RC23369 (W0410-063) October 11, 2004
Computer Science

IBM Research Report

A J2EE Application for Process Accounting, LPAR Accounting, and Transaction Accounting

C. Eric Wu, William P. Horn
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

A J2EE Application for Process Accounting, LPAR Accounting, and Transaction Accounting

C. Eric Wu, *Senior Member, IEEE* and William P. Horn

***Abstract*—Accounting is critical for information technology budgeting and chargeback. Traditional accounting in UNIX/Linux systems is known as process accounting, in which an accounting record is created when a process ends. System administrators then aggregate accounting records based on individual users, groups, or projects. As Web and application servers as well as databases handle requests and transactions for multiple entities in various Web applications and services, LPAR accounting and transaction accounting become increasingly critical for service providers in shared resource environments. In this paper we present the design and implementation of a J2EE accounting application for resource usage metering. For process accounting the resulting system can generate usage reports by projects, by groups, by users, by commands, or by a combination of these identifiers. For dynamically changing partitions it generates reports for shared resources including CPUs, memories, disks, file systems, and network interfaces. For transaction accounting it generates reports based on account classes provided that applications are instrumented. It is the first known J2EE accounting application for UNIX/Linux transaction accounting.**

***Index Terms*—ARM transactions, resource usage, project accounting, process accounting, transaction accounting**

I. INTRODUCTION

INFORMATION technology (IT) is usually viewed as critical to modern businesses and organizations. The increases in user numbers, demands for new technologies and complexities of computing environments has frequently caused IT costs to grow faster than other costs. As a result, organizations are often unable or unwilling to justify expenditure to improve services or develop new ones. To understand whether an IT organization is doing its best, it has to understand the true cost of providing a service and manage those costs professionally. This in turn requires accounting for computing resources.

Most UNIX and Linux systems today provide some form of process accounting that records a collection of information for each and every process completed by the kernel into a file. The recorded information is typically specified in a header file. Originated from either UNIX System V or BSD, the methodology of process accounting on many UNIX variants was developed long time ago, and the tools available are useful yet primitive. An overview of process accounting can be found in [1].

Interval accounting and project accounting improve process accounting in different ways. Interval accounting is required because long-running processes for applications such as databases and web servers can run for months without termination, resulting in significant delays in accounting data collection. Interval accounting enables intermediate accounting records to be produced and collected at intervals specified by a system administrator. A number of accounting records instead of one will then be created periodically for a long-running process.

Project accounting is a capability that records a project identifier along with the process accounting data. A project identifier is a tag defined by system administrator and is associated with processes via project assignment policies. Having process accounting capabilities available on a version of UNIX does not guarantee the availability of project accounting. There have been a number of UNIX variants that provide project accounting, including UNICOS' Cray System Accounting (CSA) from Cray, Irix Comprehensive System Accounting from SGI [2], Solaris Extended Accounting [3] from Sun Microsystems, and AIX Advanced Accounting from IBM [4, 5].

Recent advances in dynamic logical partitions (LPARs) allow multiple, independent operating systems running in a single server, one on each dynamic LPAR [6]. As the workload of an operating system changes, resources including CPUs and memory in the dynamic LPAR can

C. Eric Wu is with IBM T.J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598. He can be reached at 914-945-2629; fax 914-945-2944; e-mail: cwu@us.ibm.com.

William P Horn is with IBM T.J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598.

expand and shrink over time without requiring a reboot of the operating system. If each account entity is assigned with an LPAR, the accounting facility must provide LPAR accounting to record the expansion and shrinkage of computing resources for usage-based billing over time.

When applications handle transactions for multiple clients in a shared service environment, process accounting or LPAR accounting cannot provide accurate usage due to the lack of information on account entities. A database in a Web environment, for example, may handle all the requests on behalf of a local user regardless of request origin. In this case only the Web server or the application server can identify the users of its internal transactions for accounting purposes. To provide accurate accounting on resource usage, the account entity or class must be passed along with transaction requests. The ability to perform transaction accounting is critical in a shared service environment, in which work occurs as transactions flow through systems across networks.

In this paper we discuss the design and implementation of the J2EE accounting application for AIX systems based on its Advanced Accounting facility. The resulting J2EE application is capable of generating reports for process accounting, LPAR accounting, and transaction accounting, with management operations and interfaces for both program-to-program communication and human interactions.

II. A J2EE Accounting Application

As service oriented architecture [7] becomes more popular, it makes sense to create a Web service for resource utilization reporting. A Web service is self-describing, in that Web Service Description Language (WSDL) is used to describe service operations, including the structure of its input and output parameters. Web service clients do not need to have prior knowledge about the operation APIs of the service. They typically learn from the WSDL description of the service before invoking its operations. Thus, WSDL descriptions eliminate the potential problems resulting from changes in operation API. The resulting Web services effectively alleviate the problem for users to learn the underlying technology, i.e. the AIX Advanced Accounting facility.

Figure 1 shows the overall design of the J2EE accounting application. There are two Web services: a reporting Web service and a management Web service. We use a Model-View-Control (MVC) design for easy maintenance. The Web services provide the model with business functions and handle the backend, including the accounting facility, its accounting files, and an optional database where past records from various host systems are accumulated and aggregated. A servlet is used as the control module as well as the Web services' client to pass operation requests from users to the Web services, and deliver operation results as Java beans. A number of Java Server Pages (JSPs) are developed for presentations and human interactions. Alternatively a client could be a program that communicates with the Web service directly.

