Distributed Storage and Parallel Processing in Large-scale Wireless Sensor Networks

Yuexuan Wang, Yongcai Wang
Institute for Interdisciplinary Information Sciences (IIIS)
Tsinghua University, Beijing, China,
E-mail: {wangyuexuan,wangyc}@mail.tsinghua.edu.cn

Abstract. A Large-scale Wireless Sensor Network (LWSN). such as an environment monitoring system deployed in a city, could yield data on the order of petabytes each year. Storage and computation of such vast quantities of data pose difficult challenges to the LWSN, particularly because sensors are highly constrained by their scarce resources. Distributed storage and parallel processing are solutions that deal with the massive amount of data by utilizing the collective computational power of the large number of sensors, all while keeping inter-node communication minimized to save energy. In this chapter, we conduct a survey on the state-of-the-art of distributed storage and parallel processing in LWSNs. We will focus on the LWSN scenario in which vast amounts of data are collected prior to intensive computation on that data. We argue that current research results in this direction fall into three categories: 1) hierarchical system architecture to support parallel and distributed computation; 2) distributed data aggregation and storage strategies; 3) parallel processing, scheduling and programming methods. We highlight some important results for each of these categories and discuss existing problems and future directions.

Introduction

Recent advances in wireless communication, embedded computation, and Micro-electromechanical Systems (MEMS) technologies have enabled large-scale wireless sensor networks (LWSN) to come to fruition. More and more, LWSNs are changing the way that people cognize the physical world, and they will be an infrastructure for future Cyber-Physical Systems (CPS)[1]. The concept of LWSNs was described in [2, 3]: *a LWSN is a sensor network that contains thousands, or even tens of thousands of small sensors, which are distributed over a vast field to obtain fine-grained, high-precision sensing data.* LWSNs are used in many areas [4], ranging from ecological and precision agriculture information monitoring to intrusion detection, surveillance, and structural integrity monitoring etc. However, sensors in LWSNs are typically resource-limited; they are powered by batteries and can only communicate with neighbors using short-range radio. For the purpose of massive data collection and processing, an important problem to address in LWSN is how to store data and subsequently conduct computation in a distributed fashion and in parallel.

In practice, a LWSN could yield data in amounts approaching or exceeding petabytes each year. The storage, query and computation of such amounts of data by resource-limited sensors are a highly challenging tasks. If the (massive amount of) data is processed centrally, all data needs to be transmitted to a central server using multi-hop transmissions. In such cases, a LWSN will suffer high communication costs. More concretely, the energy complexity for data collection is $O(m+n\log n)$ [5], where n is the number of nodes and m is the number of links.

When *m* and *n* are large, sensors will deplete energy quickly thus limiting the lifetime of the network. A promising solution is to exploit the advantages of the distributed storage and parallel processing capabilities of a LWSN. Instead of transmitting data to a central server, data is stored and processed in-network. This can markedly reduce the communication costs. The intensive computational task is decomposed into many small tasks, each of which is affordable to a single sensor, and computation is executed in parallel by the distributed sensors. Using such an approach, the "swarm intelligence" of the distributed sensors can be employed, computing by utilizing the cooperation of the resource-limited sensors.

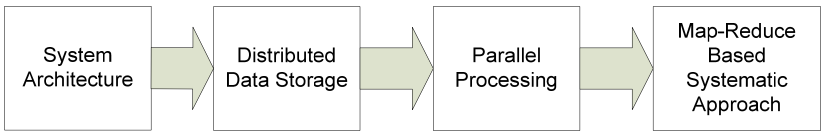


Figure 1. Summary of the research problems and their relationships in distributed and parallel computing in the LWSN.

The distributed storage and parallel computing problem can be decomposed into four sub-problems, each of which has attracted myriad research in recent years. Fig.1 summarizes these sub-problems and their relationships.

- The fundamental problem with distributed storage and parallel computing is system architecture design [2, 3, 6, 7], which concentrates on energy efficient topology control and sensor management. Both distributed storage and parallel processing depend heavily on the type of network organization (topology, architecture, etc.)
- *Distributed data storage.* Researchers have treated the sensor network as a distributed database, which has triggered a series of studies on distributed storage, indexing and querying [14-15, 17-19]. Distributed data storage not only saves the cost of data collection and transmission, but also forms the foundation for parallel processing as distributed data storage is the general basis for computation task decomposition.
- T*he parallel processing methods.* Parallel processing concentrates on how to decompose a computationally intensive task into many "small-sized" tasks and how to execute these tasks on the distributed sensors in parallel. Minimizing the communication cost and the execution time is the general consideration in the design of the parallel processing. Existing parallel processing methods include parallel fusion, parallel Kalman filter, etc., which were mainly designed in a case by case basis to meet the requirements of specific applications.
- *The Map-Reduction parallel programming model* was proposed for LWSNs most recently, in order to provide a systematic solution for parallel processing. It aims at providing a general Map-Reduce framework for various applications.

In this paper, we summarize the state-of-art of distributed and parallel computing in LWSNs from the above four categories. Our survey is different from the previous survey papers [36] and [37] in several ways. First, our survey presents a bottom-up summary of the distributed storage and parallel processing problem via the aforementioned categories of 1) fundamental architecture; 2) distributed storage and indexing; 3) task decomposition and parallel processing; and finally 4) the map-reduction based systematic approach. The relationships among these four aspects are also analyzed, while [37] only gives a summary of the in-network processing methods in LWSN and [36] focuses just on the middleware for supporting distributed services in sensor networks. In both [36] and [37], the studies of the relationship between the distributed storage and parallel processing were not discussed. This chapter reviews existing results and analyzes their relationships.

This chapter is organized as follows. In Section 1, the motivation and desired properties of distributed storage and parallel processing are introduced by investigating the complexity of the LWSN. In Section 2, the system architecture related to distributed storage and parallel processing are presented. In Section 3, distributed storage, indexing, querying and in-network data aggregation results are reviewed and analyzed. In Section 4, task decomposition and parallel processing algorithms are presented. The Map-reduction based systematic programming model is also introduced. In Section 5, we evaluate the existing problems and discuss future directions.

