面向物联网海量传感器采样数据管理的数据库集群系统框架

丁治明[1)] 高 需[1),2)]

[1)](中国科学院软件研究所基础软件国家工程研究中心 北京 100190)

[2)](中国科学院研究生院 北京 100049)

摘 要 物联网是目前国际和国内新兴的一项热门技术,正在给人们的生产和生活方式带来深刻的变革.物联网在带来诸多好处的同时,也给软件乃至整个信息技术领域带来了前所未有的挑战.该文针对物联网传感器采样数据管理中所面临的数据海量性、异构性、时空敏感性、动态流式特性等问题,提出一种面向物联网海量传感器采样数据管理的数据库集群系统框架IoT-ClusterDB.实验结果表明,IoT-ClusterDB具有良好的传感器数据接入与查询处理性能,为物联网海量异构传感器采样数据的存储与查询处理提供了一种可行的解决方案.

关键词 物联网;传感器;时空数据;海量数据管理;数据库集群系统

中图法分类号 TP311 **DOI号**: 10.3724/SP.J.1016.2012.01175

A Database Cluster System Framework for Managing Massive Sensor Sampling Data in the Internet of Things

DING Zhi-Ming[1)] GAO Xu[1),2)]

[1)](*National Fundamental Software Research Center, Institute of Software, Chinese Academy of Sciences, Beijing* 100190)

[2)](*Graduate University of Chinese Academy of Sciences, Beijing* 100049)

Abstract In recent years, the Internet of Things (IoT) has become increasingly important and is changing the way how people live and work. IoT has a lot of benefits and meanwhile, it also brings about great challenges to the software and the whole IT community. In this paper, we mainly focus on the challenges in IoT data management. In IoT systems, the data sampled from sensors are massive and heterogeneous. Besides, they are spatial-temporal and dynamically changing stream data. To meet these challenges, we propose an IoT Database Cluster System Framework for Managing Massive Sensor Sampling Data (IoT-ClusterDB) in this paper. The experimental results show that IoT-ClusterDB has satisfactory sensor data uploading and query processing performances and thus provides a good solution for managing and querying massive sensor data in the Internet of Things.

Keywords Internet of Things; sensor; spatial-temporal data; massive data management; database cluster system

1 引 言

物联网(The Internet of Things,IoT)的概念是在1999年提出的[1-2],简单来讲就是把各类物品通过射频识别(RFID)、传感器件与设备、全球定位系统等种种装置与互联网结合起来而形成一个巨大的网络,实现智能化的识别与管理,进而实现各类物品

收稿日期:2011-08-26;最终修改稿收到日期:2012-04-12.本课题得到国家自然科学基金重大研究计划·重点支持项目"面向非常规突发事件主动感知与应急指挥的物联网技术与系统"(91124001)资助.**丁治明**,男,1966年生,博士,研究员,博士生导师,主要研究领域为数据库与知识库系统、时态与空间数据库、物联网与云计算数据管理.E-mail:zhiming@iscas.ac.cn.**高 需**,男,1980年生,博士研究生,讲师,主要研究方向为数据库与知识库系统、时态与空间数据库.

的远程感知和控制，由此生成一个更加智慧的生产和生活体系.

目前，物联网已经成为一个新的技术热点，得到了世界各国研究者和产业界的广泛关注.然而，物联网的研究与产业化还存在着诸多局限，大部分的工作还集中在物联网中单个传感器或小型传感器网络方面(如智能传感器技术、压缩传感技术等)，或者集中在物联网硬件和网络层面(如新型网络互连技术、高通量服务器技术等)[3-7]，而对于物联网欲与互联网相比肩所面临的核心问题，即海量异构传感器数据的存储与查询处理、大量传感器的智能分析与协同工作、复杂事件的自动探测与有效应对等技术的研究还比较有限.

在物联网系统中，对传感器海量采样数据的集中存储与查询处理是十分重要的.通过对海量传感器采样数据的集中管理，用户不仅可以直接在数据中心获得任一传感器的历史与当前状态，而且通过对集中存放的群体数据进行分析，可以实现复杂事件与规律的感知.此外，传感器采样数据的集中管理还使得物物互联(Web of Things)、基于物的搜索引擎、传感器采样数据的统计分析与数据挖掘等成为可能.

在传感器采样数据的集中管理系统中，大量的传感器结点根据预先制定的采样及传输规则，不断地向数据中心传递所采集的数据，从而形成海量的异构数据流.数据中心不仅需要正确地理解这些数据，而且需要及时地分析和处理这些数据，进而实现有效的感知和控制.通过分析我们不难看出，物联网的以下4个特点对数据处理技术形成了巨大的挑战：

(1)物联网数据的海量性.物联网系统通常包含着海量的传感器结点.其中，大部分传感器(如温度传感器、GPS传感器、压力传感器等)的采样数据是数值型的，但也有许多传感器的采样值是多媒体数据(如交通摄像头视频数据、音频传感器采样数据、遥感成像数据等).每一个传感器均频繁地产生新的采样数据，系统不仅需要存储这些采样数据的最新版本，而且在多数情况下，还需要存储某个时间段(如1个月)内所有的历史采样值，以满足溯源处理和复杂数据分析的需要.可以想象，上述数据是海量的，对它们的存储、传输、查询以及分析处理将是一个前所未有的挑战.

(2)传感器结点及采样数据的异构性.在同一个物联网系统中，可以包含形形色色的传感器，如交通类传感器、水文类传感器、地质类传感器、气象类传感器、生物医学类传感器等，其中每一类传感器又包括诸多具体的传感器，如交通类传感器可以细分为GPS传感器、RFID传感器、车牌识别传感器、电子照相身份识别传感器、交通流量传感器(红外、线圈、光学、视频传感器)、路况传感器、车况传感器等.这些传感器不仅结构和功能不同，而且所采集的数据也是异构的.这种异构性极大地提高了软件开发和数据处理的难度.

(3)物联网数据的时空相关性.与普通互联网结点不同，物联网中的传感器结点普遍存在着空间和时间属性——每个传感器结点都有地理位置，每个数据采样值都有时间属性，而且许多传感器结点的地理位置还是随着时间的变化而连续移动的，如智能交通系统中，每个车辆安装了高精度的GPS或RFID标签，在交通网络中动态地移动.与物联网数据的时空相关性相对应，物联网应用中对传感器数据的查询也并不仅仅局限于关键字查询.很多时候，我们需要基于复杂的逻辑约束条件进行查询，如查询某个指定地理区域中所有地质类传感器在规定时间段内所采集的数据，并对它们进行统计分析.由此可见，对物联网数据的空间与时间属性进行智能化的管理与分析处理是至关重要的.

(4)物联网数据的序列性与动态流式特性.在物联网系统中，要查询某个监控对象在某一时刻的物理状态是不能简单地通过对时间点的关键字匹配来完成的，这是因为采样过程是间断进行的，查询时间与某个采样时间正好匹配的概率极低.为了有效地进行查询处理，需要将同一个监控对象的历次采样数据组合成一个采样数据序列，并通过插值计算的方式得到监控对象在指定时刻的物理状态.采样数据序列反映了监控对象的状态随时间变化的完整过程，因此包含比单个采样值丰富得多的信息.此外，采样数据序列表现出明显的动态流式特性——随着新采样值的不断到来和过时采样值的不断淘汰，采样数据序列是不断的动态变化的.

针对物联网海量数据管理所面临的上述挑战，目前尚没有有效的解决方法.在海量数据处理方面，最有效的方法之一是云数据管理技术，但几乎所有的云数据管理系统均为“键-值”数据库，即按照主关键字对数据进行分布组织和查询处理.这种方法无法有效地支持对传感器采样数据的时空查询处理.另一种海量数据管理方法是并行数据库技术，通过将多个关系数据库组织成数据库集群来支持海量结构化数据的处理，但这种方法在处理关键字查询时

的性能要远低于“键－值”数据库；此外，由于采用了严格的事务处理机制，在传感器采样数据频繁更新的条件下，并行数据库的数据处理效率十分低下.

在传感器采样数据的表示方面，最自然的想法是采用时空数据库的有关方法.但是，传感器采样数据除了时空属性之外，还包含物理目标的种种其它属性(如温度、压力、交通流密度、速度等)，如何对这些异构的物理属性进行统一的表示是目前尚未解决的问题.此外，目前的时空数据库方法主要针对单个数据库结点，在大规模时空数据库集群方面尚没有有效的解决方案.

针对上述问题，本文提出一种“面向物联网海量传感器采样数据管理的数据库集群系统框架”(IoT Database Cluster System Framework for Managing Massive Sensor Sampling Data，IoT-ClusterDB).在IoT-ClusterDB中，同一个监控对象的历次传感器采样值被组织成采样数据序列进行存储，通过查询操作及时空计算，可以支持对传感器采样数据的复杂逻辑条件查询.此外，IoT-ClusterDB是一个由大量数据库组成的分布式系统，通过建立分布式的全局关键字索引和全局时空索引，IoT-ClusterDB可以支持高效率的关键字查询和时空查询.

本文第2节综述物联网海量数据管理方面的相关工作；第3节阐述IoT-ClusterDB的体系结构及其核心工作机理；第4节和第5节分别阐述IoT-ClusterDB的单个结点及集群系统的组织方法；第6节讨论系统的实现与实验结果；第7节给出相关结论.

2　相关工作

在传感器采样数据的存储与查询处理方面，传统的方法主要是分布式的存储方法.分布式存储方法[8-10]是指将采样数据直接存储在各传感器结点，或者存放在Sink节点，查询处理时再通过远程访问获取数据.如果查询涉及到不同的传感器网络，则需要通过中间件来进行模型转换，进而实现异构传感器网络之间的互操作[11-12].上述方法的好处是：由于传感器数据直接存储在本地或就近的Sink结点，因此避免了数据存储时的传输通信开销.但是，上述方式并不适合于以密集复杂查询和异构数据集成为特征的大规模物联网环境，原因如下：

(1)物联网系统中通常涉及密集型的复杂查询处理，如时空约束条件查询、群体数据查询、统计分析查询等，这类查询往往需要对大面积的传感器进行反复的扫描，因此查询处理会涉及密集的远程访问，带来庞大的通信开销，抵消了本地数据采集不需要进行通信传输的好处.