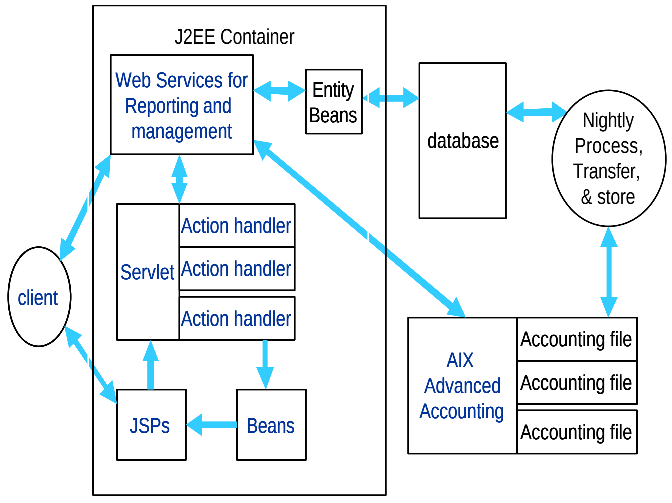


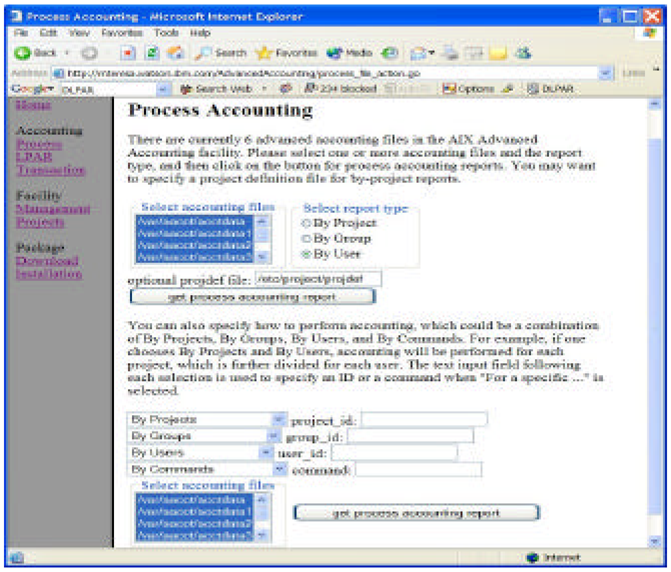
Figure 1 Architecture of the J2EE Accounting Application

The reporting Web service is responsible to aggregate accounting records and generate reports based on user requirements. The management Web service is developed to manage the accounting facility with operations such as "start accounting", "stop accounting", "check status", and will be discussed in the next section.

The reporting Web service defines and implements aggregate routines for process accounting, LPAR accounting, and transaction accounting. The routines for process and transaction accounting are relatively straightforward, while the ones for LPAR accounting are partitioned to get statistics for CPU & memory, file systems, disks, network interfaces, etc., one for each resource category. Two more operations are used to get the names of the accounting files currently available in the accounting facility, and to get accounting file information such as the timestamps when it was first and last accessed, host name, partition name and ID, etc. Typically accounting file information is displayed at the beginning in each report.

A. *Process Accounting*

Figure 2 shows the web page for process accounting where a user can use a browser to interact with the reporting Web service and submit requests. The JSP web page lists all accounting files currently available in the accounting facility. A user selects accounting files and the report type, which could be by project, by group, or by user. An optional project definition file may be provided to convert project IDs to more meaningful project names. Reports can also be generated by a combination of project, group, user, and command, as shown in the lower part of the snapshot.


Process Accounting

There are currently 6 advanced accounting files in the AIX Advanced Accounting facility. Please select one or more accounting files and the report type, and then click on the button for process accounting reports. You may want to specify a project definition file for by-project reports.

optional projdef file: /etc/project/projdef

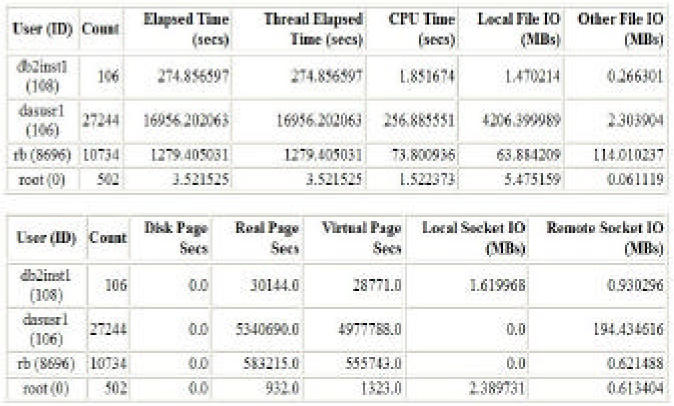
You can also specify how to perform accounting, which could be a combination of By Projects, By Groups, By Users, and By Commands. For example, if one chooses By Projects and By Users, accounting will be performed for each project, which is further divided for each user. The text input field following each selection is used to specify an ID or a command when "For a specific ..." is selected.

Select report type: ○ By Project ○ By Group [✓ By User]

Figure 2 Web page for process accounting.

Figure 3 shows a process accounting by-user report in which accounting file information is omitted. It reports for each user the total elapsed time, total thread elapsed time, CPU time, local and remote file I/O, disk page, real page, etc. If a combination of by-project, by-group, by-user, and by-command is used, e.g. by-user and by-command, each row in the table would be further divided into a number of lines, one for each command the user has issued. The interface also allows user to specify a specific identifier (e.g. a specific user with a specific project) or command for report generation.

User requests are carried out by the reporting Web service through the servlet, which is implemented as the client of the Web service and could be installed in a different system to interact with interactive users.



