USENIX Association

Proceedings of the 17th Large Installation Systems Administration Conference

San Diego, CA, USA
October 26–31, 2003

Designing a Configuration Monitoring and Reporting Environment

Xev Gittler and Ken Beer – Deutsche Bank

ABSTRACT

The Configuration Monitoring and Reporting Environment (CMRE) is a tool designed to collect and report on the many configuration details of systems within an enterprise. It is designed to gather information on systems about which initially very little is known. CMRE needs few prerequisites in order to do its job. CMRE is modular, flexible and runs on many different platforms. It is written in a combination of Perl, Korn shell and PHP and uses proprietary as well as open-source software. CMRE currently collects data on thousands of UNIX and Windows systems at Deutsche Bank world wide. This paper will describe the conditions that led to the need for such a tool, its design and limitations as well as the difficulties found along the way. This paper will also touch upon our performance monitoring tool (PMRE), whose data we use to help us understand our systems environment.

Accommodating Rapid Growth

Over the last ten years, Deutsche Bank (DB) has executed a strategy which has led to its rapid growth as a dominant player in the investment banking industry. This strategy involved organic growth, acquisitions and mergers. The natural outgrowth of this strategy is that our systems environment contains a diverse set of systems built in a number of organizations, with differing standards and procedures. There was an obvious case for improving the infrastructure by consolidating our infrastructure to provide an environment that is more efficient, uniform and easy to service. We wanted to provide a platform that would allow us to more easily answer questions from management about configuration and security issues.

Some of the questions that we wanted to be able to more easily answer include:

- Security: List all machines that are patched appropriately
- Capacity Planning: List number of machines with empty CPU slots.
- Physical moves: List all machines pointing to a particular DNS server

Given that the different environments came with their own standards and procedures, we wanted to develop a uniform mechanism for retrieving information to enable us to understand our environment as a whole. Once we had a uniform method, we could further improve our environment by adding proactive management tools.

Our goal is to proactively manage all our systems, including all configuration options. We want to be able to specify from a central location exactly how a system is configured without having to wait for an SA to specify the data, but which requires some initial consistency, which we cannot easily get until we have a good view of what the systems look like.

Design Specifications

When designing the CMRE system, our primary goals were to put a system in place that could be deployed quickly (within a few months), did not require significant effort or expense, yet vastly improved our ability to understand our environment. We examined a number of systems deployed internally as well as external products, such as Explorer, cfengine, and SNMP-based solutions, but ultimately determined that the simplest, most flexible method was to develop a single homegrown system, based on the design principles of an existing tool already deployed in one of our branch offices.

We needed to develop an inexpensive, simple, powerful, multiplatform solution. The client software would run as root on every UNIX system at DB globally, but would switch to the minimal permissions required for each particular command. Therefore, we needed to ensure that the software would be safe and would not add unexpected load to our client systems. We could not risk using software that we did not completely understand. On the other hand, we could not build a lab to fully test the software because we did not know what kinds of systems were installed. Finding, testing, and adapting another's solution was therefore deemed to be as time consuming as writing a simple custom application.

In order to determine the primary functionality of the system, we conducted extensive discussions with the various groups that we considered stakeholders in this project and took their needs into account:

- System Administrators – need to examine the systems in detail, and view consolidated system data in order to troubleshoot, upgrade and move systems.
- Security/Audit – need to examine summary reports and drill down to see specific problems.

- Developers – need to analyze the impact of their applications on their systems and determine resources available to them.
- Engineering groups – need to find systemic bottlenecks and need to be able to plan for infrastructure changes.
- Business Managers – need to examine resource utilization and system status across their business.
- Senior Management – need to see high level overviews of status and performance across all systems globally, summarized by group.