(2)物联网系统中通常会涉及多种异构的传感器类型，通过模型转换的方式不仅表示能力有限，而且模型转换需要额外的计算开销，导致查询处理的效率较为低下.

(3)物联网中通常需要存储长时间的历史与当前采样数据，以满足溯源处理和统计分析的需要，而传感器和Sink结点的存储能力是相对有限的，难以满足长时间采样数据存储的要求.

鉴于分布式存储方式的上述局限，近年来越来越多的研究转向了集中式的传感器数据管理.在集中式数据管理系统中，各传感器按照一定的采样规则，将所采集的数据上传到数据中心进行统一的存储管理，使得查询处理可以直接在数据中心完成，而不需要给传感器或Sink结点带来额外的计算与通信开销.由于数据中心具有相对强大得多的存储与计算能力，因此这种方式可以支持各种复杂的、密集型的查询，更加适合于物联网的相关应用环境.

在集中式的传感器数据处理方面，最直接的方法是采用云数据管理及其相关技术[13-16].云计算是最近几年新兴的一个技术领域，其核心特点是通过一种协同机制，动态管理几万台、几十万台甚至上百万台计算机资源所具有的总处理能力，并按需分配给全球用户，使它们可以在此之上构建稳定而快速的存储以及其它IT服务，因此为物联网海量数据处理提供了一种可能.然而，目前绝大多数的云数据管理系统属于“键-值”数据库，如Bigtable[17]、Dynamo[18]、HBase[19]、PNUTS[20]、HIVE[21]等(少量的云数据库系统采用下面将讨论的并行数据库技术，如SQL Azure[22]等).“键-值”数据库能够高效地处理基于主关键字的查询，但不能有效地支持物联网数据的时空关系表示与存储、时空逻辑条件查询以及属性约束条件查询等.

另一种集中式的传感器数据管理方法是采用并行数据库技术[23-25].并行数据库通过将多个关系数据库组织成数据库集群来支持海量结构化数据的处理，但这种方法在处理关键字查询时的性能要远低于“键-值”数据库，无法根据传感器的标识快速地检索到所需要的数据.此外，由于采用了严格的分布式事务处理机制(如两阶段提交协议、数据加锁协议等)，在传感器采样数据频繁上传和更新的情况下，

数据处理的效率十分低下.最后,传统的并行数据库技术主要针对通用的数据类型,尚不能有效地支持物联网中传感器时空相关数据的并行存储与查询处理.

文献[26]讨论了传感器数据中心(SDC)的数据复制与负载均衡问题.作者虽然对集中式传感器数据中心的最终目标进行了描述,但是目前的研究还比较初步.所给出的 SDC 框架是直接建立在计算机集群上的,而不是通过数据库集群的方式,因此没有解决传感器异构时空数据的表示模型、全局索引、全局查询机制等关键技术问题.

在异构传感器采样数据的统一表示方法方面,最直接的思路是采用空间数据库及时空数据库的有关方法[27-28],解决传感器数据的异构性和时空相关性问题.但是,传感器采样数据除了时空属性之外,还包含物理对象的种种其它属性(如温度、压力、交通流密度、速度等),如何对这些异构的物理属性进行统一的表示是目前尚未解决的问题.此外,空间数据库主要面向静态的空间对象,而时空数据库虽然可以处理动态上传的数据(如时空轨迹数据),但目前的研究主要针对单个时空数据库结点,在大规模的时空数据库集群方面的研究尚处于空白.

从数据流处理的角度来看,虽然物联网传感器采样数据表现出动态上传的流式特性,但是其处理方法与流数据管理技术有着根本的不同:目前的绝大多数流数据库,如 TelegraphCQ[29]、Gigascope[30]、System S[31]、Net-Fli[32]等,主要解决如何在数据流到来的过程中,实时地解析出查询结果(通常针对相对简单的查询,如查找某个特定的数据项或序列模式等),数据库内并不存储长时间的历史数据;而物联网中需要保存长时间的历史数据,并在此基础上实现复杂和灵活得多的查询处理.

综合以上分析可以看出,针对物联网中传感器采样数据管理所面临的海量性、异构性、时空相关性及动态流式特性等挑战,目前尚没有成熟的解决方案,需要有针对性地进行专门的研究.

3 IoT-ClusterDB 的系统结构

在本节,我们描述 IoT-ClusterDB 的体系结构(见图 1)与工作机理.

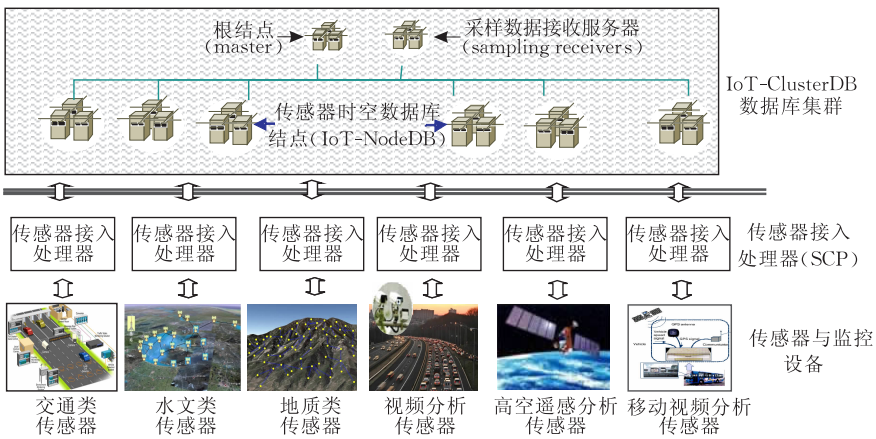


图 1 IoT-ClusterDB 的体系结构

如图 1 所示,IoT-ClusterDB 是由多个传感器时空数据库结点(IoT Sensor Spatial-Temporal DataBase Node,IoT-NodeDB)所组成的一个数据库集群.在 IoT-ClusterDB 中,并不是原封不动地存放所有的传感器采样数据,而是只存放数值型的关键采样数据.为此,系统需要通过传感器接入处理器对数据进行预处理.此外,原始的传感器采样数据也保存在各个传感器接入处理器中,以满足溯源处理的要求.

从层次划分的角度来看,传感器接入处理器位于传感器网络层之上,因此对各种传感器网络本身的数据采集、通信模式没有特殊要求,从而增加了整个系统的灵活性和可扩充性.

3.1 传感器的接入与预处理

在 IoT-ClusterDB 中,允许接入的监控设备包括物联网系统中所管理的各类传感器设备、视频与音频监控设备、部署在某个区域的无线传感器网络(WSN)等.此外,IoT-ClusterDB 还允许以人工的方式输入感知数据.所有这些方式获得的数据,我们均统一地称之为"传感器采样数据".

通常情况下,传感器的采样数据是数值型的,如温度传感器、压力传感器、GPS 传感器、无线传感器网络等所获得的数据.但是,IoT-ClusterDB 也允许多媒体设备(如视频监控设备、遥感成像设备、高空成像设备等)接入系统,通过相应的多媒体分析,可

以从这些设备所获得的多媒体数据(如交通摄像头采集的视频数据流)中提取出有意义的数值型数据(如道路交通流的平均速度、车辆密度、交通流量等).因此,多媒体设备连同传感器接入处理器(见图 2)中相应的多媒体分析模块,可以达到与普通传感器一样的效果,所以在本文中我们也将它们统一地纳入传感器的范畴.

通过上述分析可以看出,本文所说的"传感器数据"事实上是一个广义的概念,包括任意类型带有时空特性的、反映物理世界状态的数据.例如,在发生 SARS 疫情时,每一个病例登记即可看成一个传感器采样数据,该数据反映了病例发生的时间、地点、规模等信息.又如,在大面积布设的 WSN 中,系统本身即可根据多个传感器个体的采样值获得汇总数据(如污染带的面积、位置等),这些汇总数据也可以看成传感器采样数据(在这种情况下,可以将整个 WSN 看成一个传感器,而不需要对其中的单个传感器进行管理).

传感器将采样数据上传给传感器接入处理器(Sensor Connecting Processor,SCP)的方式可分为两种:主动上传与被动上传.其中,主动上传是根据预先定义的条件,由传感器自身进行计算和判断,只有当规定的条件满足时才上传数据,这种方式具有较好的数据传输效率,但需要传感器具备一定的计算能力;而被动上传则是以一定的频率周期性地上传数据,这种方式虽然具有较大的通信代价,但对传感器的计算能力要求很小,因此也得到了广泛的应用.

在 IoT-ClusterDB 中,传感器采样数据是以"原子监控对象(atomic monitored object)"为单位进行组织的,而不是以传感器为单位.同一个监控对象的所有传感器采样值按照时间序列组织在一起,形成该监控对象的"采样数据序列",并作为一个属性存放在该监控对象的元组记录中.

在绝大多数情况下,一个监控对象即对应于一个传感器(如太湖中的一个温度传感器可看成是一个监控对象,对应于一个具体的温度监测点,因此太湖中可包含大量的监控对象);但某些时候两者并不具有一一对应的关系,例如 RFID 传感器与所监控的车辆或货物之间并没有固定的对应关系——带有 RFID 标签的监控对象的采样数据序列是由整个系统中的多个 RFID 传感器所采集的数据汇总而成的.

根据监控对象的位置是否移动,我们可以将它们分为两大类:静止监控对象(如太湖中固定布设的温度传感器、车库中的剩余车位计数器、无线传感器网络等)和移动监控对象(如带 GPS、RFID 标签、条形码的车辆与货物、浮动车上布设的视频交通流传感器等).

传感器接入处理器可以实现大量传感器的接入,实现对传感器原始采样数据的分析、过滤与转换,完成原始采样数据的本地存储,并将处理后的数值型关键采样数据上传到 IoT-ClusterDB 数据库集群中做进一步的处理.传感器接入处理器分担了整个系统的很大一部分数据处理与存储任务,使得 IoT-ClusterDB 只需处理相对较少的、带有语义信息的数值型关键采样数据.图 2 给出了传感器接入处理器的工作过程.

