Requirements and implementation of printing subsystem in educational computer system

Hideo Masuda, Masahide Nakamura, Akinori Saitoh, Koichi Kondo, Michio Nakanishi
Informedia Education Division, Cybermedia Center, Osaka University
1-30, Machikaneyama, Toyonaka, Osaka, 560-0043, Japan
Tel: +81 6 6850 6076 Fax: +81 6 6850 6084
Email: h-masuda@cmc.osaka-u.ac.jp

Abstract

This paper presents an effective framework to avoid waste printing in an educational computer system. We first point out four primary requirements to achieve the goal: (1) easy previewing, (2) easy shrinking (e.g., page selection, n-up), (3) quota of papers (user-wise and printer-wise accounting), and (4) authorized printer access.

Then, to meet the above requirements, we have implemented a printing subsystem that consists of LPRng-based servers with sophisticated settings, quota management tools, and GUI applications for Linux clients. Statistics show that the number of papers printed is significantly reduced after deploying the proposed system.

1 Introduction

2 Requirements for printing system in educational computer system

2.1 Overview of educational computer system

The educational computer system of the Cybermedia Center, Osaka University, provides computing services to all Osaka University students. Its computer system consists of nearly 800 Linux PCs and 10 unix server workstations. The PC labs are distributed throughout 14 schools and faculties, including university libraries.

The educational computer system has only two gateway servers, namely the mailserver and the proxy server, to the Internet. The internal network, to which Linux PCs and printers are connected, uses private network addresses.

The system is used by more than 14,000 students. Each student is given a login account and disk space up to 20 megabytes, and can access the same computer environment from any Linux PC. Lectures concerning the use of the center's facilities cover computer literacy education as well as advanced level education, e.g. simulation using the super computer of the center.

2.2 Requirements for printer subsystem

We, administrators of the educational computer system, do not simply want to reduce the running cost of printers, but do hope that users should make good use of the limited resources (paper and toner). To achieve this policy, it was necessary to count the correct number of pages for each job, and to set the upper limit of pages for each user. At the same time, we should teach students to avoid waste printings, and provide them with easy-to-use interface for the saving operations.

The major requirements for printer subsystem can be divided into two categories. First category items are for administrators and should be implemented in the management system.

A1: collect the number of pages of each print job

A2: set the quota of pages for each user

A3: check the user's quota and add up the page count in the database

A4: limit access from unauthorized PC's

Requirements of the second category is of the GUI tool. The tool should be implemented so that unskilled students can do the following operations with ease.

U1: preview all outputs in a uniform manner

U2: shrink (e.g., page selection, n-up, etc.)

U3: check the quota of papers and its balance

U4: manage printing his/her jobs in the queue.

3 Implementing printer servers

In this section, we show how to design and implement the printing servers. This covers the administrator requirements from A1 to A4.

3.1 Subsystem design

The printers in our system are ordinary PostScript network printers. The printers themselves do not have such special features to implement mechanisms satisfying the administrator requirements, specifically:

F1: Authentication and accounting functions.

F2: Access restriction of services (e.g., ftp, http, lpd) and clients (hosts, networks).

In order to achieve these features, we decided to deploy printer servers. The printer servers first intercept all printing data from Linux PCs, and then perform transactions on authentication and accounting. Finally, after the transacrion, the servers pass the data to the printers. So, the feature F1 could be achieved by implementing sophisticated mechanisms for the transactions on the servers.

However, for the access restriction (F2), we need to prohibit direct access from the Linux PCs to the printers. So, we divided the network into two disjoint subnets for the printers and the Linux PCs, and deployed the printer servers between two subnets. For this, note that the printer servers must not act as IP routers.

3.2 Printer servers with LPRng

We assume in the system side that all printing jobs are sent in PostScript format. All printing data is sent to the printer servers with LPD protocol (Line Printer Daemon Protocol, RFC1179). The PostScript is a de-facto standard as printing format in UNIX, and there are many tools that convert any data to the PostScript. So, we leave this conversion to the user side with the GUI printing panel, which will be presented in Section 5.

First of all, we focused on how to count pages of each printing job (for the requirement A1). For this, two methods could be considered.

Method A: Before sending a job to a printer, the printer server asks the printer its current counter value c_{bef}. After sending the job, the server asks again a updated counter value c_{aft}. The value of $c_{aft} - c_{bef}$ is regarded as the number of pages on the job.