1. Motivation of Distributed Storage and Parallel Processing in LWSNs

1.1 Desired Properties of LWSNs

Scalability and longevity are the main challenges in large-scale wireless sensor networks. Let's consider a LWSN deployed for agriculture information monitoring. Its coverage area includes up to thousands of hectares, and its expected lifetime is at least several seasons--consistent with the growth periods of the crops. Sensors collect data periodically and report to a remote center, which are common scenarios in many applications of LWSNs. In these applications, the desired properties of the LWSN can be summarized as:

1. **Scalability:** The data collection and processing algorithm should scale to the size of the network.
2. **Longevity:** Wireless sensors have extreme energy limitations, but are required to work for a long period of time.
3. **Self-Adaptivity:** The sensor network systems must be self-adapting to the network and environmental dynamics, such as node failures, weather or environment changes, etc.
4. **Massive Data Processing Capability:** The sensor network needs to aggregate and process massive data of various kinds.

1.2 The Complexities of the Centralized Method

Some studies have focused on how the above desired functionalities can be achieved by a LWSN using the centralized method. This problem is investigated by studying the time and energy complexity of sensor networks [5]. Some recent results have shown that the network's lifetime is highly dependent on the scale of the network.

1. The energy complexity for data collection in a network is $O(m+nlogn)$, where n is the number of nodes and m is the number of links. Minimizing the energy is equivalent to finding shortest paths from the sink to all other nodes in the network.
2. For the data aggregation problem, total message complexity using any tree T is $\Theta(n)$, where n is the number of nodes of the network.
3. For data selection in a sensor network, any deterministic distributed algorithm needs at least $\Omega(\Delta+Dlog_D N)$ time to find the median of all data items, where D is diameter of the graph and Δ is the maximum degree of the network[5].

The above results show that a LWSN will suffer high energy usages and message complexities when it collects and processes data in a centralized way. Centralized algorithms for LWSNs lack scalability. In addition, the network lifetime is short when a LWSN is processing massive data in a centralized way. Furthermore, the lack of self-adaptation is also a big issue with centralized algorithms. Because global information and global coordination are required by centralized computing, centralized algorithms have difficulty adapting to dynamic changes in a network.

1.3 Distributed Computing Will Be the Solution

Distributed computing extends the traditional centralized computing by allowing computational components and data to be distributed across a network and seamlessly interoperate with each other to perform a task [8]. In LWSNs, distributed computing has many attractive characteristics: it is scalable, self-adapting, easy-to-implement, and recoverable. Distributed data storage can drastically reduce communication costs as multi-hop data transmission is avoided. Parallel processing not only reduces the computation burden of an individual sensor, but also utilizes the joint power of the great number of sensors in order to deal with the intensive computation task efficiently. Further, distributed computing lends itself to self-adaptation to environment dynamics. In distributed computing, each sensor conducts computing based on localized information. When the network topology changes, such as when a node joins or leaves, a sensor moves, or the environment changes, sensors can quickly respond to the changes by local computation, without global information acquisition. Therefore, distributed computing and parallel processing offer an effective solution to scalability, longevity and adaptivity problems of LWSNs [9].

2. System Architecture for Distributed Computing

2.1 Hierarchical LWSNs are More Scalable than Flat Ones

System architecture describes how sensors are organized to cooperate and how data is transmitted in the network; this is the foundation of distributed storage and parallel processing. In simple terms, the architecture of a sensor network can be flat or hierarchical. In a flat architecture, all sensors are treated as equivalent peers that communicate in an ad-hoc manner. On the other hand, in a hierarchical architecture, sensors are organized into clusters and can form multiple tiers. Each cluster is composed of a set of cluster members and cluster members are managed by the cluster head. The cluster head can be a homogeneous node or more powerful heterogeneous nodes [2, 6]. When the cluster heads form a high-level new cluster, a multi-tiered architecture is formed. A comparison of the flat and hierarchical architecture in ad-hoc wireless sensor networks is presented in [10]. The flat architecture generally generates a highly connected network, suffering from scalability issues, i.e. the number of connections grows $O(n^2)$ with the number of nodes n, making it impractical for the large LWSNs. In contrast, the distributed nature of a hierarchical network is effective for distributed storage and parallel processing for LWSNs. The reasons for the suitability of a hierarchical network are two-fold: 1) In a tiered architecture, sensors transmit data only to the local cluster head, which efficiently reduces multi-hop data communication complexity; 2) sensors in the sensing field are organized into small size subnets in the tiered architecture, and are connected by cluster heads, which forms a proper architecture for task decomposition and parallel processing. The interoperations between the subnets during parallel processing can be carried out by the cluster heads.

2.2 Hierarchical Architecture is the Foundation for Distributed Computing

The hierarchical architecture can be categorized into 2-tiered, 3-tiered and many-tiered architectures. Generally, increasing the number of tiers makes the network more scalable. For further evaluation of network architectures, *network capacity* [3, 7] is used to evaluate whether a network architecture is efficient or not. *Network capacity* is defined as the data collection rate at the sink. Studies in [3, 7] showed that both the 2-tiered and 3-tiered network architectures can increase the system capacities compared to the flat architecture, and that this improvement can be independent from the routing protocols used.

In addition to the improvement of capacity, a hierarchical architecture provides efficiency for distributed storage and parallel processing. In [11], a *motes-master* type two-tier 'Tenet' network is proposed, where cluster heads are called *master* and cluster members are called *motes*. Multimode data fusion is implemented in the master tier, and local data capturing and processing are carried out by the motes. The approach emphasizes local processing of data to avoid sending large sequences of continuously collected data to upper layers. This method of data fusion improves the network's capacity and lifetime efficiently. In [12], a three-tiered network architecture is proposed for distributed storage and parallel processing. A general three-tier architecture for distributed computing in LWSNs is illustrated in Figure 2. In the figure, nodes in different tiers are heterogeneous devices; although they can also be homogeneous. In general, increasing the number of tiers will increase the scalability performance of the network at the expense of requiring more time for data aggregation.