User (ID)	Count	Elapsed Time (secs)	Thread Elapsed Time (secs)	CPU Time (secs)	Local File IO (MBs)	Other File IO (MBs)
db2inst1 (108)	106	274.856597	274.856597	1.851674	1.470214	0.266301
dasusr1 (106)	27244	16956.202063	16956.202063	256.885551	4206.399989	2.303904
rb (8696)	10734	1279.405031	1279.405031	73.800936	63.884209	114.010237
root (0)	502	3.521525	3.521525	1.522373	5.475159	0.061119

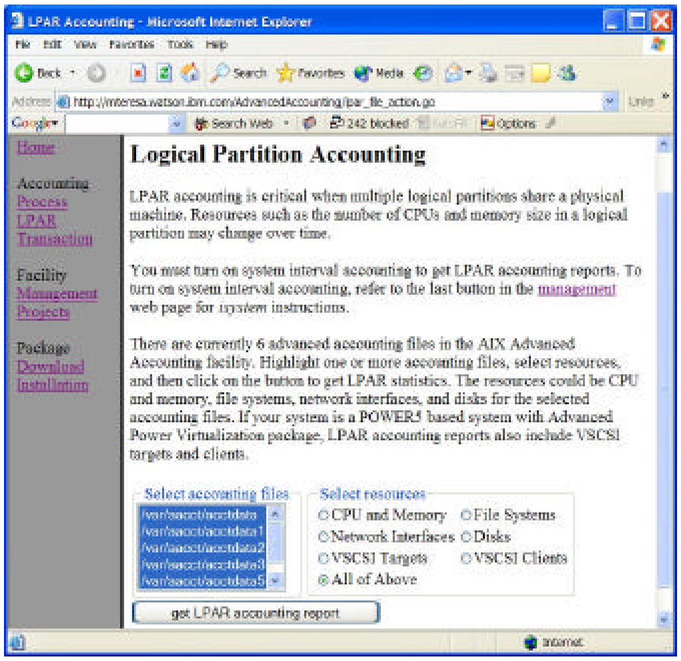
User (ID)	Count	Disk Page Secs	Real Page Secs	Virtual Page Secs	Local Socket IO (MBs)	Remote Socket IO (MBs)
db2inst1 (108)	106	0.0	30144.0	28771.0	1.619968	0.930296
dasusr1 (106)	27244	0.0	5340690.0	4977788.0	0.0	194.434616
rb (8696)	10734	0.0	583215.0	555743.0	0.0	0.621488
root (0)	502	0.0	932.0	1323.0	2.389731	0.613404

Figure 3 Report by users for process accounting

The reporting Web service uses Java NIO mapped files for high performance file access. Byte order, i.e. big-endian and little-endian, can be specified based on machine architecture in Java NIO mapped files. The choice allows us to run command line utilities and read AIX advanced accounting files on Intel based PCs, and will be helpful for future expansions in a heterogeneous environment.

B. *LPAR Accounting*

LPAR accounting is important when account entities use individual logical partitions. Since each partition expands and shrinks over time based on its workload and needs, the accounting facility must capture these changes of resource allocation. LPAR accounting report relies on the reporting Web service to aggregate accounting records for four types of resources: CPU & memory, file systems, disks, and network interfaces. For systems with Power 5 processors and the Advanced Power Virtualization package, resources also include virtual SCSI targets and clients.


Logical Partition Accounting

LPAR accounting is critical when multiple logical partitions share a physical machine. Resources such as the number of CPUs and memory size in a logical partition may change over time.

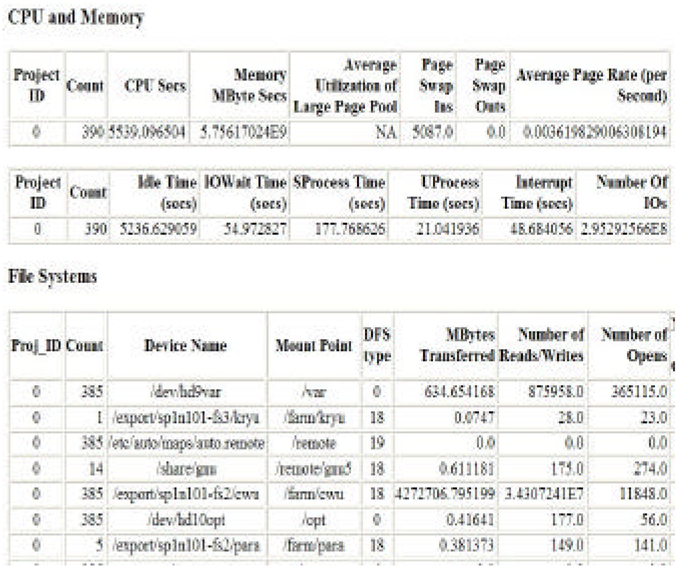
You must turn on system interval accounting to get LPAR accounting reports. To turn on system interval accounting, refer to the last button in the management web page for *system* instructions.

There are currently 6 advanced accounting files in the AIX Advanced Accounting facility. Highlight one or more accounting files, select resources, and then click on the button to get LPAR statistics. The resources could be CPU and memory, file systems, network interfaces, and disks for the selected accounting files. If your system is a POWER5 based system with Advanced Power Virtualization package, LPAR accounting reports also include VSCSI targets and clients.

Select resources: ○ CPU and Memory ○ File Systems ○ Network Interfaces ○ Disks ○ VSCSI Targets ○ VSCSI Clients [✓ All of Above]

Figure 4 Web page for LPAR accounting

Figure 4 show the JSP web page for LPAR accounting. The JSP page requests the list of available accounting files through the servlet, which in turn contact the reporting Web service for the list. A user then selects accounting files and resource report before requesting the report. Figure 5 shows an LPAR accounting report for CPU & memory, and a partial report for file systems.