Method B: Before sending a job to a printer, the printer server counts the number of pages of the job by using internal PostScript engine (e.g., Ghostscript).

Method B might cause mis-counting depending on printer's specification, since the specification of the internal Postscript engine is usually different from the printer's one. Therefore, we adopt Method A, which directly accesses the printer's PostScript engine, in order to eliminate such the specfication problem.

The UNIX's `lpd` is an ordinary printer spooler for the LPD protocol. It has basic features such as to forward the received jobs, and to obtain simple account information. Advanced features like counting pages are left to other filter programs invoked from `lpd`.

In Method A, the printer server has to communicate with the printer several times while forwarding the job to the printer. Also, we need other sophisticated features such as job cancellation and accounting for other requirements. Hence, we found it difficult to controll such complex mechanisms with the ordinary `lpd`.

To cope with the problem, we adopt an enhanced printer spooler LPRng (lpr Next Generation). The LPRng has the following features.

- Filters and accounting can be defined independently. Hence, an external accounting program can be invoked.
- It is possible to forward a job to other printer after applying filters to the job.
- It has options for flexible access restrictions for a printing job (which is relevant to the requirement A4).

By using the third feature above with the feature $F2$ in Section 3.2, the requirement A4 is achieved.

3.3 Customizing LPRng and ifhp

We installed the LPRng on the printer servers, together with an input filter program `ifhp` which works well with our printers. To achieve the administrator requirements, we have performed the following customizations and settings. The capitelized words are programs newly implemented by us.

1. We applied a wrapper, called IFECIP, to `ifhp`. For each job of a user, IFECIP looks up a database of printing quota (presented in Section 4). If the user exceeds his/her quota, then IFECIP returns an error. Otherwise, IFECIP passes the job to `ifhp`.
2. `ifhp` executes a script `accounting.sh` before sending a job to a printer. Hence, we added a program PAGECOUNT, which asks the printer its current counter value (c_{bef}, see Section 3.2), to `accounting.sh` [1].

[1] `ifhp` is supposed to have a PS method to count pages. However, it did not work with our printers.

3. If the job is successgully sent to the printer, `ifhp` activates again the accounting.sh, in which PAGECOUNT gets the counter value (c_{aft}).

By those customizations, we can put c_{bef} and c_{aft} in a log file for each job. With the log file, we can easily attain the requirements A1, A2 and A3.

PAGECOUNT needs bi-directional communication with a PostScript engine in the printers. However, the LPD protocol is uni-directional. It can send only simple messages such as error codes. As other options, we could use two-way protocol such as AppleTalk, Centronix, SNMP with Print MIB(RFC1759). However, since our printers support a special mode called 'raw socket mode', we are using it. The 'raw socket mode' provides a data communication between a server and a PostScript engine in the printer on a unique TCP port. Some venders (e.g., HP) implements it in their printers.

Even if the PostScript engine completes the job, the printer may be still printing the data to physical papers. Therefore, if PAGECOUNT asks the current counter value while the printer is still working, PAGECOUNT would miscount pages less than actual output. To overcome the problem, we have implemented the following algorithm:

1. PAGECOUNT sends a "pagecount" Postscript command to a printer (command is "`statusdict begin pagecount (*) print = flush end`"). If no response received from the printer within 10 seconds, PAGECOUNT gives up the request.

2. PAGECOUNT receives a current counter value (let it be $prev$) from the printer.

3. PAGECOUNT waits 5 seconds.

4. Do the above 1. again.

5. PAGECOUNT receives the current counter value (let it be new) from a printer.

6. If $prev \neq new$ (which means that printer is still working), goto 2 with "$prev := new$"

7. PAGECOUNT outputs the total number of print(new).

By this algorithm, PAGECOUNT can strictly count the paper.

4 Managing print quotas

In this section, we show how to manage the printing database and the printing quota (A2, A3).

4.1 Analyzing logs and creating database

In order to account the printing, we create the database from the raw log file of `ifhp`. The flow chart of this procedure is as follows.

Raw log (`ifup`) $\rightarrow$ Readable printer log $\rightarrow$ Database

This procedure is performed twice a day. We firstly translate the `ifhp`'s log files to readable printer log files. Next we analyze the readable logs and calculate the number of papers that the users have printed so far. We create a data file for each PC lab.

