Energy Characterization of Filesystems for Diskless Embedded Systems

Siddharth Choudhuri
University of California, Irvine
Irvine, CA
sid@ics.uci.edu

Rabi N. Mahapatra
Texas A & M University
College Station, TX
rabi@cs.tamu.edu

ABSTRACT

The need for low power, small form-factor, secondary storage devices in embedded systems has led to the widespread use of flash memory. Energy consumption due to processor and flash for such devices is critical to embedded system design. In this paper, we have proposed a quantitative account of energy consumption in both processor and flash due to overhead of filesystem related system calls. A macromodel for such energy consumption is derived using linear regression analysis. The results describing filesystem energy consumption have been obtained from Linux Kernel running Journaling Flash Filesystem 2 (JFFS2) and Extended 3 (Ext3) filesystems on StrongARM processor with flash as secondary storage device. Armed with such a macromodel, a designer can choose to partition filesystem, estimate the application energy consumption (processor and flash) due to filesystem during the early stage of system design.

Categories and Subject Descriptors

D.4 [**Operating Systems**]: System Program and Utilities; I.6.5 [**Simulation and Modeling**]: Model Development

General Terms

Measurements

Keywords

Diskless, Flash, Macromodel

1. INTRODUCTION

Power aware design of embedded systems has emerged as one of the key design issues. The design of embedded operating system can greatly affect the performance of the overall system both in terms of performance and energy consumption. Embedded operating systems, unlike conventional desktop operating systems have limited memory footprint and functionalities depending on the underlying architecture and application requirements. Though hard disk drives are the *de-facto* standard for secondary storage in conventional desktop systems, they do not fit in the realm of embedded systems. This has lead to the widespread use of a ROM, EEPROM or *flash* based devices as means of secondary storage. With the growth of embedded systems coupled with the need for having increasingly larger amounts of storage space, we are likely to see a significant growth in the storage capacity of flash based systems. One of the components in an operating system is the filesystem layer that interacts with flash memory through flash-driver. The flash-driver is responsible to initiate activities like flash read or flash write and deals with the secondary storage. Thus, an energy efficient design of filesystem for flash memory becomes imperative to reduce the energy consumption in such devices. A study on filesystem energy consumption would require an experimental setup that adds to the cost and time to market. The aim of this work is to devise a quantitative study of filesystems for flash based device and provide energy consumptions using quantitative macromodel.

DAC 2004, June 7–11, 2004, San Diego, California, USA.

2. RELATED WORK AND OUR APPROACH

A recent study of energy management of virtual memory for diskless devices has been carried out in [1]. This work concerns energy management for the main memory and does not consider secondary storage. An energy consumption simulation framework has been proposed in [5]. Flash-memory based filesystems were implemented in [2, 8]. The implementation in [8] is a flash based journaling filesystem called JFFS2 and has been in widespread use. However, energy characterization of such filesystem has not yet been studied. Macromodeling of operating system energy consumption has been proposed in [3, 4] where, a linear equation based model has been suggested to relate energy consumption in processor to low level system calls. Energy macromodel for various operating system related activities like context switch, signal handling have been developed to find a relation between various subsystems of an operating system to processor energy consumption. Our work is motivated by these studies and extends to investigate file systems which is an important component of today's flash based embedded systems. This work focuses on macromodeling filesystem energy consumption in terms of both energy consumed by the processor and the secondary storage, hence differs from [3].

We introduce a macromodel that relates the processor and flash energy consumption as a function of system calls. To

our knowledge, this is a first step towards filesystem related energy characterization of flash based devices. Further, we have derived macromodels for two of the popular filesystems Ext3 and JFFS2 [9, 8]. An embedded systems developer can use the macromodel early in the design phase to *1.* Choose between different filesystems. *2.* Estimate the energy consumption of applications due to the filesystem activities. *3.* Tune the file transfer activities to reduce the power consumption in a system. The macromodel can be used without actually having energy measurement setup. Also, splitting the energy consumption due to filesystem activity into processor and flash shall provide opportunity for an in-depth analysis and trade offs for the designer.