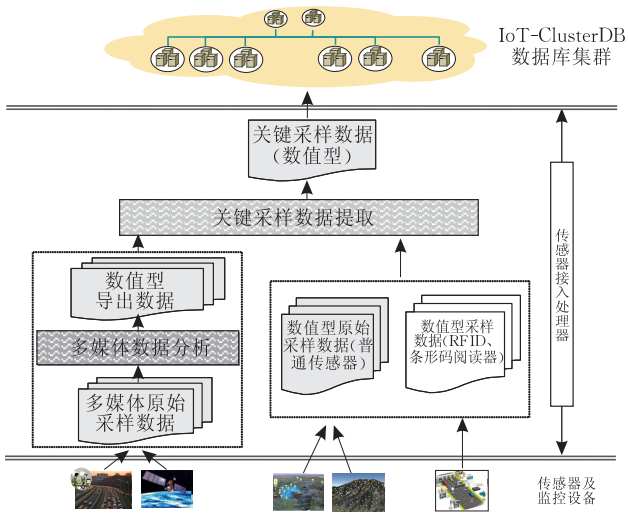


图 2　传感器接入处理器的主要工作过程

如图 2 所示,传感器接入处理器的主要工作过程如下:

(1)非数值型采样数据的数值化.对于多媒体原始采样数据,传感器接入处理器需要进行相关的多媒体数据分析,获得能够反映监控对象物理状态的数值型导出数据,如通过对交通摄像头视频图像的分析,可以提取出监控地点的车辆密度、平均车速、交通流量等参数信息;通过对高空遥感图像的分析,可以提取出污染区域及火灾区域的地理几何形状等空间数据信息.

(2)密集采集数据的稀疏化.无论是通过多媒体数据分析所获得的数值型导出数据,还是直接从传感器获得的数值型原始采样数据,它们仍然存在着采样频率过高的缺陷.如果将这些数据全部传入 IoT-ClusterDB 进行管理,则会导致数据中心数据量的急剧膨胀.为此,传感器接入处理器需要通过关键数据的提取操作,从原始数据流中抽取出能够反映监控目标物理状态变化的关键采样数据,仅将关键采样数据上传给 IoT-ClusterDB 进行处理.

在关键数据的提取方面,一种简单的方法是基于状态变化阈值的关键数据提取方法——对于每一种传感器可以定义一个状态变化阈值,如果新的采样数据与上次上传的关键采样数据之间的差值没有超过该阈值,则不需要进行任何处理;仅当它们之间的差值超过规定的阈值时,新的采样数据才成为需要上传的关键采样数据.此外,我们还可以采取其它的更加智能化的数据提取方法,在保证关键数据上传的前提下,过滤掉大量的冗余数据(注意,如果传感器采用主动上传的方式,则采样数据已经进行了稀疏化处理,SCP 不需要再进行相应的处理).

(3) RFID、条形码阅读器采样数据的提取.对于 RFID 传感器、条形码阅读器等所采集的数据,传感器接入处理器需要提取出移动监控对象的标识,并连同相关的采样时间与采样地点组成采样记录,然后将该采样记录发送给 IoT-ClusterDB 进行处理.IoT-ClusterDB 将同一个移动监控对象的、来自于不同 RFID 传感器或条形码阅读器的采样数据集中起来,可以获得该监控对象的采样数据序列,该数据序列反映了其完整的时空移动过程.

(4) 原始采样数据的存储.对于具有保留价值的原始采样数据(如视频监控多媒体数据、油库温度原始采样数据等)也由传感器接入处理器进行存储管理.在 IoT-ClusterDB 中,每个监控对象的元组数据中均含有存储该对象的原始采样数据的 SCP 标识,通过这些标识,查询用户可以连接到相应的 SCP,并通过相应的数据访问接口检索和回放完整的历史与当前原始采样数据.

3.2 IoT-ClusterDB 数据库集群

IoT-ClusterDB 数据库集群是由大量的同构传感器时空数据库结点 IoT-NodeDB 所组成的,每个 IoT-NodeDB 可以对各类异构的传感器采样数据进行统一化的管理.在 IoT-NodeDB 中,数据是以监控对象为单位进行管理的——每个监控对象对应于一个元组,该监控对象的所有关键采样数据存放在一起构成一个采样数据序列,并作为一个属性值存放在该元组中.此外,每个监控对象的元组中还包含存放原始采样数据的 SCP 标识,使得各个传感器接入处理器与 IoT-ClusterDB 能够分担系统的计算与存储任务并协同工作.

在 IoT-ClusterDB 数据库集群中,大量的 IoT-NodeDB 被组成双层树形结构,其中叶结点存储实际的传感器采样数据,而根结点则存储为了进行全局查询所需要的全局数据字典.所有的查询均提交给根结点,根结点通过全局查询处理模块,实现对查询的全局处理.在 IoT-ClusterDB 中,通过建立分布式的全局关键字索引和全局时空索引,整个系统可以同时支持快速的关键字查询、时空查询以及复杂的逻辑条件约束查询.

4 传感器时空数据库模型

在本节,我们讨论 IoT-ClusterDB 中的各传感器时空数据库结点(IoT-NodeDB)的数据管理方法,并重点讨论如何使用统一的数据库表示方式,对异构的传感器关键采样数据进行存储与查询处理.我们假设数据库中已经实现了标准数据类型、空间数据类型以及基于这些类型的相关查询操作,并采用文献[27-28]中的符号表示方法对数据类型和操作进行相关的表示.

如前所述,在 IoT-ClusterDB 中可以接入海量的异构传感器结点.每一种类型的传感器所获得的采样数据均可以具有不同的数据格式,但它们的共同特点是均具有时空特性:即每个传感器采样数据均对应于一个具体的采样时间 $t \in Instant$ 和一个具体的采集地点 $loc \in Point \cup Region$.在多数情况下,传感器数据的采样地点是一个精确的位置(即 $loc \in Point$),如风力与风向传感器、GPS 传感器等采集的数据.但是,有时候也存在采样位置不精确的情况(即 $loc \in Region$),如在无线传感器网络中,通常在某个区域中可以布设一群传感器,此时所得到的采样数据(可以是单个传感器的采样值,或者是群体传感器的汇总数据)对应的采样地点即为一个地理区域.

此外,为了快速地查询和分析各监控对象的历次采样信息,传感器采样数据应该以监控对象为单位进行组织,使得同一个监控对象的所有数据都存放在一起,并随着时间而动态变化.因此传感器采样数据表现出序列性和动态变化的流式特性.

在 IoT-NodeDB 中,为了对异构的传感器流式时空相关数据进行有效的管理,需要定义相应的数据类型与查询操作,在数据库内核一级实现传感器采样数据的高效存储与查询处理.

4.1 数据类型

在本小节,我们首先定义单个传感器采样值的表示方法,在此基础上,给出监控对象的采样数据序列的表示方法.通过这些数据类型,数据库内核可以对异构的传感器关键采样数据流进行统一的表示与存储处理.

定义 1(传感器采样值). 传感器的单个采样值 *SamplingValue* 可以表示为如下形式：

$SamplingValue=(t,loc,npos,schema,value)$，

其中：$t\in Instant$ 是该采样数据所对应的采样时间；$loc\in Point\cup Region$ 是采样地点；$npos\in String$ 是该采样数据所对应的交通网络位置(以有向道路的 ID 表示；当采样位置不在交通网络中时 *npos* 为空值)；$schema\in String$ 和 $value\in String$ 分别是采样数据的“型”和“值”，其中“型”描述了采样数据的格式及语义，“值”是具体的采样数据值. 由于 *schema* 和 *value* 均可能包含多个分量，因此需要用括号对它们的边界进行划分.

在上述定义中，采样地点 *loc* 的表示形式取决于监控对象的类型. 对于移动监控对象(通常通过 GPS、RFID 或条形码阅读器进行定位)，必然有 $loc\in Point$；而对于静止监控对象，可以包含 $loc\in Point$(如固定布设的交通流传感器)和 $loc\in Region$(如无线传感器网络中成片布设的传感器)两种情况.

表 1 给出了各种不同类型的传感器关键采样数据的例子(设 $t_1\sim t_5$ 是采样时间，$e_{201}\sim e_{202}$ 是有向道路的标识).

表 1　传感器采样值的例子

传感器类型	传感器采样值(SamplingValue 数据类型)
温度传感器(部署在某区域的 WSN)	$(t_1,region_1,$ NULL，(*temperature*：real)，(27.5))
GPS 传感器	$(t_2,(39.3,144.3),e_{201},$(*speed*：real，*direction*：real)，(62.5,22))
风速风向传感器	$(t_3,(39.3,144.3),$ NULL，(*windspeed*：real，*winddir*：real)，(62.5,22))
交通流视频分析传感器	$(t_4,(39.3,144.3),e_{202},$(*averageSpeed*：real，*jam*：bool)，(62.5,true))
高空遥感识别传感器(火灾区域识别)	$(t_5,(39.3,144.3),$ NULL，(*disasterArea*：region)，($region_2$))

传感器的采样值可以由多个分量组成，如表 1 中 GPS 传感器的采样值具有两个分量：速度和方向(注意：GPS 所采集的经度、纬度信息是传感器采样值的基本信息，表示在 *loc* 属性中，所以并不属于采样值的分量). 为了进一步表示传感器采样值的分量，我们定义如下 *SamplingComponent* 数据类型.

定义 2(采样值的分量). 传感器采样值的分量 *SamplingComponent* 可以表示为如下形式：

$SamplingComponent=(cSchema,cValue)$，

其中，$cSchema\in String$，$cValue\in String$ 分别是采样值分量的“型”和“值”. 例如，表 1 中 GPS 传感器采样值的速度与方向分量分别表示为(*speed*：real，62.5)和(*direction*：real，22).

定义 3(采样数据序列). 同一个监控对象的历次采样值(可以来自于同一个传感器，也可以来自于多个传感器(如 RFID、条形码阅读器等))按照时间序列存放在一起，构成该对象的采样数据序列 *SamplingSequence*，表示为如下形式：

$SamplingSequence=(schema,((t_i,loc_i,npos_i,value_i,flag_i))_{i=1}^{n})$，

其中，$schema\in String$ 是采样序列中各采样值的“型”，$t_i\in Instant$、$loc_i\in Point\cup Region$、$npos_i\in String$ 和 $value_i\in String$ 分别是第 i 个采样值的采样时间、采样地点、采样地点对应的网络位置以及实际的采样数值，$flag_i$ 表示第 i 个采样值是否为该数据序列中的一个“间断点”，即一个新片断的起点.

