WIP: Energy Container for Database-Oriented Sensor Networks

[Extended Abstract]

Simon Kellner
System Architecture Group
Universität Karlsruhe
simon.kellner@kit.edu

ABSTRACT

Energy remains the most critical resource in sensor networks. In static sensor-net applications where most events can be calculated a priori, energy management is often done implicitly and manually by application developers.

With the advent of database interfaces to sensor nets, this is no longer possible due to dynamically created queries. There are several scenarios in which it is desirable to get an accurate account of the resources a sensor net consumed on behalf of a certain query.

In this work, we propose to adapt the Resource Container concept from the desktop computing world to sensor networks in order to facilitate dynamic energy accounting.

1. INTRODUCTION

Energy still is the most critical resource in sensor networks. Current energy supplies already take up most of a sensor node's space, but can provide the desired node lifetimes of years only when sensor-net application designers give a high priority to a long sensor-net lifetime. Sensor-net Operating Systems (OSes) like TinyOS [2] encourage energy saving by not providing a convenient CPU-abstraction such as threads, which could, for example, tempt application developers into creating CPU-intensive waiting loops and thus into wasting energy.

Database interfaces to sensor nets like TinyDB [3] make it easier for users to retrieve sensor data: A sensor-net application is formulated as a request in an SQL-like language and interpreted by the sensor net until the request expires. The program on the sensor nodes only needs the ability to interpret and execute such requests. This eliminates the need to reprogram sensor nodes and allows multiple queries to be processed simultaneously.

Such dynamic systems can support multiple users in a sensor net, each with his own set of queries. In this scenario it is desirable to account the energy consumption of each query, e.g. to bill users based on their sensor net "usage", or to find the query with the highest energy consumption and cancel it before it wears down the energy supplies.

On traditional computers on the desktop or in the server room, Resource Containers (RCs) are used for this purpose. In this work, we investigate how this concept can be adapted to TinyOS, an OS widely used on sensor nodes. We consider RCs for energy accounting only, although the concept can be used for all resources an OS manages.

2. BACKGROUND

In this section, traditional Resource Containers as they are used in PCs are introduced as well as the properties of TinyOS that are relevant to this work.

2.1 Resource Containers

Resource Containers are an OS-abstraction introduced by Banga, Druschel and Mogul [1] in 1999 and consist basically of OS-provided storage for accounting data. The idea is to separate OS abstractions for CPU and resource accounting, because resource usage is independent of CPU abstractions.

Instead, it is dependent on *tasks*: A task is loosely defined as something a user wants the system to do (e.g., serving a web page or drawing a picture). In modern systems, such a task is no longer identical to a process: A web-server can use threads to serve different web pages simultaneously (one process working on several tasks), or several processes cooperate to accomplish one task (e.g., graphical application and X-server).

In an RC-enabled system, every process can create RCs, change its active RC and share an RC with another process. Continuing both examples above, a web-server can bind each of its threads to a separate RC, and a graphical application can share an RC with the X-server.

One such implementation of RCs for Linux is described in [4]. Here, when a process creates a new RC, the RC is bound to a file descriptor. This has the advantage that existing code used for user-land handles to kernel objects can be reused. The RCs are organized in a hierarchy in which each parent RC holds the accounting data aggregated over all its children. So a process cannot cheat the system by creating new RCs, since they will all have the same parent.

In summary, RCs give administrators and users the ability of accounting tasks, which usually has a higher significance than process-based accounting.

2.2 TinyOS

TinyOS, one of the most prevalent OSes for sensor nodes, is event-driven. It does not provide "convenient" CPU ab-

stractions known from other OSes like processes or threads. All activities in TinyOS can be seen as responses to interrupts. These responses typically consist of few instructions and some commands to peripheral hardware that will trigger the next interrupt, allowing the processor to sleep in the meantime. This design does not tempt inexperienced programmers into creating energy-expensive polling routines.

Instead of high-level processor abstractions like threads or processes, TinyOS opts for a low-level abstraction in order to save stack memory, the TinyOS `task`s (not to be mistaken for the RC-related tasks). TinyOS `task`s run until completion and cannot be interrupted by other TinyOS `task`s. This is a strong incentive to keep them short, as other operations would suffer serious delays. Instead, a long-running TinyOS `task` should enqueue (`post`) itself in the run queue and quit, postponing its work in effect. In summary, a typical workflow of packet reception, sensor queries, a bit of computation and packet transmission on a sensor node is distributed across several of its devices and held together by TinyOS `task`s and interrupts, interspersed with sleep intervals.

The original RC concept associates processes or threads with one or more RCs on which the OS can account resource usage. In the absence of these processor abstractions the association of resource use with RCs is more difficult.

3. RESOURCE CONTAINERS IN TinyOS

The focus of this work is on how to attach RCs to queries executed by a TinyOS application, since dynamically created queries are the most interesting targets for on-line accounting. However, RCs can be used to account other tasks as well.

In the following we assume that the application can identify queries through an ID which is present in all packets related to that query.

3.1 Normal Resource Containers

A normal RC is associated with a query. As soon as an application learns the ID of the query currently being processed, it informs TinyOS that it wishes to switch to the RC associated with this query. The selected RC is then bound to the current TinyOS `task`.