3. EXPERIMENTAL SETUP

Our experimental setup consists of a LART board [6] based system with Intel StrongARM SA1100 processor and Linux 2.4.18 as the OS. A PCI based data acquisition system integrated with Lab-view serves as the power measurement system for the processor. In order to profile the energy consumption, GPIO signals are sent using a driver that is implemented as a */proc/trigger* interface in the kernel. This signal serves to start and stop the energy measurements during filesystem activities. The processor energy consumption measured between the start and stop intervals (separated by the two triggers), is given by the following equation

$$E_{cpu} = \int_{TRIG_{start}}^{TRIG_{stop}} \frac{V_{sense}(t)}{R_{sense}} * V_{dd}\ dt \qquad (1)$$

The energy consumption of flash is calculated using traces taken from flash memory accesses. We added code to the existing flash driver that calculates the energy consumption depending on the mode and access time of the flash memory. A new kernel data structure consisting of linked list is introduced, that has per process flash read and flash write energy consumption information. The energy consumption per process is calculated using the following equation

$$E_{flash} = V_{dd} * I_{mode} * t_{access} \qquad (2)$$

The value of I_{mode} is obtained from [7] depending on what kind of operation is being done (read/write/program/erase). The flash access duration, t_{access} is calculated using the timer function provided by the kernel. Whenever a process accesses flash, the energy consumption is automatically calculated by the kernel using Equation (2). This data is logged in a */proc* interface based on a per process energy consumption. There are two separate proc entries */proc/wtrace* and */proc/rtrace* that log the write and read energy consumption respectively. The overall procedure for profiling the flash energy consumption is given in Figure 1. The Virtual Filesystem (VFS) layer converts the "generic" filesystem related *system calls* into filesystem "specific" calls and delegates it to the appropriate filesystem layer below it. The profiler layer on top of the flash driver profiles the write and read requests separately and outputs information on two different files in */proc* (for read and write). The data is used for generating macromodel for flash energy consumption. The 4 MB of NOR flash on the LART board has three partitions for boot-loader (128 KB), Linux kernel (896 KB) and a 3MB partition for the filesystem under study to be profiled. The root partition is mounted by the kernel as a *ramfs* filesystem in the DRAM. The reason is that the root partition writes a

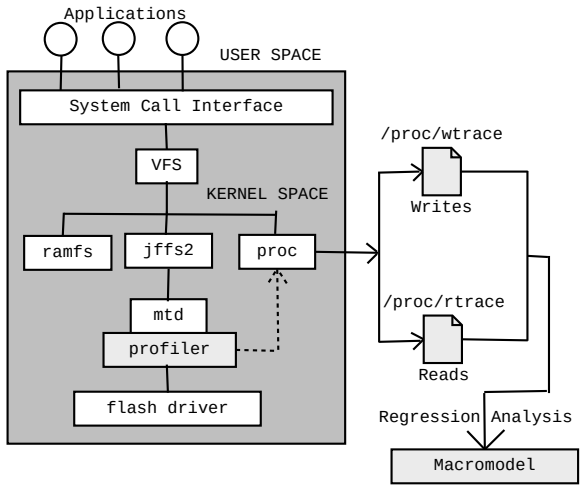

Figure 1: Profiling flash accesses

number of files in */tmp* and */var* directories. If the root partition was to be mounted on flash, these writes and reads would interfere with the actual read and write operations being measured.

4. MACROMODELING

A mathematical macromodel for the energy consumption analysis is developed as follows. Let $E_{cpu}(x)$ = Energy consumption of CPU due to x bytes of data in a filesystem operation. Similarly, we have $E_{fw}(x)$ and $E_{fr}(x)$ as the write and read energy respectively. Since the energy consumption of the processor and the flash are directly proportional to the number of bytes that are written to or read from the file, we have the following linear relation between Energy and bytes:

$$\mathbf{E}(x) = \mathbf{A}x + \mathbf{B} \qquad (3)$$

Specifically, we will have three such equations corresponding to CPU, write and read operations. In order to solve the above equations, we need to establish the values of constants $\{(A_i,\ B_i)\}$ using variable x. If we have a set of n values $\{(e_0,\ x_0),\ (e_1,\ x_1),\ (e_2,\ x_2),\ ...,\ (e_n,\ x_n)\}$ that relate the energy consumption e_i to bytes x_i, using standard results from regression analysis [10], the relation can be expressed in the form of a general matrix relation

$$\begin{pmatrix} x_0 & 1 \\ x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix} * \begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} e_0 \\ e_1 \\ \vdots \\ e_n \end{pmatrix} \qquad (4)$$

5. PROFILING FILESYSTEM ACTIVITIES

This section gives a description of the process adopted to profile filesystem activities. The following steps illustrate the methodology

STEP 1 The macros are written such that they highlight the required filesystem activity.