The list of PC labs is

$B = \{$"kyositu", "kanri", "bungaku", "hoken", "igaku", "kougaku", "kyotu", "ryugaku", "yakugaku", "genbun", "hougaku", "jinka", "kisokou", "rigaku", "sigaku"$\}$.

$$\begin{cases} \text{"kyositu":} & \text{Classroom in the center.} \\ \text{"kanri":} & \text{Administrator's computers.} \\ \text{the others:} & \text{Computer labs of the faculties.} \end{cases}$$

The database is the set of plain text files. The data file is named by the computer lab, and it is constructed with the lines of the user name and the number of papers which the user has printed at this computer lab.

All of the data files are plain text files. That is why it is easy to debug the program and edit the data files by UNIX commands and script languages.

4.2 Limitation rule

The quota of papers for a user is determined by the combination of user's faculty and the faculty of the computer labs. The limitation rule of our system is designed for any combination of faculties and computer labs. Moreover, we can gain the quota for each user at each lab. For example, we show some limitation rules.

Ex. 4.2.1 The quota of a standard user is 300 papers.

Ex. 4.2.2 The quota of a student of Department of Science is 1000 papers for the computer lab "rigaku". And it is 300 papers for the other computer labs.

Ex. 4.2.3 The quota of a administrator is no limitation for "kyositu" and "kanri". And it is 100 papers for each computer labs.

4.3 Checking quota

In order to get the information about printing account, we use the command `lpacctinfo`. This command first reads the configuration files of limitation rules, and accesses the database. Then it returns the information about printing and check whether the number of printed papers is over the quota or not.

It is very easy to use `lpacctinfo`. We show some examples.

Ex. 4.3.1 Show the number of printed papers.

```
lpacctinfo -u user -h host -p printer \
                    --print-page
```

Ex. 4.3.2 Show the quota.

```
lpacctinfo -u user -h host -p printer \
                    --print-border
```

Ex. 4.3.3 Show the numbers of remained printable papers.

```
lpacctinfo -u user -h host -p printer \
                    --print-remainder
```

The returned value of `lpacctinfo` give us the permission or the prohibition to printout.

5 GUI applications for Linux

This section presents GUI applications for printing, implemented on the Linux clients. The applications cover the user requirements from U1 to U4 described in Section 2.

5.1 GUI printing panel

Compared to other GUI-based operating systems (e.g., MS Windows), Linux lacks user-friendly interface for printing. Although Linux has a variety of commands and filters for advanced printing, most of them are used from shell command lines. Hence, users must be familiar with Linux shell and sophisticated options, which is often difficult for unskilled students.

To cope with the problem, we have developed a GUI, called *LPRGUI*, shown in Figure 1. LPRGUI is a front-end of the Linux standard printing command `lpr`, and provides GUIs for advanced printing options. LPRGUI was written in the Perl/Tk language with about 1,200 lines of code. Table 1 summarizes features implemented in LPRGUI.

In the Linux clients, all applications pass the printing data (plain text or PostScript) to `lpr` via a file or standard input. LPRGUI first intercepts the data and spools it as a temporary file, converting into PostScript if necessary (by using `a2ps`). Since the target data is always stored as a temporary file in PostScript, we can use and manipulate the data without touching the original data.

Next, for the temporary file, LPRGUI launches the related filters and/or commands with given options, and apply modification. In order to allow users to specify the options easily, LPRGUI provides graphical interfaces such as buttons, entry boxes and radio buttons.

Finally, LPRGUI passes the data with modified options to `lpr`, which completes printing operation. In the following subsections, we will explain features of LPRGUI.