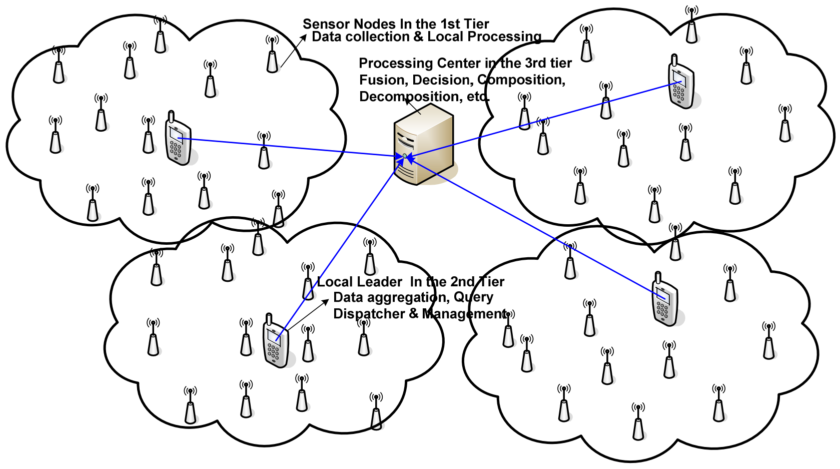

Sensor Nodes In the 1st Tier
Data collection & Local Processing
Processing Center in the 3rd tier
Fusion, Decision, Composition,
Decomposition, etc.
Local Leader In the 2nd Tier
Data aggregation, Query
Dispatcher & Management

Figure 2. Hierarchical architecture for a wireless sensor network

Figure 2 shows a general, three-tiered network architecture of a LWSN. An example of how the distributed data storage and parallel processing is carried out by the nodes in different tiers is also illustrated. In this example,

- The sensor nodes in the first tier implement the tasks of data capturing, local processing, and data transmission to the local leader.
- The local leaders in the second tier carry out the tasks of data aggregation, data transmission, query dispatching, and local sensor management.
- The central server in the highest decision making tier implements information fusion, task decomposition, and assembly of the processed results, etc.

Hierarchical architecture of a LWSN not only improves the network's scalability and capacity, but also offers a convenient network structure for distributed storage and parallel processing.

3. Distributed Data Storage and Query in LWSN

In distributed data storage and query, a LWSN is treated as a distributed database. It is different from traditional distributed databases because the *data-centric* feature is a cornerstone in LWSNs.

First, transmitting all the data to a central server is not only energy intensive, but also not necessary in many applications in LWSNs. For example, in environment monitoring, users may care about only the average temperature of a building or whether the specific temperature values of several buildings are abnormal or not. For the first case, the application requirement will be satisfied if the local leaders send the local averaging to the central server. For the second case, the application requirement will be satisfied if the central server issues a query with spatial attributes, only needing the sensors in the particular spatial region to respond to the query. It is clear the application requirements in these circumstances can be satisfied by distributed data storage and query.

Second, data plays the leading role in distributed storage and query, known as the *data-centric* feature of the wireless sensor networks. This can be seen from the aforementioned examples; the applications care only about the querying result and not about how/where data is stored or transmitted. This provides an extreme level of flexibility for the design of distributed data storage and indexing algorithms.

3.1 Distributed Data Storage

By means of the flexibility introduced by the *data-centric* feature, sensors can have different choices for distributing the stored data. A sensor can save data into its own memory, send the data to a local leader to store or send the data to higher tier leaders. The problem of *where to save* is generally a trade-off problem between energy efficiency and access complexity. Storing data locally will have no communication cost during storage, but is inefficient for querying. On the other hand, while data stored in a local leader or in higher tier leaders leads to efficient data querying, the communication costs increase for remote storage. For data centric storage, after an event is detected, data is stored by name (i.e., at a storage node that need not be the same as the detecting node) within the sensor net. How queries can find data and how data can be transmitted to the central server can be handled by routing protocols such as Directed Diffusion [16]. If the query has geographic information embedded in it, the central server sends interests to specific locations; otherwise, the query is flooded throughout the network. When the data that matches the interest is found, it is sent to the central server along the reverse path of the query.

We consider the case of distributed data storage in the hierarchical architecture without indexing, and the case of data centric storage using data indexing.

Local Storage (LS)

When data is stored in local memory, it costs O(n) for a query to find the distributively stored data, where n is the number of sensors and O(m) transmissions to transmit the queried result to the central sever, where m is the number of tiers. Since m<<n, the overall complexity for storing and querying a piece of data is O(n) .

Hierarchical Storage without Indexing (HS)

If data is stored in the second tier leader, it costs O($\sqrt{n/n_c}$) communications for data storage, where n_c is the number of the second tier leaders. This is because the communication complexity for storing each piece of data is of the order of the diameter of the local cluster. If data is stored without indexing, it costs O(n) for a query to find the distributively stored data. Thus, the overall complexity for storing and querying a piece of data is also O(n) .

Data Centric Storage with Indexing (DCS) [14, 15]

In DCS, the communication cost to store the event is O($\sqrt{n}$). Queries are directed to the node that stores events of that name, after which a response is returned; both of these occur at a cost of O($\sqrt{n}$) because the communication complexity is of the order of the diameter of the network. The overall complexity for storing and querying a piece of data under this scenario is therefore O($\sqrt{n}$) .

This shows that DCS with indexing has the lowest communication complexity in data storage and querying. Without data indexing, the query complexity in LS and HS is O(n), i.e., if the queries are frequent, the sensors in the network consume energy quickly due to the complexity of data searching. Therefore, when the network scale is large, data indexing is necessary to reduce the complexity of the distributed query.

3.2 Distributed Data Indexing

Distributed indexing aims to reduce the complexity of a data search. The data-centric storage (DCS) approach, proposed in [14, 15, 17], avoids the flooding of queries. The events or the data are named, and then stored at a network location based on that name. The name of the data provides a logical rendezvous mechanism between data and queries.

Geographic Hash Table (GHT)

Authors in [15] implement a specific DCS by hashing a key *k* into geographic coordinates. Thus, queries for data of a certain type are likely to be satisfied by a small number of nodes, significantly improving the speed of query responses.

Distributed Index for Features (DIFS)

DIFS [18] proposes a spatially distributed index for efficient indexing and range-based search. DIFS provides load-balanced communication over indexed nodes by using the governing property that the wider the spatial extent known to an index node, the more constrained the value range covered by that node.

Distributed Index for Multidimensional data (DIM)

Distributed index scheme for multidimensional data (DIM) is proposed in [19] to support range queries. DIM uses the idea that events whose attribute values are *close* are likely to be stored at the same or nearby nodes. The events are hashed to "locations" using the above idea. GPSR (Greedy Perimeter Stateless Routing [20]) routing is then applied to route events, and queries to the corresponding nodes in a distributed fashion.

