验证海量感知数据实时处理系统的效果, 结合一个实际的城市车辆实时监管系统中的套牌车自动甄别应用进行了相应的实验. 套牌车自动甄别

应用是车辆实时监管系统中的一项最基本且重要的检测应用. 所谓套牌车是指两辆不同的车使用一个相同的车牌号码. 套牌车自动甄别的检测原理是利用车辆的时空矛盾性质自动判断套牌车, 即同一车牌号的车辆不可能在一定的时间阈值范围内出现在两个不同的监测点, 若出现则说明两个车辆中有一个具有套牌车的嫌疑.

按照本文提出的并行处理模型, 基于实时感知的道路车辆数据流和一定时间阈值范围内历史车辆数据的套牌车计算模型如图4所示. 其中 TaoPaiCT 是开发用户需要按照上述套牌车处理逻辑编写的计算任务. 由于篇幅关系, 在这里我们不再给出其具体的伪码描述.

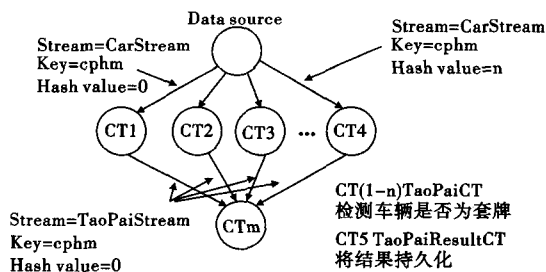


图4 套牌车并行计算模型示例

本实验所使用的测试数据是在某城市一天真实道路车辆监测数据(100万条)基础上通过模拟程序生成的道路车辆数据(1亿条), 这样能够保证节点及计算任务负载的均衡.

具体实验方法如下:

① 搭建只有一台计算节点的车辆感知数据实时处理系统, 部署运行套牌车检测应用, 并且在计算节点上启动4个TaoPaiCT实例, 即将车辆监测数据平均分成4部分, 每个实例处理四分之一的数据.

② 使用测试程序以不同速率向系统发送车辆监测数据, 并记录系统在各种速率下吞吐率. 共进行5次该过程, 最后的吞吐率指标取5次实验的平均值.

③ 在StreamX在新添加的计算节点上也启动4个TaoPaiCT实例, 现在共启动了8个实例, 即将车辆监测数据平均分成了8部分, 每个实例处理八分之一的数据.

④ 再次对现在的系统进行步骤②.

⑤ 再向系统添加一台计算节点, 并且在新添加的计算节点上同样启动4个TaoPaiCT实例, 现在共启动了12个实例, 即测试数据平均分成了12部分, 每个实例处理十二分之一的数据.

⑥ 再次对系统进行步骤②.

⑦ 比较分析系统在计算节点数目变化前后的性能变化情况.

在实验环境方面, 测试客户端使用1台配置为Intel P8700 2.53GHz 双核处理器、3G内存、250G硬盘的服务器, 计算节点和路由节点共使用了3台配置为Xeon 2.66GHz 8核处理器、16G内存、300G硬盘的服务器. 所有服务器的操作系统均为Linux CentOS 5.4 并且通过带宽为1 Gbps的局域网相连.

如图5所示, 当系统中只有一台计算节点时, 系统的处理性能在数据流的速度为10000条/秒时最好, 吞吐率接近10000条/秒. 但是当数据流的速度继续增大时, 系统的吞吐率则开始下降. 造成上述情况的原因是: 当数据流速度增加时, 同一时刻, 相比于速度较低的数据流, 每个TaoPaiCT的缓存数据将会增多, 这也就增加了每条数据的计算量, 当数据流速度到达10000条/秒时, 单台计算节点的处理性能达到了极限. 当系统中有两台计算节点时, 增加了4个TaoPaiCT实例, 因为每个实例的数据负载比较均衡, 所以在相同的数据流速度情况下, 每个实例中的缓存数据是仅使用一台计算节点时的一半, 当数据流的速度到达20000条/秒时, 系统的处理性能达到极限, 如果数据流的速度继续变大, 将会遇到和使用一台计算节点时同样的问题, 系统的处理性能将开始下降. 当使用3台计算节点时, 共有12个TaoPaiCT实例, 所以现在每个实例中的缓存数据量是一台计算节点时的三分之一, 所以在数据流速度到达30000条/秒时, 系统的性能达到极限.