下面让我们来进一步讨论 $flag_i$. 在 IoT-ClusterDB 中，数据是根据地理区域进行分布的(详见第 5.1 节). 对于移动监控对象(如 GPS 传感器、RFID、条形码阅读器监控的对象)来说，其采样数据序列可以被分割成多个片段，并被存储在不同的数据库结点中. 如果某个移动监控对象 *mobj* 多次进入同一个数据库结点所对应的地理区域，则该数据库结点中存放的 *mobj* 的采样数据序列包含多个片段(如图 3 所示)，而 $flag_i$ 则用于表示对应的采样值是否为一个新片段的起点.

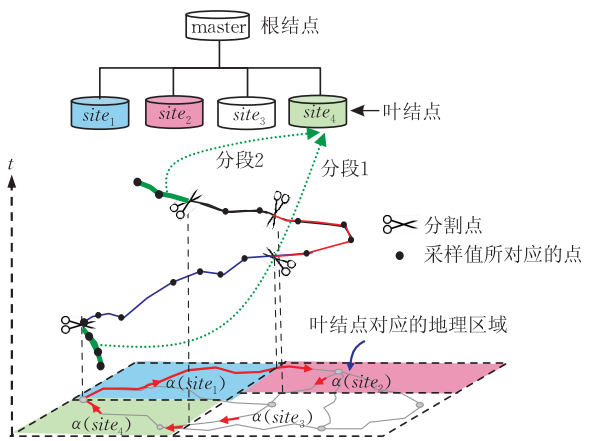


图 3　采样数据序列包含多个分段的情况

对于静止监控对象，由于其历次采样值的 *loc* 属性保持不变，因此 *SamplingSequence* 的格式可以简化为(注意此时尽管不存在采样数据序列分割的问题，但 $flag_i$ 仍然是需要的，用于表示监控过程被暂时挂起的情况)：

$SamplingSequence=(schema,loc,npos,((t_i,value_i,lag_i))_{i=1}^{n})$.

上述两种 *SamplingSequence* 表示方式在格式

上稍有区别，由数据库系统自动进行区分和处理.

在系统实现时，*SamplingSequence* 数据类型被设计成一个指向外存文件数据块的指针，真实的采样数据序列被存放在文件块中，而不是直接存放在元组中.这样当新的采样数据到来并淘汰旧的采样数据时，只需要直接修改文件数据块即可，而不需要对数据库元组进行修改，从而提高了处理的效率.

通过上述数据类型，我们可以建立关系表用以表示和存储物联网中各监控对象的采样数据序列，如

CREATE TABLE IoTData (ObjID: *String*,
ObjType: *String*, DeployedBy: *String*,
DepoyedTime: *Instant*, HostObj: *String*,
Samplings: *SamplingSequence*);

在上述关系表模式中，ObjID、ObjType、DeployedBy、DepoyedTime 分别是监控对象的标识、监控对象的具体类型(如 GPS、RFID、摄像识别传感器、温度传感器等)、部署者、部署时间，HostObj 是该监控对象的宿主对象(如太湖中的传感器的宿主对象为太湖)，Samplings 是该监控对象的采样数据序列.此外，IoT-NodeDB 中每个监控对象的元组还隐式地包含一个 SCPSet 属性和一个 StaticMov 属性，其中 SCPSet 中存放该监控对象的原始采样数据的 SCP 集合，而 StaticMov 是一个 Bool 型的标记，用于区分静止监控对象和移动监控对象.

4.2 查询操作

上述数据类型允许我们在数据库内核中以统一的数据格式表示异构的传感器采样数据流.为了对传感器数据进行查询，我们还需要在这些数据类型的基础上定义一系列的查询操作.

4.2.1 针对 *SamplingSequence* 数据类型的操作

针对 *SamplingSequence* 数据类型的最重要的查询操作是 atInstant 操作，该操作用于计算监控对象在某个给定时间 t_q 的状态值，其语法格式如下(在操作的语法格式定义中，符号"→"两边分别是该操作的输入数据和输出数据的数据类型；如果操作有多个输入数据，则输入数据类型之间用"×"进行连接)：

atInstant：

$SamplingSequence \times Instant \rightarrow SamplingValue$.

假设 atInstant 操作的两个输入分别是 $sequ \in SamplingSequence$ 和 $t_q \in Instant$，其中 $sequ = (schema, ((t_i, loc_i, npos_i, value_i, flag_i))_{i=1}^{n})$.如果 t_q 正好为某个关键采样值所对应的时间，即 $t_q = t_j$ $(j \in [1,n])$，则 atInstant 操作直接将该关键采样值 $(t_j, loc_j, npos_j, schema, value_j)$ 作为结果返回，否则进行如下处理：

(1) 如果 $t_q < t_1$，则直接返回"未定义"(用"⊥"表示)；

(2) 如果 $t_{j-1} < t_q < t_j$ $(j \in [2,n])$，则需要进一步检查 $flag_j$：如果 $flag_j$ = True，则表明第 j 个采样值为间断点，此时返回"⊥"；如果 $flag_j$ = False，则需要通过插值的方法得到 t_q 时刻的结果.在进行插值计算时，根据采样值的 *npos* 属性是否为有效的网络位置，可以选用基于交通网络的插值方法或基于 Euclidean 空间的插值方法；

(3) 如果 $t_q > t_n$，则返回 $(t_n, loc_n, npos_n, schema, value_n)$(注意，此时返回值中用的是 t_n 而不是 t_q，用以表明采样的实际时间).

通过 atInstant 操作，数据库可以支持对监控对象在监控时间段内任意时刻的状态查询.

空间投影操作 sProjectLines、sProjectPoint、sProjectNetPos 和时间投影操作 sProjectTime 分别将采样数据序列向空间平面和时间轴上投影，它们的语法格式如下：

sProjectLines: $SamplingSequence \rightarrow Lines$
(针对移动监控对象)
sProjectPoint: $SamplingSequence \rightarrow Point$
(针对 $loc \in Point$ 的静止监控对象)
sProjectRegion: $SamplingSequence \rightarrow Region$
(针对 $loc \in Region$ 的静止监控对象)
sProjectNetPos: $SamplingSequence \rightarrow Set(String)$
sProjectTime: $SamplingSequence \rightarrow Periods$

在上述空间投影操作中，sProjectPoint、sProjectLines 和 sProjectRegion 分别针对特定的监控对象，对于其它类型的监控对象则返回"⊥"；sProjectNetPos 操作返回空间投影值所对应的网络位置.

空间截取操作 sTruncateGeo 和时间截取操作 sTruncateTime 分别根据给定的空间范围和时间范围截取采样数据序列的一部分，它们的语法格式如下：

sTruncateGeo:
$SamplingSequence \times Region \rightarrow SamplingSequence$;
sTruncateTime:
$SamplingSequence \times Periods \rightarrow SamplingSequence$.

为了支持新采样值到采样数据序列的插入，我们定义 samplingAppend 操作.该操作将一个新的采样值(连同表示是否为间断点的 *flag* 标识)附加到采样数据序列的末尾，同时根据系统中预定义的监控时间长度(如 3 个月)淘汰采样数据序列中过时

的采样数据. 该操作的语法格式如下：

samplingAppend：

$SamplingSequence \times SamplingValue \times Bool \rightarrow SamplingSequence$.

4.2.2　针对 *SamplingValue* 数据类型的操作

针对 *SamplingValue* 的操作主要包括对采样值的投影操作 vProjectTime、vProjectPoint、vProjectRegion 和 vProjectNetPos 以及数据提取操作 getComponent. 它们的语法格式如下：

vProjectTime：$SamplingValue \rightarrow Instant$；

vProjectPoint：$SamplingValue \rightarrow Point$

（针对 $loc \in Point$ 的监控对象）；

vProjectRegion：$SamplingValue \rightarrow Region$

（针对 $loc \in Region$ 的监控对象）；

vProjectNetPos：$SamplingValue \rightarrow String$；

getComponent：

$SamplingValue \times integer \rightarrow SamplingComponent$.

如果将 *SamplingSequence* 看成监控对象的状态曲线，则一个 *SamplingValue* 值实际上对应于监控对象状态曲线中的一个点. 因此，vProjectTime、vProjectPoint 和 vProjectNetPos 操作的结果分别是 *Instant*、*Point*/*Region* 和 *String* 型的值.

由于 *SamplingValue* 型的值可以有多个分量，getComponent 操作根据指定的分量序号 i，取出采样值的第 i 个分量.

4.2.3　其它操作

为了实现 *SamplingComponent* 数据类型与数据库中其它数据类型的交互操作，还需要通过“lifting”[27-28]的方式，对数据库中的各种标准查询操作（如＋、－、×、/、＜、＝、＞等）和空间查询操作（如 inside, intersect, touches, distance, direction, overlap 等）进行扩充，使得 *SamplingComponent* 可以作为这些操作的输入数据类型参与这些操作的计算. 例如，“＝”操作在扩充之后的语法格式为（设“*BASE*”和“*SPATIAL*”分别是标准数据类型的集合和空间数据类型的集合）：

$$=: \alpha \times \beta \rightarrow Bool,$$

其中 $\alpha, \beta \in \{samplingComponent\} \cup BASE \cup SPATIAL$.

此外，在 IoT-NodeDB 中，所有的查询均是以 SQL 语句的格式提交的，即便对于关键字查询也套用 SQL 语句的格式. 为此，我们定义如下 keySearch 操作，以提供关键字查询接口：

keySearch：$String \rightarrow set(Tuple)$.

如前所述，IoT-NodeDB 中每个监控对象的元组均隐含一个 SCPSet 属性，用以表示存放该监控对象原始采样数据的 SCP 的标识. 为了获得 SCPSet 属性的值，我们定义操作 GetRawSites，该操作的输入为监控对象的标识 ObjID$\in$*String*，输出为 SCP 的标识集合. GetRawSites 操作的语法格式如下：

GetRawSites：$String \rightarrow set(String)$.

查询用户通过 GetRawSites 操作获得相应 SCP 的标识之后，可以直接与相应的 SCP 进行通信，并通过原始采样数据浏览接口进行相关数据的存取与访问.