Adaptive Ring-based Index (ARI)

The Adaptive Ring-based Index (ARI) scheme was proposed in [21] to facilitate data dissemination in large scale WSNs. The sensing data is collected, processed and stored at the nodes close to the detecting nodes, and the location information of these storing nodes is pushed to certain number of index nodes, which then act as the rendezvous points for sinks and sources. The index nodes for the same event type are connected via a set of forwarding nodes to form an "index ring". The number and locations of the index nodes on an index ring, as well as the shape of the ring, can be adaptively changed to achieve load balance and optimize system performance. A lazy index updating (LIU) and a lazy index query (LIQ) mechanism are proposed to reduce the overhead of index updating and index querying, and hence improve overall system performance.

Connected dominating set Based Indexing(CBI)

Connected dominating set Based Indexing (CBI) was proposed in [22] to support scalable handling of large amounts of sensing data in LWSNs. In CBI sensing data is collected and stored at storage nodes, which form a *k*-hop dominating set of the whole network. Meanwhile, the information of high level, semantically rich data is stored and maintained at the index nodes formed by a connected *m*-hop dominating set. This leads to the ability of queries to only have to be routed to the appropriate index nodes instead of flooding into the whole network.

3.3 Distributed In-network Query

A sensor network can be seen as a large, distributed database where users can access data through queries [23, 24] by accessing the data using a query-like declarative specification (such as SQL). By in-network queries, laborious data collection efforts can be avoided allowing the LWSN to be more energy efficient. In addition, data accessing via query is more user-friendly. In-network querying is based on distributed data storage and distributed indexing.

COUGAR System

The first SQL-like in-network query system for sensor networks was proposed by the Cougar project[17]. This is based on the idea that local computing can reduce unnecessary traffic in sensor networks. Specifically, a new layer is introduced between the application and network layers, which is called *query layer*. The query layer on each sensor consists of a query proxy, which interacts with the application as well as the network layer. A query optimizer is located on a set of gateway nodes. This query optimizer generates query plans that determine which nodes are involved in routing (the data flow) and which computations need to be performed at which node. Each query has a leader node that performs the main computational task. Thus, each query generates two computation plans: one plan for the leader node to perform the main processing, and one plan that is disseminated to all relevant nodes. The COUGAR platform was implemented on Mica Mote and Sensorial demos.

TinyDB System

TinyDB [25] is a query processor implementation running on top of the TinyOS [26] operating system. By taking advantage of a user-friendly interface, remote users are able to easily query the WSN using the appropriate SQL syntax. TinyDB has some design aspects similar to COUGAR: TinyDB also manages a metadata catalogue to provide the information about the kind of readings available in the sensor network. Unlike COUGAR though, TinyDB allows multiple queries to be run on the same group of sensor nodes at the same time. TinyDB optimizes the query processing schedule at the base station, and then disseminates the queries into the network. TinyDB performs neighbor tracking and routing table maintenance.

TINA: Temporal coherency-aware query

TINA [27] is an enhancement over TinyDB as it introduces temporal coherency tolerance to reduce the amount of data transmissions. The main idea is to send the readings of sensor nodes only in those cases where that reading differs from the last recorded reading by more than some stated tolerance. In TINA the transmission interval is therefore dynamic. This technique is also known as DELTA [28] transmission. In TinyDB every reading is transmitted periodically, regardless of how close that reading was to the prior one.

HQP: Hybrid Query Processing

Hybrid query processing was proposed in [27], which considers both continuous queries and ad-hoc queries. The query performances considered by HQP are not only in regards to data traffic but also the overhead of query dissemination. HQP introduces a query caching scheme. When a continuous query is executed, its results are stored in the cache of each node. When an ad-hoc query is raised, the cached data can help reduce the query overhead and data traffic.

Overall, to deal with the communication complexity of centralized data storage, distributed data storage, indexing and querying methods were proposed. In distributed data storage, distributed indexing mechanisms are necessary for reducing the communication complexity of queries. Data features are abstracted to build the index, and data with similar features can be stored in close spatial proximity. The abstracting and indexing methods also provide an important foundation for in-network data query systems. The main results in distributed storage, indexing and querying are listed in Table 1.

Table 1. Summary of distributed data storage and query

Category	Solutions	Properties	References
Storage in hierarchical network	Local storage	O(1) cost in storage; O(n) for query	
	Hierarchical storage without indexing	O($\sqrt{n/n_c}$) cost in storage; O(n) for query	
	Data centric storage with indexing	O($\sqrt{n}$) for storage; O($\sqrt{n}$) for query,	[14, 15]
Distributed Indexing	Geographic Hash Table (GHT)	Hashing a key k into geographic coordinates	[15]
	Distributed Index for Features (DIFS)	Spatially distributed index for range queries	[18]
	Distributed Index for Multidimensional data (DIM)	Distributed index for multidimensional data to support range queries	[19]
	Adaptive Ring-based Index (ARI)	The index nodes for the same event type are connected to form an index ring	[21]
	Connected dominating set Based Indexing(CBI)	Storage nodes form a k-hop dominating set of the network; Indexing nodes form a m-hop dominating set on the storage nodes.	[22]
In-Network query	COUGAR System	The first SQL-like in-network query; Query optimizer	[17]
	TinyDB System	On TinyOS; SQL syntax; allows simultaneous multiple queries;	[25]
	TINA: Temporal coherency-aware query	Introduced temporal coherency tolerance	[27] [28]
	HQP: Hybrid query processing	Considers both continuous query and ad-hoc query	[27]

4. Parallel Processing in LWSN

In addition to data collection and querying, wireless sensor networks can execute much more complex computations by exploiting the parallel processing capabilities of the sensor nodes. Numerous researchers have studied embedded engineering algorithms, such as autoregressive model fitting, wavelet transforms, and fast Fourier transforms within the computational core of a network of wireless sensors [29]. Traditionally, such complex computations are executed by a central server after full collection of all the data from the sensors over the lifetime of the network. However, such centralized approaches are commonly communication intensive, limiting the lifetime of the network as communication is the most energy consuming operation in these kinds of networks. Parallel processing in LWSNs can offer several distinct improvements over traditional centralized methods:

1) *Decomposition of complex computational tasks into sensor affordable tasks.* Because wireless sensors are deployed in ad-hoc networks featuring peer-to-peer communication, many analytical routines can be easily decentralized and distributed across a large number of wireless nodes with individual processing capabilities.