STEP 2 In this step the macro is run and the energy consumption due to processor is obtained from the energy measurement setup. The energy consumption due to flash is obtained by reading the profiled data from the */proc/wtrace* and */proc/rtrace* interface.

STEP 3 Steps 1-2 are repeated with 'n' different data sizes. The size of data is varied from 4 bytes to 64KB in increasing powers of two. This is done in order to ensure that a new block is created in the *inode* to store the large files. This size represents the variable x in Equation (3). After repeating this for 'n' different data sizes, we have a set of n values $\{(e_0, x_0), (e_1, x_1), (e_2, x_2), ..., (e_n, x_n)\}$ that relate energy consumption to bytes.

STEP 4 The values of processor and flash energy consumption is tabulated against varying values of x (from the previous step). Solving Equation (3), we get the values of A_{cpu}, B_{cpu}, A_{fw}, B_{fr}, A_{fr}, A_{fw} for each filesystem operation of Step 1.

STEP 5 The error due to such a model is given by

$$error = \sqrt{\sum_{1}^{n} \frac{1}{n}(\frac{E_m - E_a}{E_a})^2} \quad (5)$$

E_m = The measured energy using macromodel equations
E_a = The actual energy from measurement or simulation

6. EXPERIMENTAL RESULTS

We considered two filesystems in our case studies, Journaling Flash Filesystem (JFFS2) and Extensible Filesystem-3 (Ext3). JFFS2 is a log structured filesystem developed for flash based devices [8]. Extensible filesystem-3 (Ext3) is a journaling filesystem based on conventional EXT2 filesystem [9]. The reason for using Ext3 and JFFS2 was to study two filesystems that have the same design goals of having journal information available in the filesystem along with data and meta-data in order to improve availability and robustness.

System Call	JFFS2		Ext3	
	CPU Energy	Err	CPU Energy	Err
creat	21750	0.09	18000	0.17
link	23750	0.09	17775	0.15
chown	20000	0.15	15750	0.02
mkdir	19750	0.02	19750	0.06
rmdir	23250	0.02	19500	0.05
mkfifo	21750	0.15	19000	0.13
write	12.67x + 48885	0.02	80x + 14320	0.06
rename	33500	0.01	18500	0.06
unlink	93x - 27755	0.40	1.8x + 14320	0.29
read	29x + 18976	0.06	46x + 12365	0.03

Table 1: Processor Energy (nJ): System Calls

System Call	Write Energy	Err	Read Energy	Err
creat	21211	0.12	2122	0.0
link	8984	0.24	169	0.31
chown	13258	0.12	1083	0.0
mkdir	21332	0.06	2068	0.0
rmdir	12187	0.07	166	0.05
mkfifo	21070	0.07	228	0.03
write	178x + 30496	0.06	0.03x + 169	0.03
rename	18870	0.21	227	0.26
unlink	0.25x + 11918	0.89	90	0.10
read	0.04x + 262	0.19	35x + 356	0.40