4.2.4　对传感器采样数据进行查询的例子

本节前部分定义的所有数据类型和查询操作均实现为数据库系统的内嵌形式，因此查询语言（包括数据定义语言 DDL 和数据操纵语言 DML）是扩充后的 SQL 语言形式. 下面给出一些查询的例子：

［查询 $Q1$］通过关键字查询获得监控对象 $objID1$ 的采样数据序列

```
SELECT keySearch(objID1). Samplings
FROM IoTData;
```

在上述查询中，keySearch 操作返回一组元组的集合（由于 $objID1$ 只对应一个元组，因此该集合中只含一个元素），而 keySearch（$objID1$）. Samplings 返回这些元组的“Samplings”属性值.

［查询 $Q2$］通过关键字查询获得所有由“BeijingTraffic”部署的监控对象的采样数据序列

```
SELECT keySearch("BeijingTraffic"). Samplings
FROM IoTData;
```

［查询 $Q3$］通过属性约束条件实现查询 $Q2$

```
SELECT Samplings
FROM IoTData
WHERE DeployedBy="BeijingTraffic";
```

查询 $Q2$ 和 $Q3$ 的区别在于 $Q2$ 采用关键字查询，因此其执行速度可能会快于采用 SQL 查询的 $Q3$.

［查询 $Q4$］查询所有位于地理区域 $region_1$ 内且在 t_1 时刻的风速大于 50 的风力监控对象，返回除了“Samplings”属性之外的其它属性值

```
SELECT
ObjID, ObjType, DeployedBy, DepoyedTime, HostObj
FROM IoTData
WHERE ObjType="Windsensor" AND
  inside (sProjectPoint(Samplings), region1) AND
  getComponent(atInstant(Samplings, t1), 1)>50;
```

［查询 $Q5$］查询所有 t_1 时刻位于地理区域 $region_1$ 内的视频交通流监控移动目标（如安装在公交车辆上的摄像头）

```
SELECT *
FROM IoTData
WHERE ObjType="BusTrafficVideo" AND
  inside(vProjectPoint(atInstant(Samplings,t1)),region1).
```

5 IoT-ClusterDB 数据库集群全局处理

物联网系统中海量的传感器对各种物理目标的状态进行着实时的监控. 为了对海量传感器数据进行快速处理,我们需要大量的数据库结点并将它们组织成一个协同工作的物联网集群存储系统.

如前所述,IoT-ClusterDB 采用一种双层树形结构,其中叶结点存储真正的传感器采样数据,而根结点则存储为了进行全局查询所需要的全局数据字典. IoT-ClusterDB 的体系结构如图 4 所示.

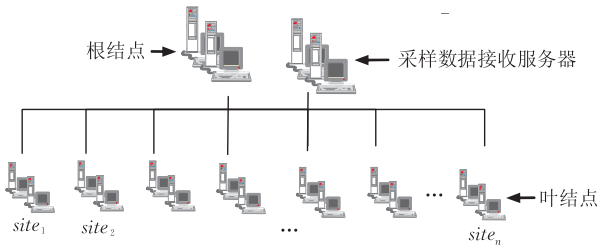


图 4 IoT-ClusterDB 的体系结构

在 IoT-ClusterDB 中,每个数据库结点(包括根结点和各个叶结点)均为传感器时空数据库. 各结点除了具有第 4 节所描述的数据类型和查询操作之外,还协同建立了分布式的全局索引及全局查询处理模块.

为了提高系统的可靠性,根结点和每个叶结点都包含两个或多个数据库副本,如果其中一个副本失效,其它的副本可以接管其工作.

5.1 数据分布策略

在 IoT-ClusterDB 中,每个叶结点 *site* 对应于一个地理区域,称之为该结点的"管辖区域",记为 $\alpha(site)$. 所有叶结点的管辖区域均登记在根结点和采样数据接收服务器中的管辖区域分区表(Service Area Partitioning Table,SAP-Table)中.

设 IoT-ClusterDB 系统包含 n 个叶结点:$site_1$,$site_2$,…,$site_n$,且总的应用地理区域为 A,则有如下条件成立:

(1) $\forall i,j(i\neq j)$: $\alpha(site_i)\bigcap\alpha(site_j)=\varnothing$;

(2) $\bigcup_{i=1}^{n} site_i=A$.

在 IoT-ClusterDB 中,数据是按照地理区域进行分布的. 各采样值根据其采样地点所属的管辖区域被分布到不同的叶结点进行存储,总的数据分布原则如下(设数据库采用第 4.1 节中所描述的模式):

(1) 对于任意一个移动监控对象 *mobj*,设 *sites*(*mobj*)是 *mobj* 在监控时间范围内经过了其管辖区域的叶结点的集合,则 *mobj* 对应于多个元组,*sites*(*mobj*)中的每个叶结点中存放其中的一个元组,这些元组除了"Samplings"属性之外的其它属性值是相互复制的,而其"Samplings"属性值则在 *sites*(*mobj*)中的各叶结点之间进行分割:对于任一叶结点 $site\in sites(mobj)$,它仅存放"Samplings"与 $\alpha(site)$ 在空间上相交的部分,即 sTruncateGeo(Samplings,$\alpha(site)$),如图 5 所示.

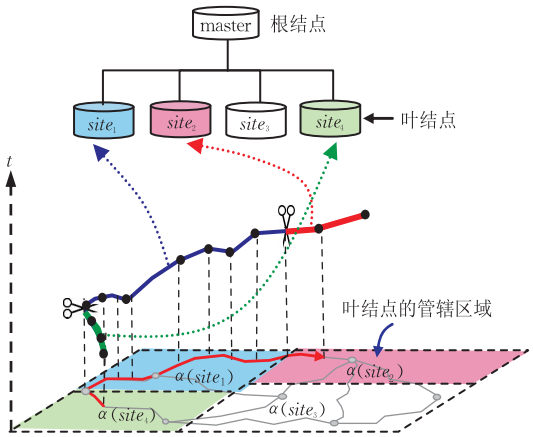


图 5 "Samplings"属性值在各叶结点之间的分割

(2) 对于静止监控对象 *sobj*,如果其采样地点 $loc\in Point$ 且 $loc\in\alpha(site_i)$,则 *sobj* 仅对应于一个元组,该元组存放在叶结点 $site_i$ 中;

(3) 对于静止监控对象 *sobj*,如果其采样地点 $loc\in Region$ 且 sites(*sobj*)是管辖区域与 *loc* 相交的叶结点集合,则 sites(*sobj*)中的每个叶结点存放 *sobj* 的一个元组,这些元组是彼此复制关系,且每个元组的"Samplings"属性均包含完整的采样数据序列.

在 IoT-ClusterDB 中,所有新的采样值均被发送给采样数据接收服务器(见图 4),采样数据接收服务器根据 SAP-Table 将采样数据值发送给不同的叶结点进行存储. 如果新的采样值属于移动监控对象且与上次采样值相比跨越了不同叶结点的管辖区域,则采样数据接收服务器需要通过插值计算得到边界处的采样值,并将插值点也发送给相应的叶结点进行存储(详见第 5.3 节).

5.2 全局索引及全局查询处理

为了支持全局查询处理,在 IoT-ClusterDB 中需要建立分布式的全局关键字索引和全局时空索引.

5.2.1　分布式全局关键字索引及全局关键字查询

为了在 IoT-ClusterDB 中支持关键字查询，需要建立一个全局关键字 B^+ 树索引（Global Full-Text Keyword B^+-Tree，GFTKB$^+$-Tree），图 6 给出了 GFTKB$^+$-Tree 的结构.

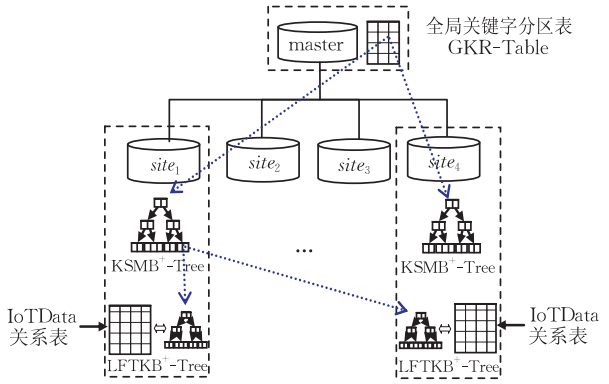


图 6　GFTKB$^+$-Tree 的结构

如图 6 所示，GFTKB$^+$-Tree 是一个分布式的索引，采用 3 层结构，其中：

第 1 层是存放在 IoT-ClusterDB 根结点中的全局关键字分区表（Global Keyword Range Table，GKR-Table）. GKR-Table 的记录格式为（*keyRange*，*siteID*），其中 *keyRange* 是关键字值域中的一个区域范围，*siteID* 是与该范围相对应的叶结点的标识，该叶结点中的 KSMB$^+$-Tree 索引该范围内的关键字.

第 2 层是一组存放在 IoT-ClusterDB 各叶结点中的 *keyword* 到 *SiteID* 的映射 B^+ 树（keyword-to-SiteID Mapping B^+-Tree，KSMB$^+$-Tree）. KSMB$^+$-Tree 的叶结点记录格式为（*keyword*，set(*siteID*)），其中 *keyword* 是关键字，set(*siteID*)是所有包含该关键字的叶结点的标识的集合. 在创建 KSMB$^+$-Tree 时，每个叶结点从自己的本地元组中提取关键字，并对每个关键字生成一个（*keyword*，*siteID*）偶对（其中 *siteID* 即为该叶结点自己的标识）；随后，该叶结点根据 GKR-Table 将各个偶对发送给相应的叶结点，这样每个叶结点可以收到一组来自于整个系统中其它各个叶结点的（*keyword*，*siteID*）集合，并在此基础上构建 KSMB$^+$-Tree.

第 3 层是一组存放在 IoT-ClusterDB 叶结点中的本地全文关键字 B^+ 树索引（Local Full-Text Keyword B^+-Tree Index，LFTKB$^+$-Tree）. LFTKB$^+$-Tree 是 IoT-NodeDB 针对本地数据建立的全文关键字索引，其叶结点的记录格式为（*keyword*，set(*tupID*)），其中 *keyword* 是一个关键字，set(*tupID*)是所有包含该关键字的所有元组的标识集合.