2) *Reduction of the communication cost.* Parallel processing in LWSNs processes the raw data taken up to a given period of time locally at each sensor and transmits only the results of these computations to the central server, which are typically only a fraction of the size of raw data itself. Therefore, the amount of required communication within a sensing network is reduced, and power consumption and network bandwidth problems are greatly mitigated.

3) *Leverage of distributed computation intelligence.* Although the processing capabilities of the individual sensors are limited, the number of sensors is massive. Leveraging the distributed intelligence of a great many sensors can be used to solve difficult computational tasks.

As a result, researchers have begun to look at various parallel processing techniques for distributed data processing on wireless sensing networks.

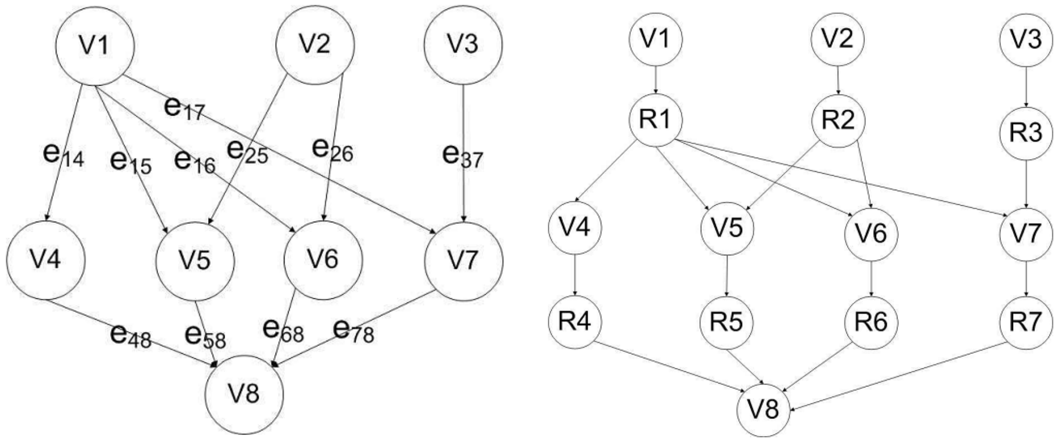
4.1 Task Mapping and Scheduling In Parallel Processing

Task mapping and scheduling [30, 31] plays an essential role in parallel processing, which solves the following problems, subject to certain design objectives: assign tasks to sensors, the execution sequence of those tasks, and the communication schedule between sensors.

Localized task mapping and scheduling is more suitable for a LWSN, because it is hard for a central server to maintain awareness of the network status. In localized task mapping and scheduling, solutions focus on the performance optimization in the hierarchical network architecture.

In [30], an Energy-constrained Task Mapping and Scheduling (*EcoMapS*) method for clustering sensor networks was proposed. It aims to map and schedule tasks of an application with the minimum schedule length subject to energy consumption constraints. EcoMapS is based on the high-level application model that describes task dependencies through Directed Acyclic Graphs (DAG). A DAG is described by $T = (V,E)$, which consists a set of vertices V representing the tasks to be executed and a set of directed edges E representing the dependencies among the tasks. If an edge e_{ij} exists between vertex i and j, v_i is called the *immediate predecessor* of v_j , and v_j is called the *immediate successor* of v_i. An immediate successor v_j depends on all its immediate predecessors such that v_j cannot start execution before it receives results from all of its immediate predecessors. An example showing these kinds of task dependencies is given in Figure 3(a). The weight of a task is represented by the number of CPU clock cycles required to execute the task.

In sensor networks, if a task v_j scheduled on one node depends on a task v_i scheduled on another node, then communication between these two nodes is required. The communication events between computational tasks are explicitly represented by extending the task DAG graphs to hyper-DAG graphs. A node R_i is added between node v_j and v_i to model the communication dependency. R_i represents the communication task to send the result of v_i to its immediate successors in the DAG. An example is shown in Fig3(b) in which the DAG in Figure 3(a) is transformed into a Hyper-DAG to model the task dependencies and the communication dependencies. The wireless channel is modeled as a virtual node *C* that executes one communication task at any given time instance. Hence, a cluster can be modeled as a star-network where all sensors only have connections with the virtual node *C*.



(a) An Example DAG (b) Hyper-DAG Extension of (a)

Figure. 3. DAG and Hyper-DAG Examples [30]

Based on the Hyper-DAG model, *EcoMapS was* proposed to map and schedule tasks. *EcoMapS* has two phases: the initialization phase and the quick recovery phase. In the initialization phase, the authors propose a scheduling algorithm based on the wireless channel model and the Hyper-DAG representation of the applications. The communication scheduling is embedded in the scheduling algorithm to satisfy the *Dependency Constraint*. In case of sensor failure, the schedules generated in the initialization phase will be adjusted by the *Quick Recovery Algorithm*. With these algorithms, *EcoMapS* minimizes the schedule lengths under energy consumption constraints and enables energy efficient parallel scheduling in one-hop clustering networks.

4.2 Parallel Processing Applications in LWSN

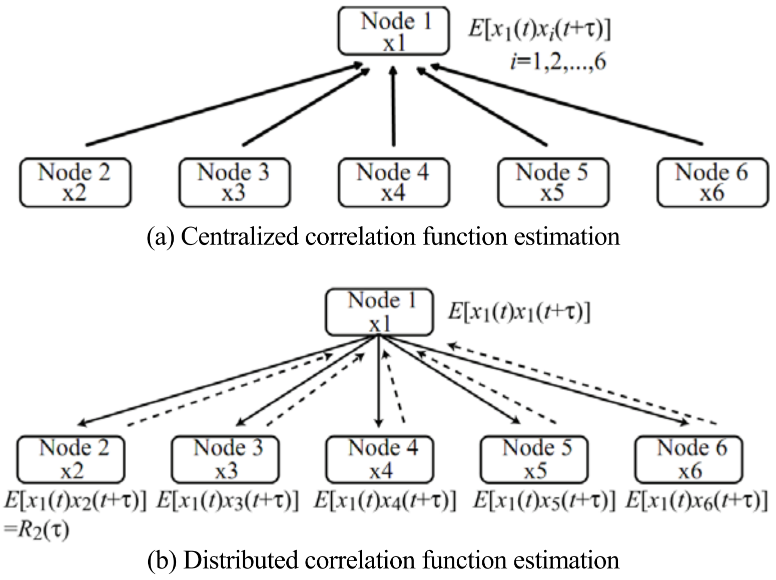
Parallel processing methods in LWSNs are mainly designed on a case by case basis for specific applications. However, the design criteria is similar in all these applications, i.e., reducing the inter-node communication while executing the computation tasks distributively and in parallel. Three representative applications are introduced in this chapter.