Table 2: JFFS2 Flash Energy(nJ) : System Call

Tables 1, 2 and 3 show the energy consumption macromodel in terms of *system calls*. This is an important metric as every user level filesystem related operation eventually maps to kernel level system calls. The equations from the system call level can be used to develop a tool to profile energy consumption due to higher level filesystem operations. As can be seen from the tables, the energy consumption of flash due to JFFS2 filesystem is better than that of Ext3. JFFS2 works directly with flash chip driver to issue read/write requests of the required number of bytes. Ext3 on the other hand is a filesystem designed for block based device. It sees the flash as a block device and uses a translation layer called *mtdblock*, to issue the requests to the flash chip. Thus the read/write requests are made in multiples of block size. This implies that Ext3 has a poor performance on flash for small read/write requests. The energy consumption due to processor however is higher in case of JFFS2. This can be attributed to the fact that JFFS2 tries to compress data being written into flash during a write operation. While doing a read operation, it decompresses the data on fly. The macros used to generate the macromodel considered random data to create files, so that the compression is not optimal and the equations give a worst case bound on the energy consumption. Tables 4, 5 and 6 are used to validate the macromodel by profiling some of the frequently used benchmark programs. The profiled results were compared with the results from actual measurements to estimate the errors. The percentage error shows the accuracy of the macromodel based energy characterization of filesystem.

System Call	Write Energy	Err	Read Energy	Err
creat	262144	0.31	13745	0.24
link	169887	0.25	20608	0.21
chown	95572	0.12	13746	0.22
mkdir	195883	0.08	20548	0.00
rmdir	247599	1.06	27481	0.59
mkfifo	165325	0.10	20605	0.07
write	75x + 242850	0.56	20645	0.01
rename	247832	1.02	27487	0.50
unlink	5.2x + 24210	0.66	9550	0.16
read	8234	0.24	4.6x + 14016	0.45

Table 3: Ext3 Flash Energy(nJ) : System Call

Program	JFFS2 Processor Energy			Ext3 Processor Energy		
	Actual	Eval	Err	Actual	Eval	Err
compress	68254	69513	-1.8	62134	58341	7.0
ucbqsort	210742	207561	1.5	124200	112320	9.5
v42	96600	96420	0	824000	798657	3.0
jpeg	159890	160896	0	5810000	6176320	-6.3
adpcm	148000	148785	0	91500	92721	-1.3

Table 4: Benchmark Processor Energy (nJ)

7. ANALYSIS OF MACROMODEL

The analysis of energy consumption due to processor is shown in Figure 2. In order to analyse, we used a high level filesystem operation of creating a new file (This would use *creat* and *write* system calls) . Since JFFS2 compresses data that is to be written to the secondary storage, two cases are considered. The worst case performance is where data stored is inherently random so that the compression ratio is minimal. The second case is the average case. The following results for processor energy consumption can be summarized from Figure 2. 1. JFFS2 is not suited for small file size (< 100 bytes) due to the fact that the overhead due

Program	Flash Write Energy(nJ)			Flash Read Energy (nJ)		
	Actual	Eval	Err	Actual	Eval	Err
compress	62011	59742	3.6	340	358	-5.2
ucbqsort	139693	147376	-5.2	335	358	-6.8
v42	372899	385900	-3.4	153	162	-7.0
jpeg	1282484	1314795	-2.5	2420	2301	4.9
adpcm	85102	94190	-10.6	326	344	-5.5

Table 5: Benchmark Flash Energy (nJ) : JFFS2

Program	Flash Write Energy(nJ)			Flash Read Energy (nJ)		
	Actual	Eval	Err	Actual	Eval	Err
compress	267546	256370	4.1	39954	40496	-1.3
ucbqsort	209450	229706	9.6	1900	2321	2.2
v42	564550	570326	1.0	64250	66077	-2.8
jpeg	5808450	6002850	-3.3	36500	34390	5.7
adpcm	285550	270950	5.1	42500	41684	1.9

Table 6: Benchmark Flash Energy (nJ) : Ext3

to compression is of the order of file size itself. For small sizes Ext3 is better off by storing the data "as-is" without compression. The advantages due to compression are significant and visible as the file sizes increase. For large files, the overhead due to compression is insignificant compared to the actual data.

2. JFFS2 with worst case compression and Ext3 consume almost the same amount of processor energy for large file sizes. JFFS2 consumes slightly higher in this case because processor cycles are wasted in trying to compress data. However, this worst case upper bound is only for data that is so random that it cannot be compressed and hence not likely to occur frequently in normal filesystem activities.

The analysis for flash energy consumption for the same activity to create a new file is shown in Figure 3. The following observations can be made from this figure.