让我们结合第 4.2.4 小节中的查询 *Q*1 和 *Q*2 来讨论对全局关键字查询的处理. 在 IoT-ClusterDB 中，所有的查询均发送给根结点进行处理. 当根结点接收到一个关键字查询时，将首先查询 GKR-Table，从而判断应该进一步查询哪个叶结点的 KSMB$^+$-Tree；然后，根据对应叶结点的 KSMB$^+$-Tree，可以得到一组叶结点的 *siteID*，这些叶结点均包含被查询的关键字；最后，根结点将查询广播给这些叶结点并行执行，执行结果由根结点汇总并返回给查询用户. 各叶结点在执行来自于根结点的关键字查询时，将调用本地的 LFTKB$^+$-Tree 快速地得到查询结果.

算法 1 给出了全局关键字查询的处理过程.

算法 1.　全局关键字查询处理算法.

输入：全局关键字索引　GFTKB$^+$-Tree

　　关键字查询　*Q*

输出：查询结果　*R*

1. *key*=*getkey*(*Q*)；
2. *ksmb_site*=*retrieve*(*key*，GFTKB$^+$-Tree.GKR-Table)；
3. *ksmb_tree*=*get_tree*(*ksmb_site*，KSMB$^+$-Tree)；
4. *lftkb_sites*=*retrieve*(*key*，*ksmb_tree*)；
5. FOR EACH *site*∈*lftkb_sites* DO (IN PARALLEL)
6. 　*lftkb_tree*=*get_tree*(*site*，LFTKB$^+$-Tree)；
7. 　*tupleids*=*retrive*(*key*，*lftkb_tree*)；
8. 　*res*=*evaluate*(*Q*，*tupleids*)；
9. 　*sendMaster*(*res*)；
10. ENDFOR；
11. 根结点收集各叶结点发来的查询结果并存入 *R*；
12. Return(*R*).

在算法 1 中，*getkey*(*Q*)函数返回查询 *Q* 要检索的关键字；*retrieve*((*key*，*ds*)函数根据关键字 *key* 在数据结构 *ds* 中进行检索，并返回相应的结果；*get_tree*(*site*，*treeName*)函数返回指定叶结点 *site* 中的 *treeName* 索引子树，*evaluate*(*Q*，*tupleids*)函数在 *tupleids* 元组集合的基础上执行查询并返回结果，*sendMaster*(*res*)将结果 *res* 发送给根结点.

5.2.2　分布式全局时空索引及全局时空查询处理

除了关键字查询之外，物联网系统中的另一类重要的查询类型是带有时空约束条件的查询，如第 4.2.4 小节中的查询 *Q*4 和 *Q*5. 为了支持这类查询，我们需要建立全局时空 R 树索引（Global Spatial-Temporal R-Tree，GSTR-Tree）. 由于在 IoT-ClusterDB 中数据是根据其空间属性进行分布的，因此 GSTR-Tree 的构建可以得到简化，图 7 给出了其总体结构.

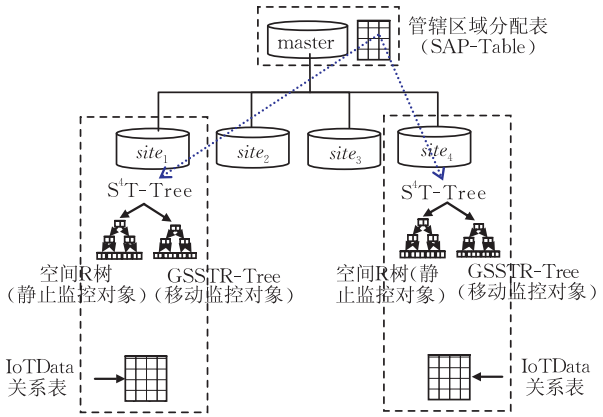
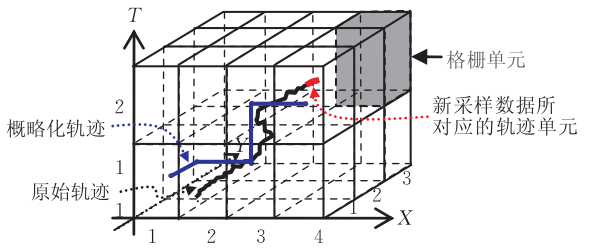


图 7 GSTR-Tree 的结构

如图 7 所示,GSTR-Tree 是一个两层的分布式全局索引,其中:

第 1 层是 IoT-ClusterDB 根结点中的管辖区域分区表(SAP-Table). SAP-Table 中的记录的格式为(*area*, *siteID*),其中 *area* 为一个管辖区域,*siteID* 是与该管辖区域对应的叶数据库结点的标识;

第 2 层是 IoT-ClusterDB 各叶结点中的针对本地数据的传感器采样数据序列时空索引树(Sensor-Sampling-Sequence Spatial-Temporal Tree, S⁴T-Tree),用以对本地数据中采样数据序列的时空属性进行索引. S⁴ T-Tree 实际上包括两个子树:一个空间 R 树(Spatial R-Tree,SR-Tree)用以对静止监控对象的空间位置(包括 $loc \in Point$ 和 $loc \in Region$ 两种情况)进行索引,一个格栅概略化时空 R 树(Grid-Sketched Spatial-Temporal R-Tree,GSSTR-Tree)用于对移动监控对象的随时间动态变化的位置(亦称为"时空轨迹-trajectory")进行索引. 图 8 给出了 GSSTR-Tree 的结构.



(a) 移动对象的轨迹及概略化轨迹

(b) GSSTR-Tree的结构

图 8 格栅概略化时空 R 树(GSSTR-Tree)

从图 8 可以看出,移动监控对象时空轨迹的变化十分频繁,因为每一次新采样值的到来均会导致向轨迹中插入新的轨迹单元(轨迹单元是轨迹时空曲线中相邻两个采样点构成的一条直线段). 如果直接以轨迹单元作为索引记录的基本单位,则索引的更新频率将等同于采样数据插入数据库的频率,从而导致索引更新代价升高.

为了解决上述问题,我们首先将 $X \times Y \times T$ 空间划分成粒度较粗的等距格栅. 然后,将每个移动监控对象的轨迹映射成对应的"概略化"轨迹,概略化轨迹单元是原始轨迹所穿过的格栅单元的中心点连线,因此其粒度比原始轨迹要大得多. 最后对概略化轨迹单元进行索引并建立 GSSTR-Tree.

GSSTR-Tree 的叶结点记录的格式为(*stu*, set(*tupleID*)),其中 *stu* 是概略化轨迹单元,set(*tupleID*)是其概略化轨迹包含 *stu* 的所有移动监控对象元组标识的集合(注意,多个移动监控对象的概略化轨迹可以共享同一个轨迹单元). 概略化轨迹的变化频率比原始轨迹低得多,因此 GSSTR-Tree 有效地降低了索引更新的频率.

在进行时空查询处理时,根结点首先检查 SAP-Table,看看哪些叶结点的管辖区域与查询地理区域相交,然后将查询发送给这些叶结点并行执行;叶结点在处理来自于根结点的查询请求时,可以通过本地的 S⁴ T-Tree 快速地得到查询结果:如果查询是针对静态监控对象的(如 *Q*4),则仅需查询 S⁴ T-Tree 的空间 R 树部分;如果查询针对移动监控对象(如 *Q*5),则需要根据时空约束条件对 S⁴ T-Tree 的 GSSTR-Tree 部分进行搜索;根结点收到各叶结点返回的结果之后,还需要进行必要的合并操作,并将最后的结果返回给查询用户.

全局时空查询处理算法如算法 2 所示.

算法 2. 全局时空查询处理算法.

输入:全局时空索引 GSSTR-Tree
时空查询 *Q*
输出:查询结果 *R*

```
1. georange=getgeo(Q);
2. timerange=getperiod(Q);
3. sites=retrievegeo(georange,GSSTR-Tree.SAP-Table);
4. FOR EACH site∈sites DO(IN PARALLEL)
5.    StaticMov=getobjtype(Q);
6.    IF StaticMov=Static      //static objects
7.       sr_tree=get_tree(site,SR-Tree);
8.       tupleids=retrivegeo(georange, sr_tree);
9.    ELSE                     //moving objects
10.      gsstr_tree=get_tree(site,GSSTR-Tree);
```

11. *tupleids*=*retrivegeotempo*(*georange*, *timerange*, *gsstr_tree*);
12. ENDIF;
13. *res*=*evaluate*(*Q*, *tupleids*);
14. *sendMaster*(*res*);
15. ENDFOR;
16. 根节点收集各叶结点发来的查询结果并存入 *R*;
17. 根节点对 *R* 中的结果按照 ObjID 进行合并;
18. Return(*R*).

在算法 2 中，函数 *getgeo*(*Q*)和 *getperiod*(*Q*)分别返回*Q*的查询地理区域和时间区域；*getobjtype*(*Q*)返回*Q*所查询的监控对象类型，*retrievegeo*(*georange*, *ds*)和 *retrievegeotempo*(*georange*, *timerange*, *ds*)分别根据地理区域 *georange* 和时空区域在相关数据结构 *ds* 中查询，得到的结果为一组指针.

5.2.3　其它查询类型的处理

对于属性约束条件查询(如查询 *Q*3)，尽管不是关键字查询，但是 IoT-ClusterDB 仍然可以通过 GFTKB$^+$-Tree 中的 GKR-Table 和 KSMB$^+$-Tree 来缩小需要查询的叶结点范围，然后再在各相关叶结点上执行对应的 SQL 语句. 在执行 SQL 语句时，叶结点不能调用本地的 LFTKB$^+$-Tree，但通过属性 B$^+$ 树仍然可以获得较快的查询速度.

对于更加复杂的查询(如多个约束条件通过 AND 或者 OR 连接)，IoT-ClusterDB 可以通过相应的全局索引，针对每个约束条件获得一个叶结点集合，然后根据连接条件对这些集合进行相应的交集或并集计算，最终获得真正需要执行查询的叶结点范围，并在这些叶结点上执行查询从而得到最后的查询结果.

5.3　新采样数据触发的数据更新处理

在 IoT-ClusterDB 中包含多个采样数据接收服务器，这些服务器组成一个集群，共同对传感器接入处理器上传的新采样值进行处理.