CPU and Memory

Project ID	Count	CPU Secs	Memory MByte Secs	Average Utilization of Large Page Pool	Page Swap Ins	Page Swap Outs	Average Page Rate (per Second)
0	390	5539.096504	5.75617024E9	NA	5087.0	0.0	0.003619829006308194

Project ID	Count	Idle Time (secs)	IOWait Time (secs)	SProcess Time (secs)	UProcess Time (secs)	Interrupt Time (secs)	Number Of IOs
0	390	5236.629059	54.972827	177.768626	21.041936	48.684056	2.95292566E8

File Systems

Proj_ID	Count	Device Name	Mount Point	DFS type	MBytes Transferred	Number of Reads/Writes	Number of Opens
0	385	/dev/hd9var	/var	0	634.654168	875958.0	365115.0
0	1	/export/sp1n101-fs3/kryu	/farm/kryu	18	0.0747	28.0	23.0
0	385	/etc/auto/maps/auto.remote	/remote	19	0.0	0.0	0.0
0	14	/share/gnu	/remote/gnu5	18	0.611181	175.0	274.0
0	385	/export/sp1n101-fs2/cwu	/farm/cwu	18	4272706.795199	3.4307241E7	11848.0
0	385	/dev/hd10opt	/opt	0	0.41641	177.0	56.0
0	5	/export/sp1n101-fs2/para	/farm/para	18	0.381373	149.0	141.0

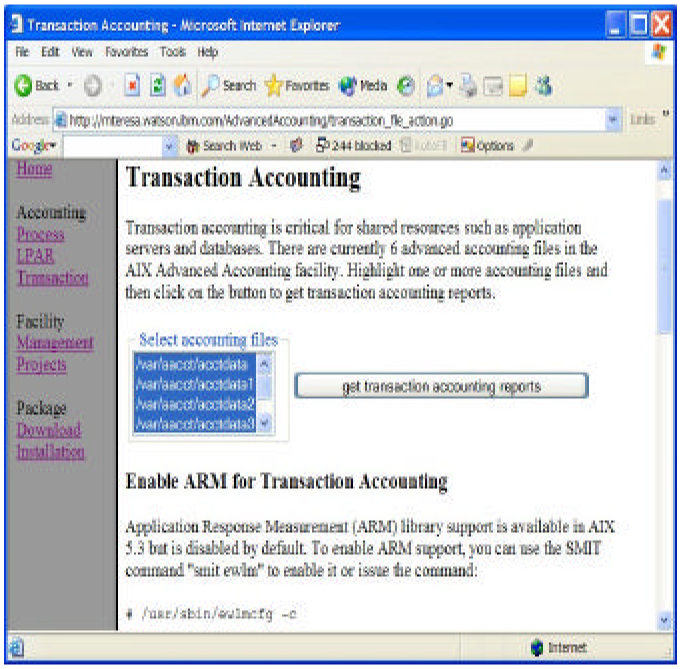
Figure 5 LPAR accounting report for CPU, memory, and file systems

LPAR accounting report for CPU & memory includes CPU allocation in seconds and memory allocation in Mbytes-seconds to indicate the sum of resources over time. If an LPAR is allocated with 1000 Mbytes memory for the 1st minute followed by 2000 Mbytes for the next minute, its memory allocation would be 180000 Mbytes-seconds. Other reported items include average utilization of large page pool, page swap in/out, average page rate, total system time, total user time, etc. Report for file systems includes device names, mount points, Mbytes transferred, number of read/write operations, number of opens, number of creates, and number of locks. The table for disks reports the total number of disk transfers, total number of read blocks, and total number of write blocks for each disk partition. For network interfaces the total number of I/Os and bytes transferred in Mbytes are reported. These two tables are not shown in Figure 5 to save space.

C. *Transaction Accounting*

Transaction accounting differs from process accounting and LPAR accounting in that account entities are not known to the operating system due to the dynamic features of individual transactions in a shared service environment. As a result it is required to add calls in the middleware and applications before and after each transaction. The Application Response Measurement (ARM) is a standard describing a common method for integrating enterprise applications as manageable entities [8, 9].

Figure 6 shows the JSP web page for transaction accounting. The web page also illustrates how to enable ARM service and authorize non-root users. We will discuss how to instrument applications for transaction accounting along with a report snapshot in section IV.


Transaction Accounting - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://mteresa.watson.ibm.com/AdvancedAccounting/transaction_file_action.go
Links
Google Search Web 244 blocked Options
Home
Accounting
Process
LPAR
Transaction
Facility
Management
Projects
Package
Download
Installation
Select accounting files
get transaction accounting reports
Internet

Transaction Accounting

Transaction accounting is critical for shared resources such as application servers and databases. There are currently 6 advanced accounting files in the AIX Advanced Accounting facility. Highlight one or more accounting files and then click on the button to get transaction accounting reports.