NetSHM for Structure Health Monitoring

Chintalapudi et al. [32] present a tiered system where data is processed distributively in-network by powerful gateway nodes; this system was designed for structural health monitoring (SHM), called netSHM. Structural health monitoring techniques detect and locate structural damages by measuring the response of a structure to ambient vibrations or forced excitation. Wireless sensor networks simplify the deployment of instrumentation, and can eliminate cabling costs of wired networks, but are challenged by both SHM's need for computationally intensive signal processing to detect and localize the structure damage and the need for the WSN to provide a user-friendly interface to engineers. To these ends, NetSHM allows structural engineers to program structure health monitoring applications in Matlab or C at a high level of abstraction without the need to understand the intricacies of wireless networking. At the same time, NetSHM leverages the hierarchical network to realize a novel functional decomposition for distributed and parallel signal processing. In NetSHM, applications are run on gateway nodes and low-tier motes are in charge of data collection and possible data processing. Low-tier motes transmit the raw or processed data to the relevant gateway node. While this method involves a top-down approach that offers a flexible and highly abstracted user interface, the computational capabilities of the prolific lower-tier motes are not fully utilized.

Correlation Function Estimation Using Parallel Processing

Correlation functions are an important data analysis tool used in a variety of applications that can be exploited to detect periodicities; measure time delay; locate disturbing sources; and identify propagation paths and velocities. In [33] a parallel correlation function estimation method was proposed in sensor networks. The communication complexity between the centralized and the distributed correlation function estimation are compared.



(a) Centralized correlation function estimation

(b) Distributed correlation function estimation

Figure 4. Comparison of Centralized and Distributed correlation function estimation

Figure 4(a) shows a case of a centralized correlation function estimation where node 1 works as a reference sensor. Assume n_s nodes, including the reference node, are measuring the structural response. Each node acquires data and sends the data to the reference node n_d times for averaging. After averaging, the inverse FFT (Fast Fourier Transform) is taken to calculate the correlation function, whereby this calculation is performed at the reference nodes. When the spectral density is estimated from discrete history records of length N, data transmitted through the radio from the sensors to the reference node is $N \times n_d \times (n_s - 1)$.

In Figure 4(b), data communication is reduced from that in Figure 4(a) by means of the distribution of computation. After the first measurement, the reference node broadcasts the time record to all sensors. On receiving the record, each node calculates the spectral density between its own data and the received record and stores the spectral density locally. Each node repeats this procedure n_d times. In this way, the spectral density is calculated by each node. Finally the inverse FFT is applied to the spectral density locally and the correlation function is sent back to the reference node. In this case, the total data to be transmitted is at most $N \times n_d + N \times (n_s - 1)$. Therefore, the communication complexity in parallel processing is

$O(N \times (n_d+n_s))$, much more efficient than the communication complexity $O(N \times n_d \times n_s)$ in the centralized method.

Automated Modal Parameter Estimation by Parallel Processing

In [34], three output-only methods for automated modal parameter estimation are proposed, including *peak picking*, *random decrement*, and *frequency domain decomposition*. They are implemented distributively by a network of wireless sensor prototypes. The software architecture is designed to emphasize parallel data processing and minimal communication so as to ensure scalability and power efficiency. The authors use an example to show that, using distributed methods, the number of messages in parallel processing can be reduced to almost 1/100 of those needed by the centralized method with *peak picking*, *random decrement*, and *frequency domain decomposition* methods.

We use parallel Peak Picking (PP) as an example to introduce the implementation of the parallel algorithms. In PP, the user first specifies the maximum number of peaks, p, that should be identified. Then, a consistent set of acceleration time history data is collected at each sensing node and is converted to frequency response functions (FRF) using an embedded version of the FFT algorithm. Each node picks the p largest peaks from its frequency response function by scanning for frequencies at which the value of the FRF is significantly and consistently higher than the value of the FRF at surrounding frequencies. Every wireless sensor communicates its identified p peaks to the central node. The central node can infer a subset of p modal frequencies or some reasonable number of fewer frequencies from the original PP data. Once the central node has determined a global set of peak frequencies, it can then share its findings of modal frequencies with the rest of the network. This sharing of data provides all wireless sensor nodes with mode shape information. A graphical representation of the implementation of the PP method on a distributed network of wireless sensors can be seen in Figure 5.

4.3 Map-Reduce based High Level Parallel Programming Model

Parallel processing methods are designed on a case by case basis depending on the specific application. In [35], the authors proposed a MapReduce based method to build a high-level parallel programming framework, aiming at generalizing the parallel programming mechanisms for different applications.

MapReduce implements parallel processing to deal with intensive computation tasks taking into account the hardware and software heterogeneity in a cluster. MapReduce enables applications with large processing power requirements to be split into smaller tasks that are executed in parallel on machines that are not occupied with some other task at the moment. The functions offered by MapReduce include a defined programming model that parallelizes user programs and a transparent fault-tolerance mechanism. In [35], a complete framework for processing massive amounts of sensor data by MapReduce is presented. Because the MapReduce model is not available by existing tiny sensors, the paper considered distributed data storage in different clusters, and studied MapReduce programming among cluster heads.

Four primitive tables were proposed to describe and implement distributed data storage: one for the sensor nodes, one for the sensor hardware, another for the sensor readings, and a final table for WSNs. These four tables can be generalized for different applications. The paper proposed a 2-dimensional tree that indexes the network in order to minimize the spatial search time. A spatial interpolation example was used to explain how the MapReduce parallel programming model can be implemented in LWSNs. In the example, the weights of a kriging function were calculated by firstly *Mapping* dataset and computation tasks to different clusters, and then *Reducing* is applied to give the final results. Full connections among the cluster heads are assumed in the paper. However, it remains an open problem of how to realize MapReduce programming in lower-tier nodes.

