Work in Progress – WiSeR Distributed File System for Heterogeneous Sensor Platforms

Jatindera S. Walia, Rong Zheng
Department of Computer Science, University of Houston,
Houston, TX 77204 jwalia@uh.edu, rzheng@cs.uh.edu

Abstract

Pervasive computing is on the horizon with ever shrinking device form factors and increasing computation power. Sensor devices such as those used to monitor temperature, pressure, humidity, flow, audio, video etc. allow acquisition of information regarding the physical environments, which can be utilized in actuation and visualization. Traditionally, sensing data is either stored locally or transported to a centralized location at the perimeter of the network. We argue that the information producers and consumers in a cyber physical system are likely to be distributed and with different capabilities. This necessitates a uniform abstraction that can hide device/platform dependent characteristics, intricacies of communication and networks, and utilize distributed heterogeneous storage among different devices. A distributed storage solution also has the benefit of fault tolerance in events of disasters, such as infrastructure failure or natural calamities, wherein remnants of information stored over the distributed storage system from the destroyed devices can be invaluable. In this paper, we propose a Distributed File System that is agnostic to platforms and operating systems of deeply embedded devices.

1. Introduction

Pervasive computing is quickly moving from scientific and industrial domain to application domain. These deeply embedded systems will integrate and work in synergy to provide a smart environment. The applications of such systems are limitless from smart surgical rooms to rapidly deployed first-response systems in event of a disaster.

Heterogeneous devices, such as those shown in Figure 1, with different processing powers, energy requirements, and storage capacity will participate in such systems. This heterogeneity significantly adds to the complexity in building applications for such systems as application developers have not only to worry about the specifications of the applications but also need to deal with complexity of heterogeneous platforms and network intricacies.

A Distributed File System in such a scenario will help tackle some of the problems faced. Our goals in developing such a file system are to share local storage, sustain device failure, and provide redundancy and some level of scalability in addition to achieving platform independence and small memory footprint.

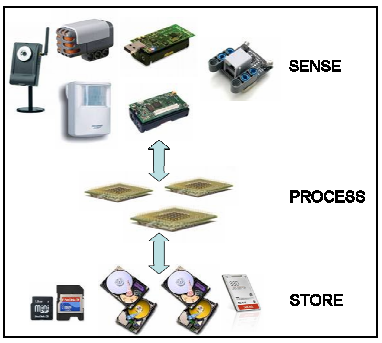

Figure 1: Deeply Embedded Systems

2. Background

Files are binary patterns that make sense to the corresponding applications. File Systems break these bit patterns into smaller data blocks of some optimum size; create metadata blocks to store information about the attributes of the files and the location of the data blocks. In addition, they try to minimize the energy consumption as well as the access time to the files. Since writes are always more expensive than reads, log based file systems, journal file systems and hash based file systems are some of the solutions that have been proposed.

Local file systems are susceptible to failure and in addition require the device to own some sort of storage

hence increasing the initial and maintenance cost of the device. Distributed File Systems transparently provides fail safety, redundancy and robustness to the file system and allows storage constrained clients to be deployed.

In sensor motes running TinyOS, Matchbox, ELF and Capsule are some of the current local File Systems. Since, these devices use flash based permanent storage; efforts have been made to optimize the file system to fully utilize the physical characteristics of the flash based memory. Distributed File Systems in the case of motes are severely penalized because of high cost of communication that decreases the life time of the deployment of such motes.

In other sensor devices such as surveillance cameras Linux is the de facto operating system. Network and Distributed file systems in UNIX and Linux are often optimized for disk drives parameters to reduce seek and rotational latency.

3. Design

In our initial design of Wiser Distributed File System (wDFS) shown in Figure 2, we have the following major components:

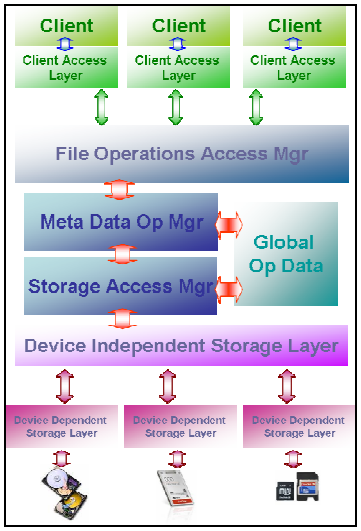

Figure 2 : wDFS Architecture

- **CAL:** Client Access Layer runs on the client and provides access to the wDFS. It provides cache to reduce traffic and communicates with the FOAM using SOAP.
- **FOAM:** File Operation Access Manager runs on the Metadata nodes and interacts with the CAL and handles clients' requests. It forwards those requests to MOM.
- **MOM:** Metadata Operations Manager handles metadata operations such as resolving file names to associated metadata blocks that contains attribute and data block information. It also interacts with the SAM and GOD to provide File semantics to the clients.
- **GOD:** Global Operations Data maintains information regarding the physical characteristics, space availability, reliability and availability of the storage nodes.
- **SAM:** Storage Access Manager handles the storage of the blocks over various distributed storage nodes. It interacts with DISL, GOD and MOM.
- **DISL:** Device Independent Storage Layer provides a consistent interface to the SAM.
- **DDSL:** Device Dependent Storage Layer is specific to the storage device and tries to optimize the block read and write as per the physical characteristics of the device.

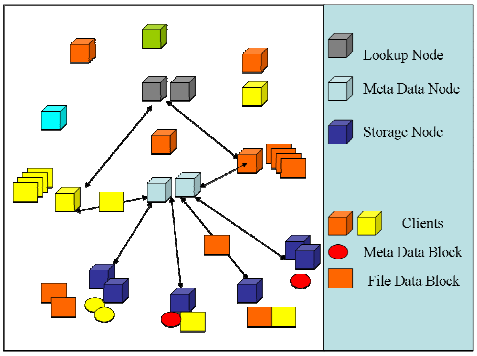

Figure 3 : Logical Interaction

Figure 3 depicts the logical interaction between the clients and various providers of the wDFS. Clients contact Lookup node to get information about Metadata nodes and then interact with them to store their files. Metadata nodes in turn interact with storage nodes to store the Meta as well as the data blocks of the files.

4. Summary

We propose a Distributed File System to allow easier deployment of storage constrained heterogeneous sensing devices. Initial attempt will be to build a stateless, flat DFS for homogeneous platform and then extend it to heterogeneous platforms followed by hierarchy, redundancy and security add-ons. We plan to implement such a system over TinyOS motes and then extend them over to wrap boards and Linux laptops in our implementation phase that is underway.