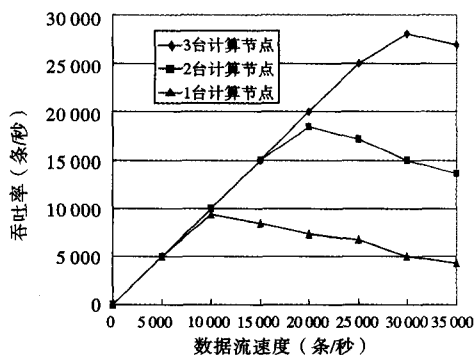


图5 系统扩展性测试结果

综上所述, 本实验表明在计算节点和计算任务的数据负载比较均衡的情况下, 海量感知数据实时处理系统的处理性能能够随着计算节点的增加而接近线性增长.

6 结束语

物联网感知数据正逐渐成为新兴信息应用中一

类典型且关键的数据,其一方面存在着海量、持续、不确定等特征,另一方面在物联网应用中对其提出了多样化、实时的处理需求.为了满足具有上述特性的感知数据实时处理需求,本文采用云计算典型的分布式无共享集群架构,并对现有批处理方式的 MapReduce 计算模型进行扩展,提出了一种并行化的海量感知数据实时处理模型,同时还给出了相应的编程接口.进一步,在此基础上设计了一种去中心化的分布式感知数据实时处理系统架构以及基于 ZooKeeper 的集群伸缩管理方案,以支撑了感知数据处理系统的实时性及扩展性.

参考文献:

- [1] 王珊,王会举,覃熊派,等. 架构大数据: 挑战、现状与展望[J]. 计算机学报, 海量数据处理专刊, 2011, 34(10): 1741-1752.
- [2] The Stream Group. STREAM: the stanford stream data manager[J]. IEEE Data Engineering Bulletin, 2003, 26(1): 19-26.
- [3] Abadi D J, Ahmad Y, Balazinska M, et al. The design of the borealis stream processing engine[C] // The 2nd Biennial Conf. on Innovative Data Systems Research(CIDR'05). Asilomar, CA, 2005.
- [4] Neumeyer L, Robbins L, Nair A, et al. S4: Distributed stream computing platform[C] // Proceedings of the 10th IEEE International Conference on Data Mining Workshops (ICDMW 2010). Sydney, Australia, 2010: 170-177.
- [5] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. ACM Communication, 2008, 51(1): 107-113.
- [6] Apache. Hadoop V1.0.0 [EB/OL]. (2011-11-10) [2011-12-15]. <http://hadoop.apache.org/>.
- [7] 倪明选, 罗吴蔓. 数据爆炸时代的技术变革[J]. 中国计算机学会通讯, 数据密集型计算专题, 2011, 7(7): 12-20.
- [8] Apache, ZooKeeper [EB/OL]. (2010-01-15) [2012-02-20]. <http://ZooKeeper.apache.org/>.
- [9] 朱定局. 并行时空模型[M]. 北京: 科学出版社, 2009, 10.
- [10] 亓开元, 赵卓峰, 房俊, 等. 针对高速数据流的大规模数据实时处理方法[J]. 计算机学报, 2012, 35(3): 477-490.

作者简介:



赵卓峰 男, (1977—), 博士, 副研究员. 研究方向为云计算、感知数据处理、服务计算

魏文飞 男, (1975—). 研究方向为数字城市、智慧城市、GIS.

马强 男, (1986—), 硕士研究生. 研究方向为数据流处理、大规模数据处理

(上接第 8 页)

- [5] 尹超, 黄必清, 刘飞, 等. 中小企业云制造服务平台共性关键技术体系[J]. 计算机集成制造系统, 2011, 17(3): 495-503.
- [6] 范文慧, 肖田元. 基于联邦模式的云制造集成体系架构[J]. 计算机集成制造系统, 2011, 17(3): 469-476.
- [7] 李春泉, 尚玉玲, 胡春杨. 云制造的体系结构及其关键技术研究[J]. 组合机床与自动化加工技术, 2011(7): 104-107.
- [8] 任磊, 张霖, 张雅彬, 等. 云制造资源虚拟化研究[J]. 计算机集成制造系统, 2011, 17(3): 511-518.
- [9] 赵宏, 杨愚鲁. 基于竞标机制的网格资源管理模型的模拟[J]. 计算机工程, 2007, 33(4): 85-87.
- [10] 杨帆, 刘士荣. 基于竞标机制的多移动机器人实时编队控制[J]. 华东理工大学学报·自然科学版, 2010, 36(1): 113-120.

作者简介:



魏乐 男, (1979—), 硕士, 讲师. 研究方向为企业信息化、服务计算