The energy consumption of all further activities coming from this TinyOS `task` is accounted to the selected RC. If the TinyOS `task` posts a new TinyOS `task` or sets up a new timer, this binding can be stored by the scheduler or timer system, and can be used to switch back to the stored RC automatically on the corresponding wake-up call.

The OS here clearly depends on the application for correct accounting, but this is both feasible and necessary in a sensor-net application. It is necessary to prevent producing hard-to-maintain code, and it is feasible because there should be only few places where this RC-switching occurs, namely when a sensor-net application starts processing a query.

3.2 Anonymous Resource Containers

Since TinyOS applications spend most of their time sleeping and perform only minimal amounts of processing, energy consumed during interrupt handling is not negligible.

For example, a timer interrupt may cause the activation of a communication device, which is subsequently used to send stored sensor data to other nodes. The sensor node is not aware of the query ID until it accesses the packet it is about to send. In the meantime, the activation of the communication device can consume a substantial amount of energy that cannot be assigned to the correct RC at that moment.

As a solution, the interrupt handler can allocate a temporary, anonymous RC and use it to account both its own energy consumption and the device activation. Later, when the application becomes aware of the query ID, it can switch to the RC associated with the query, causing the temporary RC to be merged and released.

3.3 Special Resource Containers

It may be necessary to employ special RCs to provide additional information or to handle cases where the correct RC is not known.

3.3.1 Root Resource Container

One RC worth mentioning is the RC for the whole node. It is used to collect the amount of energy consumed by the whole node, regardless of queries. This information is of interest to the nodes themselves in order to estimate the amount of remaining energy. It can also be regarded as another data source and can itself be the target of a query.

3.3.2 Idle Resource Container

Some energy consumption simply can not be clearly accounted to a query, e.g., the energy spent during sleep (*idle energy*). We call the problem of accounting this energy consumption in a fair manner *accounting fairness*.

One way to address the issue of idle energy accounting is to distribute the accounted idle energy among all queries known to the sensor node. To achieve this, a special RC for this energy class is present in the system. At certain times, this RC is cleared and its content distributed among all existing normal RCs. This has to be done both periodically and on creation/expiration of a query:

- Periodically so that the accounting information remains recent,
- At query instantiation to avoid penalizing this query by accounting sleeping energy spent before its instantiation, and
- At query expiration to avoid losing accounted energy.

The fairness of this distribution is subject to discussion and thus should be handled by a project-dependent policy. Policy examples include equal distribution and partitioning according to duty-cycle or used energy.

So, one can picture the RCs in a 3-level hierarchy: the root RC for the node, named RCs for the queries and anonymous

RCs to account energy consumed for a (yet) unknown purpose. In this hierarchy the root RC contains the aggregated accounting data of the named RCs, while the anonymous RCs will eventually be merged with one of the named RCs.

3.4 Shared Data

Caching the acquired sensor data introduces another instance of the accounting-fairness problem. Without additional measures, the first query to sample data bears the cost of acquiring it, subsequent queries can use it at almost zero cost. If the accuracy of timing or accounting can be relaxed, some trade-offs between one of them and accounting fairness can be considered.

A trade-off between timing accuracy and accounting fairness can be implemented as a subscriber model for sensor data: The sensor data is sampled either on time-out after the first subscription or when enough parties subscribed to this sensor data. The energy is split among all of the subscribed parties.

Another trade-off between accounting accuracy and fairness can be implemented by assigning a value to the sampled sensor data that decays with every access. For example, the initial query bears 3/4 of the costs, the next query 3/4 of the remaining costs, and after a time-out, the rest is distributed across all queries that acquired this data.

3.5 Resource Container Aggregation

RCs lend themselves quite naturally to sensor nets with dynamically created queries. When receiving a new query, a sensor node allocates an RC for this query, accounts the query's energy costs to that RC and sends the accounted data back together with the responses to this query.

RC contents can easily be aggregated by summation over all RCs with the same query ID. The design of RCs to store all of the energy accounted to it since its creation makes it resilient to occasional packet loss. When accounting information is lost in the network due to temporary packet loss, the aggregated accounting information at the data sink may be incorrect, but it will be correct again once the temporary packet loss is over.

To allow the data sink to compare the collected aggregated RC values and to detect packet loss, a node should additionally send the number of sensor nodes involved in an aggregate, if this information is not already present in the aggregated sensor data.

4. CONCLUSION

Resource Containers are an elegant concept for resource accounting in traditional operating systems. This concept can be adapted to the field of sensor networks, where pure event-driven operating systems prevail.

The benefit of this concept is accurate accounting of sensor network tasks, which is useful information to developers, administrators and users.

5. ACKNOWLEDGMENTS

This work is done as part of the BW-FIT project ZeuS.

6. REFERENCES

[1] G. Banga, P. Druschel, and J. Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the Third Symposium on Operating System Design and Implementation (OSDI'99)*, Feb. 1999.

[2] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 93–104, New York, NY, USA, 2000. ACM Press.

[3] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 491–502, New York, NY, USA, 2003. ACM Press.

[4] M. Waitz. Accounting and control of power consumption in energy-aware operating systems. Master's thesis, Department of Computer Science 4, University of Erlangen, Jan. 2003. SA-I4-2002-14.