/var/aacct/acctdata
/var/aacct/acctdata1
/var/aacct/acctdata2
/var/aacct/acctdata3

Enable ARM for Transaction Accounting

Application Response Measurement (ARM) library support is available in AIX 5.3 but is disabled by default. To enable ARM support, you can use the SMIT command "smit ewlm" to enable it or issue the command:

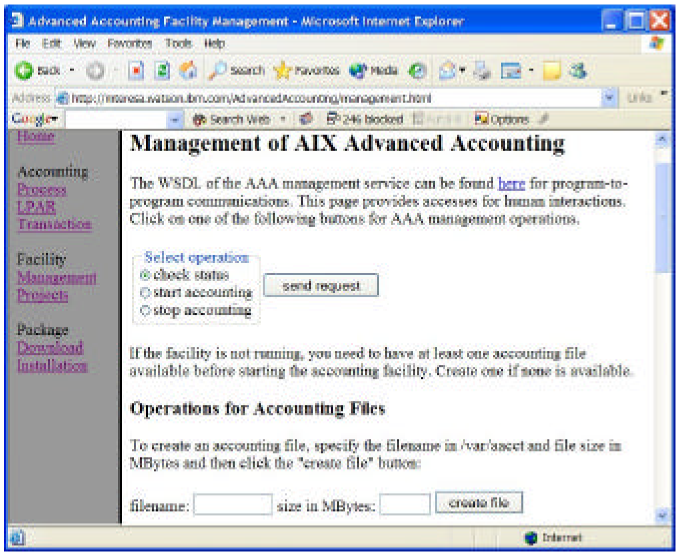
```
# /usr/sbin/ewlmcfg -c
```

Figure 6 Web page for transaction accounting

III. MANAGEMENT AND PROJECT ASSIGNMENT

The management Web service is developed to manage the accounting facility with operations such as creating accounting files and defining new projects. It exports underlying accounting and project control command line interfaces through Web service operations `acctctl()` and `projctl()`. Critical functions such as `createFile()` are built as Web service operations.

Web services help integrate multiple steps into one service operation. The `defineProj4App()` operation, for example, is used to assign a predefined project to an application when the application is running by a given user or group. The operation includes six steps. First it opens an existing administration file in a specified directory or creates the file if it does not exist, then creates a temporary file in which a UNIX sed append command is stored. The execution of the sed command against the original administration file creates a new administration file. After a successful execution the original administration file is renamed as the backup, the new administration file is renamed as the current one, and the temporary file is removed at the end. The resulting web service operation hides the details from the user and thus effectively eliminates the pain for a user to learn and use the underlying accounting facility.


Advanced Accounting Facility Management - Microsoft Internet Explorer

http://interesa.watson.ibm.com/AdvancedAccounting/management.html

Home

Accounting
Process
LPAR
Transaction

Facility
Management
Projects

Package
Download
Installation

Management of AIX Advanced Accounting

The WSDL of the AAA management service can be found here for program-to-program communications. This page provides accesses for human interactions. Click on one of the following buttons for AAA management operations.

Select operation
[✓ check status]
start accounting
stop accounting

send request

If the facility is not running, you need to have at least one accounting file available before starting the accounting facility. Create one if none is available.

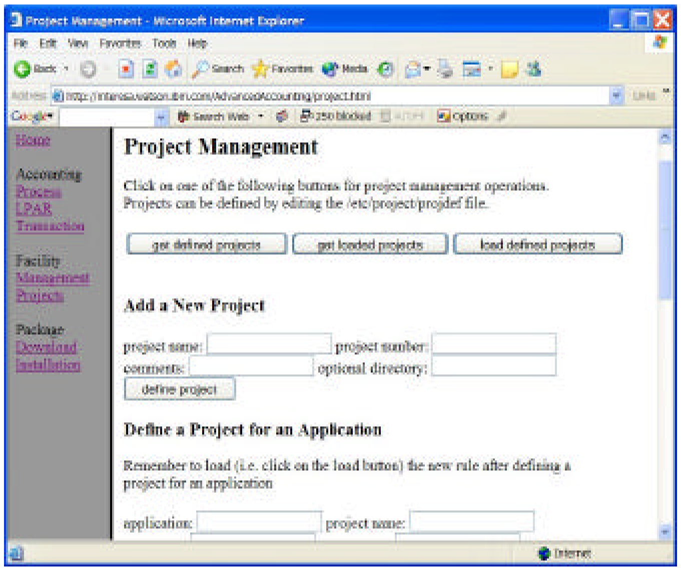
Operations for Accounting Files

To create an accounting file, specify the filename in /var/aacct and file size in MBytes and then click the "create file" button:

filename: size in MBytes: create file

Figure 7 Web page for accounting facility management

Figure 7 shows the JSP page for accounting facility management. It allows a user to perform operations such as start accounting, stop accounting, check current accounting status, create an accounting file with a specified file size, etc. Many other operations and descriptions are provided to help interactive users issue management requests, such as manage the accounting facility, enable ARM service, and authorize non-root users.


Project Management - Microsoft Internet Explorer

http://interesa.watson.ibm.com/AdvancedAccounting/project.html

Home

Accounting
Process
LPAR
Transaction

Facility
Management
Projects

Package
Download
Installation

Project Management

Click on one of the following buttons for project management operations.
Projects can be defined by editing the /etc/project/projdef file.

get defined projects | get loaded projects | load defined projects

Add a New Project

project name: project number:
comments: optional directory:
define project

Define a Project for an Application

Remember to load (i.e. click on the load button) the new rule after defining a project for an application

application: project name:

Figure 8 Web page for project management

Figure 8 shows the web page for project management. It is similar to the management page and uses the same management Web service to export project control for the accounting facility. In addition to define a project for an application, it allows other operations such as add a new project, get defined projects, get loaded projects, load defined projects, query defined projects, etc.

IV. ARM Transactions And Instrumentation

The Application Response Measurement (ARM) standard allows developers to extend their enterprise management tools directly to applications. It offers a comprehensive end-to-end management capability that includes measuring application availability, application performance, application usage, and end-to-end transaction response time [8, 9]. The ARM standard is promoted by the Open Group to maintain service quality in workload management [10]. The IBM Enterprise Workload Manager [11] uses ARM-instrumented applications to provide users with a workload management environment that monitors and reports performance statistics.