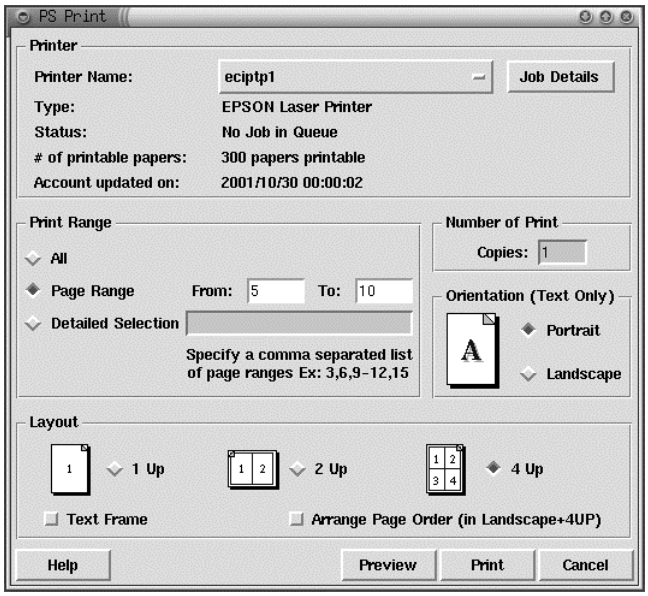

Figure 1. LPRGUI:GUI printing panel

5.2 Satisfying user requirements

First, let us see the features covering the requirement U2 in Table 1. Page selection and n-up are primary features to shrink pages. The options for them can be specified easily by radio buttons and entry boxes. The orientation also helps the saving, since it can prevents documents with large width from being split off multiple pages. Thus, the requirement U2 can be satisfied.

Next, by pressing preview button, LPRGUI launches the filters for shrinking, which are mentioned above, to modify the (temporary) file, then activates a postscript viewer `gv` with the modified file. Thus, user can easily preview the printing image through `gv`, just by mouse clicks and/or simple key typing, which fulfills the requirement U1.

LPRGUI generates a pull-down list of printers from `/etc/printcap`, by which user can easily choose printers in the lab. When a printer is selected, LPRGUI looks up the database with the lpacctinfo (see Section 4), and shows the balance: how many papers the user can use from the selected printer. If the user exceeds the quota, LPRGUI disables “print” button and tells the user “quota exceeded”. The user cannot print any more pages through LPRGUI. Thus, the requirement U3 is satisfied.

Pressing “Job details” button launches another GUI application, called *LPRMGUI* shown in Figure 2, which manages print jobs in printer queues. LPRMGUI is a wrapper of `lpq` and `lprm`. It shows the status and printer queue of each selected printer. Also, users can easily select unnecessary jobs from listbox widgets, and delete them by

Table 1. Features of LPRGUI

Requirements	Features	Description	Related commands, filters
U1:preview	Preview	Preview the printing image	`gv`
U2:shrink	Page selection	Select printing pages	`psselect`
	n up	Multiply pages per a paper	`psnup`
	Orientation	Choose page orientation	`a2ps`, `ps2ps`
U3:quota	Printer selection	Select printers	`lpr`, `printcap`
	Show balance	Show the number of printable paper	lpacctinfo
	Disable printing	Disable print operation when quota is exceeded	lpacctinfo
U4:job	Job details	View printer status and queue	LPRMGUI, `lpq`
	Jobs cancellation	Cancel print jobs	LPRMGUI, `lprm`
Misc.	Raw data filtering	Block raw data (e.g., image, sound) to be printed	`file`
	Bilingual support	Change language on labels and buttons	Environment variable `LANG`

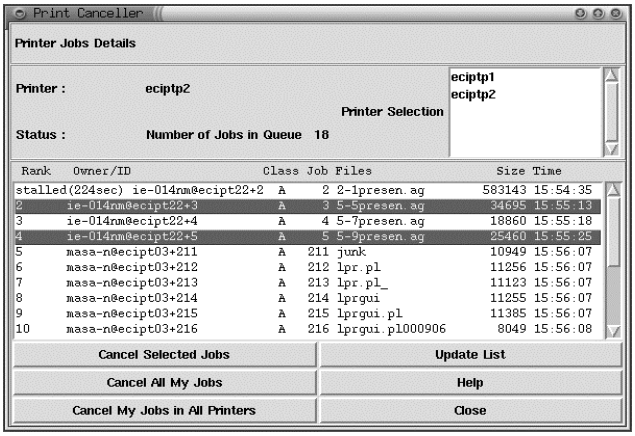

Figure 2. LPRMGUI: Print job manager

pressing buttons. LPRMGUI has options for the job cancellation, which helps user cancel multiple jobs simultaneously. Hence, even unskilled users can manage their print jobs without knowing the shell commands, which satisfies the requirement U4.

Therefore, all of user requirements from U1 to U4 are satisfied by deploying LPRGUI.

5.3 Other features

As shown Misc. in Table 1, there are other useful features implemented on LPRGUI, though they are not directly related to the requirements.

The first feature is raw data filtering. The data to be printed is not always PostScript (or plain text) processed by applications. That is, users can directly type `lpr` from the terminal with raw data files (image, sound, etc.). If not managing raw data appropriately, this results in waste of papers. To prevent this situation, LPRGUI first checks whether a given file is raw data or not with `file` command. If the file is raw data, LPRGUI rejects it by prompting user: "This file is not printable".