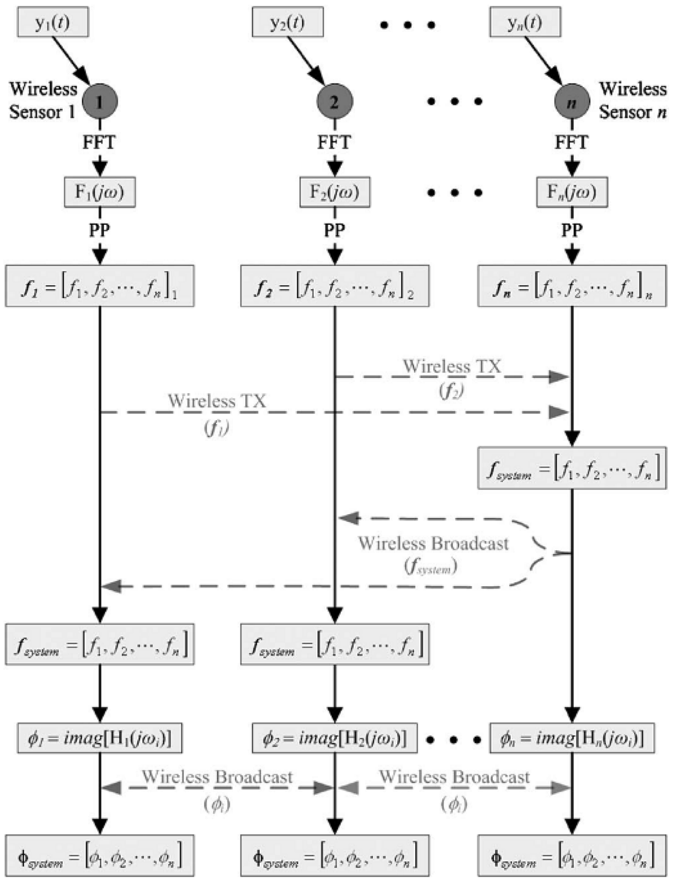

$y_1(t)$
$y_2(t)$
$y_n(t)$
Wireless Sensor 1
1
2
n
Wireless Sensor n
FFT
$F_1(j\omega)$
$F_2(j\omega)$
$F_n(j\omega)$
PP
$f_1 = [f_1, f_2, \cdots, f_n]_1$
$f_2 = [f_1, f_2, \cdots, f_n]_2$
$f_n = [f_1, f_2, \cdots, f_n]_n$
Wireless TX (f_2)
Wireless TX (f_1)
$f_{system} = [f_1, f_2, \cdots, f_n]$
Wireless Broadcast (f_{system})
$\phi_1 = imag[H_1(j\omega_i)]$
$\phi_2 = imag[H_2(j\omega_i)]$
$\phi_n = imag[H_n(j\omega_i)]$
Wireless Broadcast (ϕ_i)
$\Phi_{system} = [\phi_1, \phi_2, \cdots, \phi_n]$

Figure 5. Implementation of peak picking method on a network of wireless sensors [34]

5. Conclusion and Discussion

To deal with issues of scalability and intensive data processing in large scale wireless sensor networks, distributed storage and parallel processing mechanisms are proposed in the literature. These mechanisms implement the concepts of in-network data storage, querying and parallel processing. The distributed collective intelligence of a large numbers of sensors is exploited, and network energy is saved by reducing the communication complexity. In this chapter, we highlighted the following key issues:

- Hierarchical network architecture is a foundation for distributed data storage and parallel processing. A tiered network structure can not only increase a network's capacity and scalability, but also supports task decomposition and parallel processing.
- In distributed data storage, data-centric indexing plays a key role. When data is stored and indexed, a sensor network can be treated as a distributed database. SQL-like queries can thus be implemented to access each piece of data with communication complexity $O(\sqrt{n})$.

- For parallel processing in LWSNs, tasks need to be distributed to sensors and scheduled with considerations of latency minimization and energy efficiency. However, different tasks generally need unrelated decomposition methods, so parallel processing mechanisms are mainly studied on a case by case basis. The recently proposed MapReduce implements symmetrical programming techniques for parallel processing in LWSNs.

In the near future, LWSNs will be increasingly deployed in many different application areas. At the same time, among other improvements, the individual node will offer increased storage and processing capacity. More intensive computation tasks can be executed by a sensor network, and distributed storage and parallel processing will see rapid development in this area. An important future direction will be the emerging of distributed storage, distributed processing, and sensor fusion. Symmetrical methods for distributed storage and parallel processing will also be an important sphere to be explored and expanded upon. Programming models which make network operations transparent to the users are also desired and expected to be developed in the future.

Acknowledgement

We thank Nathaniel Hobbs for improving English presentations and Wolfgang Gentzsch and the anonymous reviewers for advices in improving the content of this chapter.

This work is supported in part by the National Basic Research Program of China Grant 2007CB807900, 2007CB807901, the National Natural Science Foundation of China Grant 61073174, 61033001, 61061130540, and the Hi-Tech research and Development Program of China Grant 2006AA10Z216.

References

[1]. Poovendran, R., Cyber-Physical Systems: Close Encounters Between Two Parallel Worlds. Proceedings of the Ieee, 2010. 98(8): p. 1363-1366.

[2]. Kim, T., et al., Hierarchical Network Protocol for Large Scale Wireless Sensor Networks. 2010 7th Ieee Consumer Communications and Networking Conference-Ccnc 2010, 2010: p. 1002-1003

[3]. Luo, H.Y., et al., TTDD: Two-tier data dissemination in large-scale wireless sensor networks. Wireless Networks, 2005. 11(1-2): p. 161-175.

[4]. Pei, Z.M., et al., Application-Oriented Wireless Sensor Network Communication Protocols and Hardware Platforms: a Survey. 2008 Ieee International Conference on Industrial Technology, Vols 1-5, 2008: p. 1176-1181

[5]. Xiang-Yang Li, Y.W., and Yu Wang, Complexity of Data Collection, Aggregation, and Selection for Wireless Sensor Networks, IEEE Transactions on Computers, 2010. To appear.