As a standard, the ARM API [12] is made up of a set of function calls, including arm_register_application(), arm_register_transaction(), arm_start_transaction(), and arm_stop_transactoin(). Using an ARM library to instrument transactions inside an application, one can measure transaction response time by reading system clock at the beginning and the end of the transaction and subtract the first reading from the second. To correlate one transaction in an application with its sub-transactions in other applications, the instrumented application calls the arm_start_transaction() routine and receives a correlator from the ARM library. The application then passes the correlator to the next application when it initiates the request for the sub-transaction. A correlator is an opaque object to the outside world, and typically contains information such as host name, IP address, application ID, and transaction ID to uniquely identify the transaction.

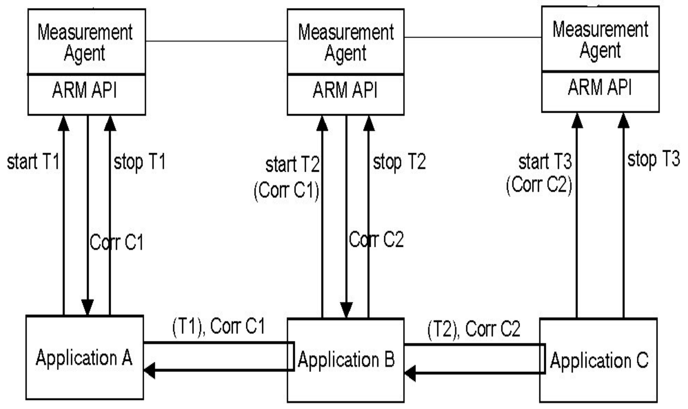


Figure 9 ARM-instrumented applications passing correlators

To classify transactions for advanced accounting in a shared service environment, we typically generate account class information at the edge of the network when a request from the outside world is received. The class information may be predefined based on URI or certain policy, or derived from the registered/login user. The class information is then stored in the returned correlator.

Figure 9 illustrates the passing of correlators among three networked applications. At the edge of the network application A receives a transaction request. It then calls the arm_start_transaction() routine and receives correlator C1, where the class information is stored. When application A is ready to initiate the sub-request to application B, it sends correlator C1 along with the request. Upon receiving the request application B calls the arm_start_transaction() routine and receives correlator C2, in which the class information is retained. Application B sends correlator C2 along with its own sub-request to application C. The pattern continues passing correlators from one application to another and retaining the class information along with the transaction flow.

The AIX 5.3 operating system comes with an ARM library. We developed three network programs that act as an edge server, middleware server, and backend server, respectively, to illustrate how they work together using the ARM library for transaction accounting. All three server programs communicate through IPC sockets. Helper routines are used to randomly select an account class at the edge server and to insert class information into a correlator.

Listing 1 An ARM-instrumented code segment

```
#include <arm4.h>
...
arm_register_application("applicationName", NULL, 0,
NULL, &aId);
arm_start_application(&aId, "groupName", "1234", 0,
NULL, &applHandle);
arm_register_transaction(&aId, "transactionName",
NULL, 0, &tranBuffer, &tId);
sk = prepareSocket(&socket, &readyfd, port_number);
for (;;) {
  receiveMessage(sk, &readyfd, buffer);
  memcpy(&parCorr, buffer, sizeof(parCorr));
  ...
  arm_start_transaction(&applHandle, &tId, &parCorr,
    ARM_FLAG_BIND_THREAD, &tBuffer, &tHandle,
  &corr);
  ...
  memcpy(buffer, &corr, sizeof(corr));
  sendMessage(backendHost, bport_number, buffer);
  receiveMessage(sk, &readyfd, buffer);
  ...
  sendMessage(frontendHost, fport_number, buffer);
  arm_stop_transaction(&tHandle, ARM_STATUS_GOOD, 0,
    NULL);
}
rc = arm_stop_application(&applHandle, 0, NULL);
```

In addition to account class, the accounting facility captures other names including the application group, application name, and transaction name. Listing 1 shows a simplified ARM-instrumented code segment for the middle tier. The application name, application group, and transaction name are defined in their corresponding ARM routines at the point of application registration, starting the application, and transaction registration, respectively. During the execution they are captured in accounting records created by the accounting facility.

The loop in Listing 1 receives a message, stores account class in its parent correlator, and calls the arm_start_transaction() before initiating the transaction. After account class is copied into the returned correlator, the correlator is sent to the backend host along with the request. Eventually the middle tier receives the returned message from the backend host and responds to the edge server before calling the arm_stop_transaction() routine.

Account classes can be specified through transaction identity properties or context properties. A property is a <name, value> pair. Identity properties are used to specify properties that never change values, and context properties are used for information that changes over time. Since individual transactions are carried out for various account classes in a shared service environment, the account class for each transaction should be specified using transaction context properties. In ARM 4.0 API context property names are defined in the transaction buffer when registering the transaction and the value is provided in another transaction buffer when calling arm_start_transaction().