Next feature is bilingual support. We have many foreign students as users. Due to the language barrier, some of them are using the Linux clients in English mode. So, we implemented a feature to switch the language (Japanese and English) on labels of GUIs. The switching is done depending on the value of the environment variable `LANG`.

6 Statistics

Table 2 shows statistics gathered by the printing database presented in Section 4. The table contains statistics of first semesters (6 months) in years 2000 and 2001. According to our experience in managing the educational computer system, we knew that for educational use, about 300 pages of printing are enough for a student per a semester.

In the year 2000, the printing subsystem had not been deployed yet. Although we announced the students to prevent waste printing, the usage was up to students' morals only. We can see in Table 2 that on average, 60.2 pages were printed by a student per a cemester. 110 students had printed more than 300 pages, and the total number of pages they printed over the 300 pages is 88,250, which is a quarter of all printed pages!

In 2001, we deployed the proposed system and set the quota to be 300. The number of students who had printed was increased [2]. However, the total number of pages remains almost the same as the one in 2000. The average is reduced to 52.9, which is 12% reduction per a user. Moreover, the number of pages printed over the quota is decreased to 23,649, which is 6.7% of all pages.

[2] We consider that this is due to curriculum changes.

Figure 3 shows a frequency distribution of users. The hotizontal axis represents ranges of printed pages. The vertical axis plots the number of users who did the printings within the ranges. From the graph, we can see that the frequency distribution of 2001 inclines towards the smaller ranges. This fact shows that more users have printed less pages after we deployed the proposed system.

Strictly speaking, it is very difficult to define what the waste printing is, and thus, difficult to conclude that the proposed system reduces the *actually waste* printing. However, we can say at least that the proposed system has successfully achieved significant reduction on total printing cost.

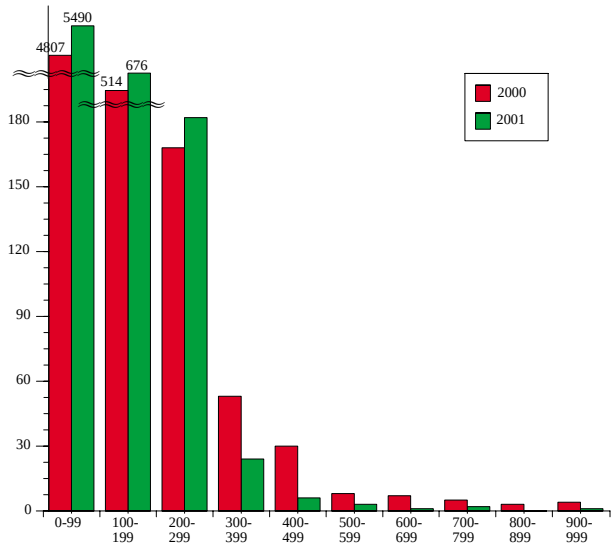

Figure 3. Frequency distribution of users

7 Discussion and concluding remarks

References

[1] Tetsuro Tanaka, Koji Ando, Akira Yoshioka: “User management in multiple OS”, Technical Report of IPSJ, DSM 16-9, pp.49–54 (1999.11) (In Japanese).

[2] Shin Maruyama, Yasuo Fujii and Jun’ichi Nakamura: “Imprementation and management of printing subsystem with quota feature” Proc. of Education Conference of Infomation Processing, p.342 (2001.10) (In Japanese).

[3] Takefumi Ogawa, Masahide Nakamura, Koichi Kondo, Hiroyuki Oosaki, Hideo Masuda, Junji Kitamiti, Michio Nakanishi: “Course tools and management tools for an educational system on Linux”, Proc. of Education Conference of Information Processing, pp.239–242 (2000.12) (In Japanese).

[4] RICOH, “Ridoc IO Gate”, http://www.ricoh.co.jp/IPSiO/utility/iogate/index.html

[5] CANON, “NetSpot Suite”, http://www.canon-sales.co.jp/Product/appli/netspotsuite/index-j.html

Table 2. Total statistics and over-quota printing (300 pages per a semester)

Year	2000	2001
# of users having printed	5,822	6,608
# of pages printed	350,509	349,256
Average # of pages per a user	60.2	52.9
# of users having exceeded the quota	110 [1.90%]	37 [0.5%]
# of pages printed above the quota	88,250 [25.2%]	23,649 [6.7%]