[6]. Wang, L.L., et al., Scale-free topology for large-scale wireless sensor networks. 2007 Third Ieee/Ifip International Conference in Central Asia on Internet, 2007: p. 21-25

[7]. Slimane, J.B., et al., A Three-tiered Architecture for Large-scale Wireless Hospital Sensor Networks. Mobilizing Health Information to Support Healthcare-Related Knowledge Work, 2009: p. 20-31

[8]. Mangalwede, S.R. and D.H. Rao, Performance Analysis of Java-based Approaches to Distributed Computing. International Journal of Recent Trends in Engineering, 2009. 1(1): p. 556-559.

[9]. Zhang, B. and G.H. Li, Survey of Network Management Protocols in Wireless Sensor Network. 2009 International Conference on E-Business and Information System Security, Vols 1 and 2, 2009: p. 850-854

[10]. Zhao, S.L. and D. Raychaudhuri, Scalability and Performance Evaluation of Hierarchical Hybrid Wireless Networks. Ieee-Acm Transactions on Networking, 2009. 17(5): p. 1536-1549.

[11]. Paek, J., et al., The Tenet Architecture for Tiered Sensor Networks. Acm Transactions on Sensor Networks, 2010. 6(4): p. -.

[12]. Zhang, M.C., J.Y. Song, and Y. Zhang, Three-tiered sensor networks architecture for traffic information monitoring and processing. 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vols 1-4, 2005: p. 96-101

[13]. Wu, X., G.H. Chen, and S.K. Das, On the energy hole problem of nonuniform node distribution in wireless sensor networks. 2006 IEEE International Conference on Mobile Adhoc and Sensor Systems, Vols 1 and 2, 2006: p. 100-107
[14]. Ee, C.T., S. Ratnasamy, and S. Shenker, Practical data-centric storage. USENIX Association Proceedings of the 3rd Symposium on Networked Systems Design & Implementation (NSDI 06), 2006: p. 325-338
[15]. Ratnasamy, S., et al., Data-centric storage in sensornets with GHT, a geographic hash table. Mobile Networks & Applications, 2003. 8(4): p. 427-442.
[16]. Intanagonwiwat, C., et al., Directed diffusion for wireless sensor networking. Ieee-Acm Transactions on Networking, 2003. 11(1): p. 2-16.
[17]. Demers, A., et al., The cougar project: A work-in-progress report. Sigmod Record, 2003. 32(4): p. 53-59.
[18]. Greenstein, B., et al., DIFS : A distributed index for features in sensor networks. Proceedings of the First Ieee International Workshop on Sensor Network Protocols and Applications, 2003: p. 163-173
[19]. Li, X., et al., Multi-dimensional range queries in sensor networks, in Proceedings of the 1st international conference on Embedded networked sensor systems. 2003, ACM: Los Angeles, California, USA. p. 63-75.
[20]. Karp, B. and H.T. Kung, GPSR: greedy perimeter stateless routing for wireless networks, in Proceedings of the 6th annual international conference on Mobile computing and networking. 2000, ACM: Boston, Massachusetts, United States. p. 243-254.
[21]. Zhang, W.S., G.H. Cao, and T. La Porta, Data dissemination with ring-based index for wireless sensor networks. Ieee Transactions on Mobile Computing, 2007. 6(7): p. 832-847.
[22]. Wu, Y. and Y. Li, Distributed Indexing and Data Dissemination in Large Scale Wireless Sensor Networks, in Proceedings of the 2009 Proceedings of 18th International Conference on Computer Communications and Networks. 2009, IEEE Computer Society. p. 1-6.
[23]. Madden, S., Database Abstractions for Managing Sensor Network Data. Proceedings of the Ieee, 2010. 98(11): p. 1879-1886.
[24]. Chen, S.M., P.B. Gibbons, and S. Nath, Database-centric programming for wide-area sensor systems. Distributed Computing in Sensor Systems, Proceedings, 2005. 3560: p. 89-108.
[25]. Madden, S.R., et al., TinyDB: An acquisitional query processing system for sensor networks. Acm Transactions on Database Systems, 2005. 30(1): p. 122-173.
[26]. Levis, P., et al., The emergence of networking abstractions and techniques in TinyOS. Usenix Association Proceedings of the First Symposium on Networked Systems Design and Implementation (Nsdi'04), 2004: p. 1-14
[27]. Sun, J.Z. and J. Zhou, Power-Aware Data Reduction for Continuous Query in Wireless Sensor Networks. 2008 Ieee International Conference on Industrial Technology, Vols 1-5, 2008: p. 731-736
[28]. Demirbas, M., K.Y. Chow, and C.S. Wan, INSIGHT: Internet-Sensor Integration for Habitat Monitoring, in Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks. 2006, IEEE Computer Society. p. 553-558.
[29]. Lynch, J.P., An overview of wireless structural health monitoring for civil structures. Philosophical Transactions of the Royal Society a-Mathematical Physical and Engineering Sciences, 2007. 365(1851): p. 345-372.
[30]. Tian, Y., E. Ekici, and F. Ozguner, Cluster-based information processing in wireless sensor networks: an energy-aware approach. Wireless Communications & Mobile Computing, 2007. 7(7): p. 893-907.
[31]. Tian, Y., F. Ozguner, and E. Ekici, Comparison of hyper-DAG based task mapping and scheduling heuristics for wireless sensor networks. Computer and Information Sciences - Iscis 2005, Proceedings, 2005. 3733: p. 74-83.
[32]. Chintalapudi, K., et al., Structural damage detection and localization using NETSHM. IPSN 2006: The Fifth International Conference on Information Processing in Sensor Networks, 2006: p. 475-482
[33]. Nagayama, T. and B.F. Spencer, Model-based Data Aggregation for Structural Monitoring Employing Smart Sensors. 2008.
[34]. Zimmerman, A.T., et al., Automated Modal Parameter Estimation by Parallel Processing within Wireless Monitoring Systems. Journal of Infrastructure Systems, 2008. 14(1): p. 102-113.
[35]. Jardak, C., et al., Parallel processing of data from very large-scale wireless sensor networks, in Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. 2010, ACM: Chicago, Illinois. p. 787-794.
[36]. Kuorilehto, M., et.al., A survey of application distribution in wireless sensor networks. EURASIP J. Wirel. Commun. Netw. 2005, 5 (October 2005): p.774-788.
[37]. Fasolo, E., and Rossi, M., et.al., In-network Aggregation Techniques for Wireless Sensor Networks: A Survey, IEEE Wireless Communications, Vol. 14, No. 2, April 2007, p. 70-87.