Listing 2 Code segment to define account class

```
#include <arm4.h>
...
const char *names[1] = { "EWLM:AIX:Account Class" };
arm_subbuffer_tran_identity_t tIden;
arm_subbuffer_t *sbarray[1];
arm_buffer4_t tranBuffer;

tIden.header.format = ARM_SUBBUFFER_TRAN_IDENTITY;
tIden.identity_property_count = 0;
tIden.context_name_count = 1;
tIden.context_name_array = names;
sbarray[0] = &(tIden.header);
tranBuffer.count = 1;
tranBuffer.subbuffer_array = sbarray;
...
rc = arm_register_transaction(&aId,
"transactionName", NULL, 0, &tranBuffer, &tId);
```

Listing 2 illustrates how to define account class using a transaction property. The identity property count is set to zero, indicating that no identity property is specified. The context property name for account class must be "`EWLM:AIX:Account Class`" or its value won't be captured in its transaction accounting record. Note that identity properties can be defined in the same buffer, although we define none in the code segment.

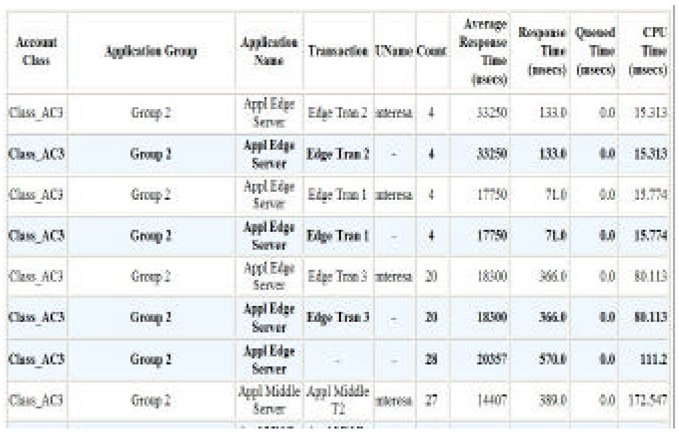
Listing 3 Code segment to specify context property value

```
#include <arm4.h>
...
arm_subbuffer_tran_context_t tCtx;
arm_char_t *values[1];
arm_buffer4_t tBuffer;

values[0] = getClass(&parCorr);
tCtx.header.format = ARM_SUBBUFFER_TRAN_CONTEXT;
tCtx.context_value_count = 1;
tCtx.context_value_array = values;
sbarray[0] = &(tCtx.header);
tBuffer.count = 1;
tBuffer.subbuffer_array = sbarray;
...
arm_start_transaction(&applHandle, &tId, &parCorr,
  ARM_FLAG_BIND_THREAD, &tBuffer, &tHandle, &corr);
```

Listing 3 shows a code segment to specify account class as a context property value. It gets the account class from the parent correlator. The account class is stored in the buffer which is passed in as a parameter when calling the ARM arm_start_transaction() routine. Although both code segments use only one sub-buffer, they can be easily modified with multiple sub-buffers for multiple properties. Since correlators are opaque, care is needed to store the account class in a correlator so that there is no conflict with the ARM library.

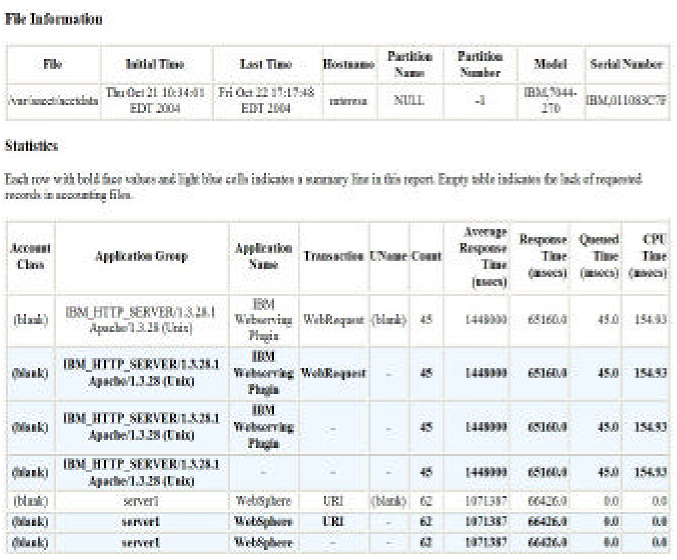
Once an application is ARM-instrumented, we need to enable the ARM service and authorize non-root users. This can be done through simple commands as specified in the transaction accounting web page or through the management web page. Because account class is passed through correlators from one application to the next, the accounting facility captures account class values for individual transactions. Along with the application name, application group, and transaction name, this in turn enables report generation for transaction accounting using ARM classes.



Account Class	Application Group	Application Name	Transaction	UName	Count	Average Response Time (usecs)	Response Time (msecs)	Queued Time (msecs)	CPU Time (msecs)
Class_AC3	Group 2	Appl Edge Server	Edge Tran 2	interesa	4	33250	133.0	0.0	15.313
Class_AC3	**Group 2**	**Appl Edge Server**	**Edge Tran 2**	-	**4**	**33250**	**133.0**	**0.0**	**15.313**
Class_AC3	Group 2	Appl Edge Server	Edge Tran 1	interesa	4	17750	71.0	0.0	15.774
Class_AC3	**Group 2**	**Appl Edge Server**	**Edge Tran 1**	-	**4**	**17750**	**71.0**	**0.0**	**15.774**
Class_AC3	Group 2	Appl Edge Server	Edge Tran 3	interesa	20	18300	366.0	0.0	80.113
Class_AC3	**Group 2**	**Appl Edge Server**	**Edge Tran 3**	-	**20**	**18300**	**366.0**	**0.0**	**80.113**
Class_AC3	**Group 2**	**Appl Edge Server**	-	-	**28**	**20357**	**570.0**	**0.0**	**111.2**
Class_AC3	Group 2	Appl Middle Server	Appl Middle T2	interesa	27	14407	389.0	0.0	172.547

Figure 10 A partial transaction accounting report

Figure 10 shows a partial transaction accounting report using the ARM-instrumented 3-tier programs. A transaction accounting report includes the average response time, total response time, total queued time, and total CPU time for each combination of account class, application group, application name, transaction name, and transaction user name. The transaction user name could be anything meaningful, for example, the host name where the application was running on. A row with light-blue (or grey) cells indicates a summary of its previous rows. The working prototype demonstrates the first known J2EE accounting application for transaction accounting produced in a UNIX/Linux system. It demonstrates transaction accounting in a networked environment in which transactions flow through systems across networks.