新采样值的数据格式为(*objID*, *staticMov*, *svalue*)，其中 *objID* 是监控对象的标识，*staticMov* 标明监控对象是静止的还是移动的，*svalue*=(*t*, *loc*, *npos*, *schema*, *value*)是一个 *samplingValue* 类型的值. 采样数据接收服务器接收到上述数据之后，首先判断 *staticMov*，如果是静止监控对象，则可通过 SAP-Table 找到管辖区域与 *loc* 相交的叶结点(*loc*∈*Region* 时可以有多个叶结点与 *loc* 相交)，然后将采样数据转发给相应的叶结点进行存储处理；如果是移动监控对象，则需要进一步判断上次提交了采样值之后该监控对象是否跨越了叶结点的管辖区域边界，若是，则需要进行相应的插值计算，并将插值结果也发送给相应的叶结点. 为此，采样数据接收服务器需要保存各移动监控对象上一次提交的采样数据值.

叶结点接收到新的采样数据之后，直接通过 samplingAppend 操作将之附加到相应监控对象的“Samplings”属性值中.

6　系统实现及性能分析

本文提出的 IoT-ClusterDB 系统框架已经在 PostgreSQL 8.2.4(带有空间数据扩展模块 PostGIS 1.3.1)的基础上进行了编码实现. PostgreSQL 是一个开源的对象关系型数据库系统，支持数据库内核一级的数据类型、查询操作以及索引的扩充. 此外，我们还在 PostgreSQL 的基础上实现了相应的全局索引机制，并在此基础上实现了一个全局查询处理器，使得多个 PostgreSQL 服务器可以组合在一起，形成一个协同工作的 IoT-ClusterDB 集群系统.

为了对 IoT-ClusterDB 的性能进行比较与分析，我们在上述原型系统的基础上进行了实验. 实验中的传感器数据采用第 3.1 节中描述的数据库模式进行组织，数据包括两个部分：

(1) 移动传感器数据是在北京 1 万个出租车所采集的真实 GPS 数据的基础上，通过一个数据模拟程序生成的，该模拟程序通过对原始 GPS 采样数据的插值和倍增，可以动态模拟 12 800～32 000 个移动传感器的数据采样；

(2) 静止传感器数据是通过一个模拟程序随机生成的，该模拟程序可以模拟出 128 000～320 000 个静止传感器的动态数据采样.

将上述两部分数据按照一定的比例(如移动传感器与静止传感器的比例为 1∶10)进行合成，从而可以得到实验中所需要的总体数据集. 表 2 列出了实验中的主要参数.

表 2　模拟实验的主要参数

参数名	参数值(单位)	参数含义
$N_{MovSensors}$	12 800～32 000	移动传感器的数目
$N_{StaticSensors}$	128 000～320 000	静止传感器的数目
$N_{Sensors}$	140 800～352 000	传感器的总数目(其中移动传感器与静止传感器的比例为 1∶10)
$\xi_{Sampling}$	5 s	每个传感器进行原始数据采样的平均时间间隔
$\xi_{KeySampling}$	300 s	SCP 结点抽取的关键采样数据的平均时间间隔
$N_{MasterNodes}$	1	根结点服务器的数目
$N_{LeafNodes}$	2～32	叶结点服务器的数目

在实验中，我们重点针对 IoT-ClusterDB 的查询响应时间及加速比进行了分析. 由于在海量传感器采样数据的云存储以及时空数据处理方面，目前尚没有系统性的相关工作，我们将 IoT-ClusterDB 与单数据库结点处理机制(Single Node Query Processing，SNQP)进行了比较. 此外，为了分析传感器数据动态采集对查询性能所造成的影响，在实验的过程中，由数据模拟生成程序根据实验数据集，持续向 IoT-ClusterDB 发送并插入传感器采样数据.

实验的测试用例分别采用第 3.2.4 节中介绍的关键字查询 $Q1$、属性约束条件查询 $Q3$ 和时空约束条件 $Q5$. 图 9 给出了 IoT-ClusterDB 处理各个查询的响应时间(为了更好地分析属性约束条件查询 $Q3$ 的性能，我们在各叶结点中没有对相关属性建立局部索引).

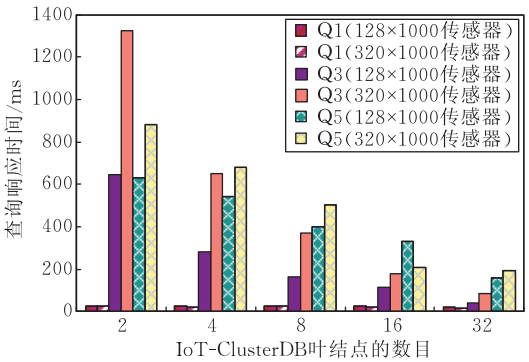


图 9 IoT-ClusterDB 的查询响应时间

从图 9 可以看出，IoT-ClusterDB 可以有效地支持关键字查询 $Q1$，且在不同的数据量及不同的叶结点规模的情况下，关键字查询的响应时间基本保持稳定. 这是因为 IoT-ClusterDB 通过 GFTKB$^+$-Tree 和 KSMB$^+$-Tree 对关键字进行了全局索引，因此数据量及结点规模的变化对查询响应时间的影响很小.

对于属性约束条件查询 $Q3$，查询响应时间明显地依赖于数据量及结点的规模——数据量越大则查询响应时间越长，且在同等数据量的情况下，查询响应时间随着结点规模的增大而减少. 这是因为在各叶结点上没有建立属性索引，因此查询的响应时间主要取决于各叶结点的平均数据量.

对于时空约束条件查询 $Q5$，其查询响应时间的总体变化趋势与 $Q3$ 相似，但其变化幅度不如 $Q3$ 明显，且查询性能总体上优于 $Q3$. 这是因为在 IoT-ClusterDB 中数据是按照地理空间属性进行分布的，根结点上的 SAP-Table 和各叶结点上的时空索引 S^{4}T-Tree 共同构成了一个全局时空索引，加快了时空查询处理的速度，并降低了叶结点数据量对查询性能的影响.

为了进一步分析 IoT-ClusterDB 中叶结点规模对非关键字查询处理性能的影响，我们在表 3 和表 4 中分别给出 IoT-ClusterDB 在处理 $Q3$ 和 $Q5$ 时的加速比实验结果. 加速比 $Speedup$ 定义为

$$Speedup=\frac{\xi_{\mathrm{SNQP}}}{\xi_{\mathrm{IoTClusterDB}}},$$

其中 ξ_{SNQP} 和 $\xi_{\mathrm{IoTClusterDB}}$ 分别是 SNQP 和 IoT-ClusterDB 的查询响应时间.

表 3 IoT-ClusterDB 在执行 $Q3$ 时的加速比

$N_{\mathrm{LeafNodes}}$	加速比	
	$N_{\mathrm{Sensors}}=64\times1000$	$N_{\mathrm{Sensors}}=160\times1000$
2	1.57	1.62
4	3.25	3.26
8	6.27	6.32
16	11.83	13.52
32	20.19	24.86

表 4 IoT-ClusterDB 在执行 $Q5$ 时的加速比

$N_{\mathrm{LeafNodes}}$	加速比	
	$N_{\mathrm{Sensors}}=64\times1000$	$N_{\mathrm{Sensors}}=160\times1000$
2	1.62	1.87
4	1.95	2.45
8	2.89	3.92
16	3.68	4.77
32	6.11	6.12

从表 3 可以看出，IoT-ClusterDB 在处理属性约束条件查询 $Q3$ 时，加速比随着叶结点个数的增加几乎呈线性增长，而且数据量越大，加速比增加得也越明显. 这是因为在 IoT-ClusterDB 中所有的数据均是统一存放在单个关系表中的，不牵涉到连接运算，因此查询的响应时间基本上取决于各叶结点的平均数据量. 此外，多个数据库结点的协同工作也分担了传感器数据更新所带来的繁重通信与计算开销.

从表 4 可以看出，IoT-ClusterDB 在处理时空约束条件查询 $Q5$ 时，加速比也随着叶结点个数的增加而随之增大，但与 $Q3$ 相比，$Q5$ 的加速比变化不如 $Q3$ 变化明显，这是由于全局时空索引的影响而导致的. 另外，轨迹的分割与合并也给全局查询处理带来了额外的计算开销.

系统的查询处理性能通常受到数据更新的影响. 为了测试传感器数据上传对系统查询处理性能

的影响，我们比较了打开数据更新和没有数据更新两种情况下的查询响应时间（每一轮实验均取 $Q1\sim Q5$ 五个查询的平均响应时间）. 由于和静态传感器相比，移动传感器的数据更新代价更大（因为需要动态维护 GSSTR-Tree），因此我们选择移动对象来进行测试. 同时，为了增加每个叶结点服务器的压力，在实验中我们选择相对较少的服务器数量（4～16个）. 图 10 给出了相关的实验结果.

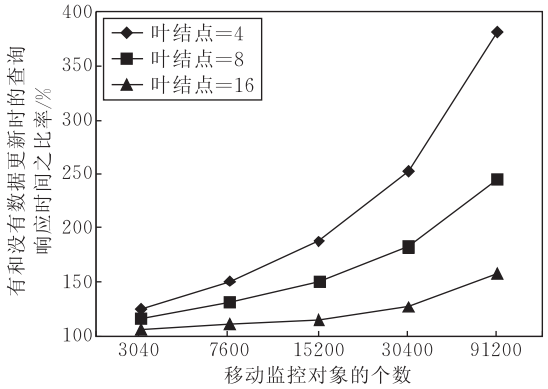


图 10　IoT-ClusterDB 中数据更新对查询响应时间的影响

从图 10 可以看出，IoT-ClusterDB 在叶结点数目一定的情况下，所接入的传感器数目越多，则采样数据更新对查询性能的影响越大. 此外，当传感器的数目一定时，随着叶结点数目的增长，采样数据更新对查询性能的影响也随之降低. 当叶结点的个数为16，所接入的传感器数目为 91 200 个时，系统的查询响应时间仅为没有数据更新时的 1.5 倍，仍然表现了良好的查询处理性能.