1. Ext3 is expensive for small files ($<$ 128K). This is due to the fact that the requests sent to the flash chip are in multiples of 128K. For small files, this would mean that 128K bytes of data is written, no matter what is the size of request (if it happens to be less than 128K). For large files, Ext3 is almost as good as JFFS2 with best case compression.

2. JFFS2 with worst case compression is always more expensive than Ext3 for file sizes $> 1K$ and JFFS2 with average case. This is due to the extra overhead of keeping the compression information along with the file itself when the compression ratio is large. It is to be noted that some of the filesystem related system calls have a constant amount of energy consumption due to the fact that they change only metadata and not actual filesystem data.

8. CONCLUSIONS

The mathematical model describing energy consumption as a function of filesystem level system calls can be a powerful tool for system designer while choosing filesystem of choice from an energy consumption point of view. Such a tool is also useful to estimate the energy consumption due to filesystem for user applications. This could well fit into embedded system design methodology, wherein a designer is interested to estimate the energy consumption before the actual product is developed. It is possible to partition the flash chips based on *apriori* knowledge of per filesystem energy consumption.

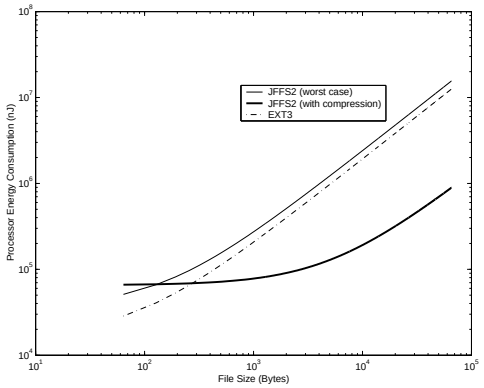

Figure 2: Comparison of Processor Energy

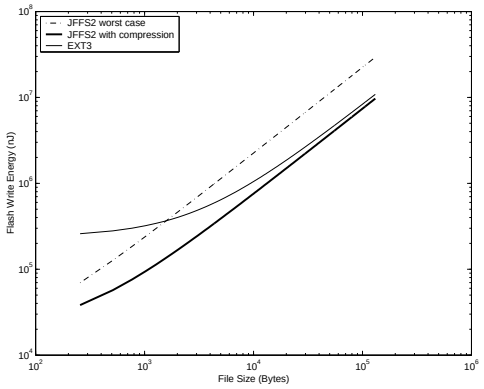

Figure 3: Comparison of Flash Write Energy

9. ACKNOWLEDGMENTS

This work was done at Embedded Systems Co-design Laboratory, Department of Computer Science, Texas A&M University and supported by funds from Ford Motor Foundation & TEES

10. REFERENCES

[1] J. Hom and U. Kremer. Energy management of virtual memory on diskless devices, 2001. in *Proceedings of the Workshop on Compilers and Operating Systems for Low Power*, September 2001.

[2] Atsuo Kawaguchi, Shingo Nishioka, and Hiroshi Motoda. A flash-memory based file system. In *USENIX Winter*, pages 155–164, 1995.

[3] N. Jha T. K. Tan, A. Raghunathan. Embedded operating system energy analysis and macro-modeling. In *ICCD*, 2002.

[4] T. K. Tan, Anand Raghunathan, Ganesh Lakshminarayana, and Niraj K. Jha, "High-level software energy macro-modeling," in *Design Automation Conference*, 2001, pp. 605–610.

[5] N. Jha T. K. Tan, "EMSIM an energy simulation framework for an embedded operating system," in *Proceedings of International Symposium on Circuit and Systems*, May 2002.

[6] "The lart pages," in *http://www.lart.tudelft.nl/*, May, 2003.

[7] Website. Intel Fast Boot Block Flash Memory, 2000. *http://www.intel.com/design/flash*

[8] *JFFS: The Journalling Flash File System.* Ottawa Linux Symposium, 2001. *http://sources.redhat.com/jffs2/jffs2.pdf.*

[9] Stephen Tweedie. Journaling the Linux ext2fs filesystem. In *LinuxExpo '98*, 1998.

[10] Seber, G. A. F., *Linear Regression Analysis*, John Wiley, Hoboken, NJ, 2nd edition, 2003.