File Information

File	Initial Time	Last Time	Hostname	Partition Name	Partition Number	Model	Serial Number
/var/aacct/acctdata	Thu Oct 21 10:34:01 EDT 2004	Fri Oct 22 17:17:48 EDT 2004	interesa	NULL	-1	IBM,7044-270	IBM,011083C7F

Statistics

Each row with bold face values and light blue cells indicates a summary line in this report. Empty table indicates the lack of requested records in accounting files.

Account Class	Application Group	Application Name	Transaction	UName	Count	Average Response Time (usecs)	Response Time (msecs)	Queued Time (msecs)	CPU Time (msecs)
(blank)	IBM_HTTP_SERVER/1.3.28.1 Apache/1.3.28 (Unix)	IBM Webserving Plugin	WebRequest	(blank)	45	1448000	65160.0	45.0	154.93
(blank)	**IBM_HTTP_SERVER/1.3.28.1 Apache/1.3.28 (Unix)**	**IBM Webserving Plugin**	**WebRequest**	-	**45**	**1448000**	**65160.0**	**45.0**	**154.93**
(blank)	**IBM_HTTP_SERVER/1.3.28.1 Apache/1.3.28 (Unix)**	**IBM Webserving Plugin**	-	-	**45**	**1448000**	**65160.0**	**45.0**	**154.93**
(blank)	**IBM_HTTP_SERVER/1.3.28.1 Apache/1.3.28 (Unix)**	-	-	-	**45**	**1448000**	**65160.0**	**45.0**	**154.93**
(blank)	server1	WebSphere	URI	(blank)	62	1071387	66426.0	0.0	0.0
(blank)	**server1**	**WebSphere**	**URI**	-	**62**	**1071387**	**66426.0**	**0.0**	**0.0**
(blank)	**server1**	**WebSphere**	-	-	**62**	**1071387**	**66426.0**	**0.0**	**0.0**

Figure 11 Transaction accounting report for Web server and WebSphere Application Server

IBM HTTP Server and WebSphere Application Server have been ARM-instrumented for IBM Enterprise Workload Manager (EWLM). Figure 11 shows the transaction accounting report for these two commonly used applications. The first table at the top shows file information, including the initial time and last time when accounting records were written into the file, host name, partition ID and name, etc. It can be seen from the report that their ARM instrumentation does not include the context property value for "`ELWM:AIX:Account Class`" and therefore the account class in display is "(blank)". This is because they were ARM instrumented for EWLM before the AIX Advanced Accounting facility became available.

V. Summary and Work In Progress

We discussed the design and implementation of the J2EE accounting application in this paper. In addition to traditional process accounting, the J2EE accounting application handles LPAR accounting for dynamic LPARs whose resources expand and shrink over time, and transaction accounting for ARM account classes in a shared service environment. A separate command line utility was also developed to generate reports without a Web environment. Both the command line utility and the J2EE accounting application are working and available for download through IBM AlphaWorks.

Future work includes the extensions to handle Linux accounting systems, to store and retrieve records in/from the optional database, and to create reports for a given time period. The extensions to handle Linux systems and use a database are required for usage-based billing in a heterogeneous environment. Periodical or nightly jobs would help convert and move accounting data to the reporting server, and we expect a handy accounting package once the extensions are implemented.

References

[1] V. G. Hazlewood, "Unix Accounting Magic," *Sys Admin*, vol. 7, no. 2, pp. 11 – 13, February 1998.

[2] Silicon Graphics Inc., "Comprehensive System Accounting," Chapter 5, *IRIX Admin: Resource Administration*, Document 007-3700-015, July 2003.

[3] Sun Microsystems, "Extended Accounting," Chapter 7, *System Administration Guide: Resource Management and Network Services*. Part number 806-4076-10, 2002.

[4] L. Browning, "Advanced Accounting for AIX 5L Version 5.3," *IBM White Paper*, July 2004, at http://www-1.ibm.com/servers/aix/whitepapers/aix_accounting.pdf,

[5] IBM, "Understanding the Advanced Accounting subsystem," *IBM document SC23-4882-00*, August 2004.

[6] IBM, "Dynamic Logical Partitioning in IBM eServer pSeries," http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/dlpar.pdf, *IBM White Paper*, October 2002.

[7] T. Erl, "Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services," *Prentice Hall*, April 2004.

[8] M. Johnson, "Monitoring diagnosing application response time with ARM," *Proceedings of the IEEE Third International Workshop on System Management*, pp. 4 – 13, April 1998.

[9] The Open Group, "System Management: Application Response Measurement (ARM) API," *Open Group Technical Standard*, July 1998.

[10] The Open Group, "Application Manageability and Quality of Service," http://www.opengroup.org/qos/app-manageability.

[11] IBM, "IBM Virtualization Engine: IBM Enterprise Workload Manager," Version 1, Release 1, *IBM Manual*, August 2004.

[12] The Open Group, "Application Response Measurement (ARM) Issue 4.0 – C Binding," *Open Group Technical Standard*, 2003.