综合以上分析，IoT-ClusterDB 在海量传感器采样数据管理方面提供了良好的数据接入与查询处理性能，为物联网海量数据管理提供了一种可行的解决方案.

7　结　论

物联网打破了物理世界和数字世界的界限，将信息技术延伸到了物理世界和人类社会，是促使未来信息技术产业变革的关键性技术. 本文从物联网海量数据处理技术的角度出发，对物联网所提出的挑战及相关对策进行了分析，并提出了一种面向物联网海量传感器采样数据管理的数据库集群系统框架 IoT-ClusterDB. 本文的的主要创新点如下：

（1）提出了一种能够同时支持“键－值”查询和普通 SQL 查询的物联网数据库集群框架. 通过在关系数据库集群的基础上建立分布式的全局关键字索引及全局关键字查询处理机制，使得 IoT-ClusterDB 可以兼容多种查询类型的快速处理，突破了目前云数据管理技术主要针对“键-值”查询、并行数据库技术主要针对 SQL 查询的局限.

（2）提出了一种能够应对传感器采样数据的异构性、时空相关性和动态流式特性的传感器时空数据库模型. 通过采样数据序列等数据类型和相应的查询操作，在数据库内核一级实现了传感器采样数据的统一表示、存储、计算和查询，突破了目前时空数据库主要针对静态的空间数据或相对单一的时空数据、缺乏有效的异构数据流表示方法的局限.

（3）提出了一种地理区域敏感的传感器采样数据分布策略，并在此基础上提出了一种分布式的全局时空索引和全局时空查询处理方法，突破了目前时空数据库主要针对单机环境、缺乏时空数据库集群相关方法的局限.

物联网海量数据处理技术具有广阔的应用前景. 例如，通过电网中的传感器结点，可以获知电网中的电力变化及用户的用电规律，并进一步实现智能化的电力传输；通过湖泊中布设的传感器，可以对污染数据进行采样并进行异常情况的报警；通过在交通系统中设置的大量传感器，可以实现车联网和先进的智能交通控制与管理等.

随着研究的不断深入及技术的不断发展，可以想见，物联网必将在国民经济和人民生活中扮演重要的角色，并给人们的生活和工作方式带来深刻的变革.

致　谢　感谢中国科学院软件研究所的郭黎敏，她在本文实验中做了大量工作！

参　考　文　献

[1] Sarma S, Brock D L, Ashton K. MIT Auto ID WH-001: The Networked Physical World-Proposals for Engineering the Next Generation of Computing, Commerce & Automatic-Identification. Massachusetts: MIT Press, 2000

[2] Ye Tian-Chun, Huang Xiao-Gang, Wang Wen-Sheng et al. The Annual Blue Book on China's Development of Internet of Things Industry. Published by CIT-CHINA, 2010(in Chinese)
(叶甜春，黄晓刚，王文升等主编. 中国物联网产业发展年度蓝皮书(2010). 中国物联网研究发展中心，2010)

[3] Sundmaeker H, Guillemin P et al. Vision and Challenges for Realizing the Internet of Things. Luxemborg: Publications Office of the European Union, 2010

[4] Ning Huan-Sheng, Ning Na et al. Layered structure and management in Internet of Things//Proceedings of the Future Generation Communication and Network (FGCN). Jeju Island, Korea, 2007: 386-389

[5] Yan Lu, Zhang Yan, Yang Laurence T, Ning Huan-Shen. The Internet of Things: From RFID to the Next-Generation Pervasive Network Systems. New York: Auerbach Publications, 2008

[6] Giusto D, Iera A, Morabito G, Atzori L eds. The Internet of Things: 20th Tyrrhenian Workshop on Digital Communications. Germany: Springer, 2010

[7] Atzori L, Iera A, Morabito G. The Internet of Things: A survey. Computer Networks, 2010, 54(15): 1-19

[8] Tsiftes N, Dunkels A. A database in every sensor//Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems. Seattle, USA, 2011: 316-329

[9] Wu Ji, Zhou Yong-Luan, Aberer Karl, Tan Kian-Lee. Towards integrated and efficient scientific sensor data processing: A database approach//Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology. St. Petersburg, Russia, 2009: 922-933

[10] Madden S, Franklin M J, Hellerstein J M, Hong W. TinyDB: An acquisitional query processing system for sensor networks. ACM Transactions on Database Systems, 2005, 30(1): 122-173

[11] Gurgen L, Roncancio C, Labbé C et al. SStreaMWare: A service oriented middleware for heterogeneous sensor data management//Proceedings of the 5th International Conference on Pervasive Services (ICPS'08). Sorrento, Italy, 2008: 121-130

[12] Kim M, Lee J W, Ryou J C. COSMOS: A middleware for integrated data processing over heterogeneous sensor networks. ETRI Journal, 2008, 30(5): 696-706

[13] Yuriyama M, Kushida T. Sensor-Cloud infrastructure: Physical sensor management with virtualized sensors on cloud computing//Proceedings of the 13th International Conference on Network-Based Information Systems (NBiS). Takayama, Japan, 2010: 1-8

[14] Baldoni R, Cerocchi A, Lodi G et al. Designing highly available repositories for heterogeneous sensor data in open home automation systems//Proceedings of the 7th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS'09). Springer, Newport Beach, USA, 2009: 1-12

[15] Abadi D J. Data management in the cloud: Limitations and opportunities. IEEE Data Engineering Bulletin, 2009, 32(1): 1-10

[16] Sun Ning-Hui, Xu Zhi-Wei, Li Guo-Jie. Sea-Computing: A novel computation model for the Internet of Things. Communications of the China Computer Federation, 2010, 6(7): 52-56(in Chinese)
(孙凝晖, 徐志伟, 李国杰. 海计算: 物联网的新型计算模型. 中国计算机学会通讯, 2010, 6(7): 52-56)

[17] Chang F, Dean J, Ghemawat S et al. Bigtable: A distributed storage system for structured data//Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI'06). Seattle, USA, 2006: 205-208

[18] DeCandia G, Hastorun D, Jampani M et al. Dynamo: Amazon's highly available key-value store//Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP'2007). Washington, USA, 2007: 205-220

[19] Carstoiu D, Lepadatu E, Gaspar M. Hbase: Non SQL database, performances evaluation. International Journal of Advancements in Computing Technology, 2010, 2(5): 42-52

[20] Cooper B F, Ramakrishnan R, Srivastava U et al. PNUTS: Yahoo!'s hosted data serving platform//Proceedings of the 34th International Conference on Very Large Data Bases (VLDB'2008). Auckland, New Zealand, 2008: 1277-1288

[21] Thusoo A, Sarma J S, Jain N et al. Hive: A warehousing solution over a map-reduce framework//Proceedings of the 35th International Conference on Very Large Data Bases (VLDB'2009). Lyon, France, 2009

[22] Campbell D G, Kakivaya G, Ellis N. Extreme scale with full SQL language support in microsoft SQL Azure//Proceedings of the 29th ACM SIGMOD International Conference on Management of Data (SIGMOD'2010). Indiana, USA, 2010: 1021-1023

[23] Waas Florian M. Beyond conventional data warehousing: Massively parallel data processing with Greenplum Database//Proceedings of the 2nd International Workshop on Business Intelligence for the Real-Time Enterprise (BIRTE'2008). Auckland, New Zealand, 2008: 89-96

[24] Poess M, Nambiar R O. Large scale data warehouses on grid: Oracle database 10 g and HP proLiant systems//Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'2005). Trondheim, Norway, 2005: 1055-1066

[25] Yu Xu, Kostamaa P. A new algorithm for small-large table outer joins in parallel DBMS//Proceedings of the IEEE 26th International Conference on Data Engineering (ICDE'2010). Long Beach, USA, 2010: 1018-1024

[26] Kang K D, Basaran C. Adaptive data replication for load sharing in a sensor data center//Proceedings of the 29th IEEE International Conference on Distributed Computing Systems Workshops (ICDCS Workshops'09). Montreal, Canada, 2009: 20-25

[27] Güting R H, Böhlen M H, Erwig M, Jensen C S et al. A foundation for representing and querying moving objects. ACM Transactions on Database Systems, 2000, 25(1): 1-42

[28] Güting R H, Almeida V T, Ding Z. Modeling and querying moving objects in networks. VLDB Journal, 2006, 15(2): 165-190

[29] Chandrasekaran S, Cooper O, Deshpande A et al. TelegraphCQ: Continuous dataflow processing for an uncertain world//Proceedings of the 1st Biennial Conference on Innovative Data Systems Research (CIDR'2003). Asilomar, USA, 2003: 276-285

[30] Cranor C D, Johnson T, Spatscheck O, Shkapenyuk V. Gigascope: A stream database for network applications//Proceedings of the 22th ACM SIGMOD International Conference on Management of Data (SIGMOD'2003). San Diego, USA, 2003: 647-651

[31] Andrade H, Gedik B, Wu K-L, Yu P S. Scale-Up strategies for processing high-rate data streams in System S//Proceedings of the IEEE 25th International Conference on Data Engineering (ICDE'2009). Shanghai, China, 2009: 1375-1378

[32] Fusco F, Stoecklin M P, Vlachos M. Net-Fli: On-the-fly compression, archiving and indexing of streaming network traffic//Proceedings of the 36th International Conference on Very Large Data Bases (VLDB'2010). Singapore, 2010: 1382-1393

DING Zhi-Ming, born in 1966, Ph. D., professor, Ph. D. supervisor. His research interests include database and knowledge base systems, spatial-temporal databases, Internet of Things, cloud computing, and information retrieval.

GAO Xu, born in 1980, Ph. D. candidate, lecturer. His main research interests include database and knowledge base systems, and spatial-temporal databases.

Background

Nowadays, the Internet of Things (IoT) has become a hot research issue. However, the massive IoT data management problem has not been thoroughly studied, which becomes a bottleneck for the development of the IoT industry. In recent years, we have conducted continuous research, software development, and industrialization on IoT data management.

The work of this paper is supported by the National Natural Science Foundation of China (NSFC)'s key project "Internet-of-Things Technology and System for Sensing Unconventional Emergencies and for Emergency Commanding" (Grant No. 91124001) and the key direction project of Chinese Academy of Sciences "Research and Development of Cloud Storage and Retrieval Systems". This paper includes the research results of the above projects.