Based on the above, we determined that our configuration monitoring system should provide:

- **Accessible configuration information** – The basic feature of the system should be to take data that was previously only available in many different, scattered, often inaccessible locations and gather and present it from a single location.
- **Planning data** – We required a comprehensive and easily accessible company-wide picture of systems in order to plan for future growth. For instance, without knowing how many machines were running Solaris 2.6, we could not assess the size of the task of upgrading these systems when the operating system was at end-of-life. Nor could we determine when applications could be retired if we did not know the application use metrics.
- **Problem Detection and Repair** – We required the ability to identify and correct system specific problems in all affected machines as soon as the problem arose or even before it became an issue. For instance, when we found an issue, we wanted to check all systems that might be similarly affected and either patch the system or mark it so that when the problem occurred, we would be able to quickly determine the corrective course of action. Similarly, we needed to be able to identify and repair security and audit issues. Without system configuration information, we had to manually examine each and every system to determine whether security or audit issues affected the system.
- **Historical Configuration Information** – We wanted to gather and store a significant amount of configuration data about each system in our environment, including historical information for comparison purposes.
- **Flexibility** – We wanted to add additional information collectors and reports easily as new requirements arose.
- **Support for Multiple Operating Systems** – We wanted the system to work across all our supported operating systems, gathering detailed information on all systems.
- **Audience Specific Views** – We wanted to be able to present support group level summarizations of configuration data, such as a list of all systems within a group running a particular operating system, or running on particular hardware, and view summary information via a web front end. In addition, we wanted to present management with global summarizations of configuration data
- **Data Filtering** – We wanted to filter data, such as flagging all systems that did not meet a particular standard, such as a minimal operating system level, or a particular security patch.
- **External Data Connectivity** – We wanted to tie into our inventory, monitoring, security and other databases, and help us improve the quality of that data.

The overarching design goal was to create a single portal where users could find any and all configuration information they might require, regardless of whether we changed the underlying tool used to collect the information. We wanted a single place that provided all the configuration information that one might require.

Recognizing that information gathering and reporting requirements would grow in sophistication as people began to use these systems, the system had to be designed with the flexibility to grow in unexpected ways. This required the ability to tailor reports to various audiences (e.g., CIOs, IT Senior Management, applications/development managers, support managers, and system and database administrators).

The Challenges

Once we had a basic design concept we needed to make it work within the diverse environment that currently existed. This section will discuss the various issues that we ran across and how we addressed them. Many of these challenges are a direct result of combining organizations and systems as part of our company's rapid growth.

Multiple Operating Systems and Revisions

As is typical in investment banks, we are required to support a number of different operating systems and versions supporting different businesses. We needed to support a range of operating systems, including multiple versions of Solaris, AIX, Linux, HPUX and Windows. While we were willing to support different sets of scripts for Windows and Unix, we wanted to maintain a single set of scripts for all Unix platforms. We also wanted output to be as similar as possible from all operating systems.

No Guarantee Of Tools On Client Systems

Because we had grown from multiple environments, one of the first major hurdles we came across was that there was not consistent tool set available across all systems. Our initial thought was to write the tool in Perl, but we were unable to do so because most of our machines did not have Perl installed. We considered deploying Perl with the client software, but decided coordinating multiple versions of Perl would be more trouble than it was worth. We chose Korn shell (ksh) as a common denominator and did not use any external tools that did not install by default on all operating systems we were using.

File System Layout and Disk Space Allowance

Since our systems were installed by a wide variety of groups that were working from different specifications, we could not rely on any particular file system to be available. Some machines had /opt, some had /apps, some /usr/local. In some cases, /usr/local was NFS mounted read-only. Many of the systems had minimal disk configurations, so we had to ensure that both our scripts and their output fit in 10-30 MB of space.

Minimal Client Processing

Many of our business areas are extremely sensitive to any additional load placed on their systems. Because of this restriction we had to minimize the processing of data on the client system. Our experience is that should our software interrupt a running application, the application's business owner could forbid the running of our code on their systems and force us to use a custom solution (or do without).

Different Trust Mechanism

Throughout the bank we had many different mechanisms for allowing access to hosts, including SSH based golden hosts[1] for large groups and a variety of other mechanisms for smaller groups. For CMRE, however, we would require access to all systems across the bank.

Unexpected Configurations

When writing our scripts, we tested across a reasonable variety of systems. However interesting things can happen when, for instance, we run a disk

[1]A golden host is a well secured machine that has secure access to other machines. It is used for a variety of system administration purposes.

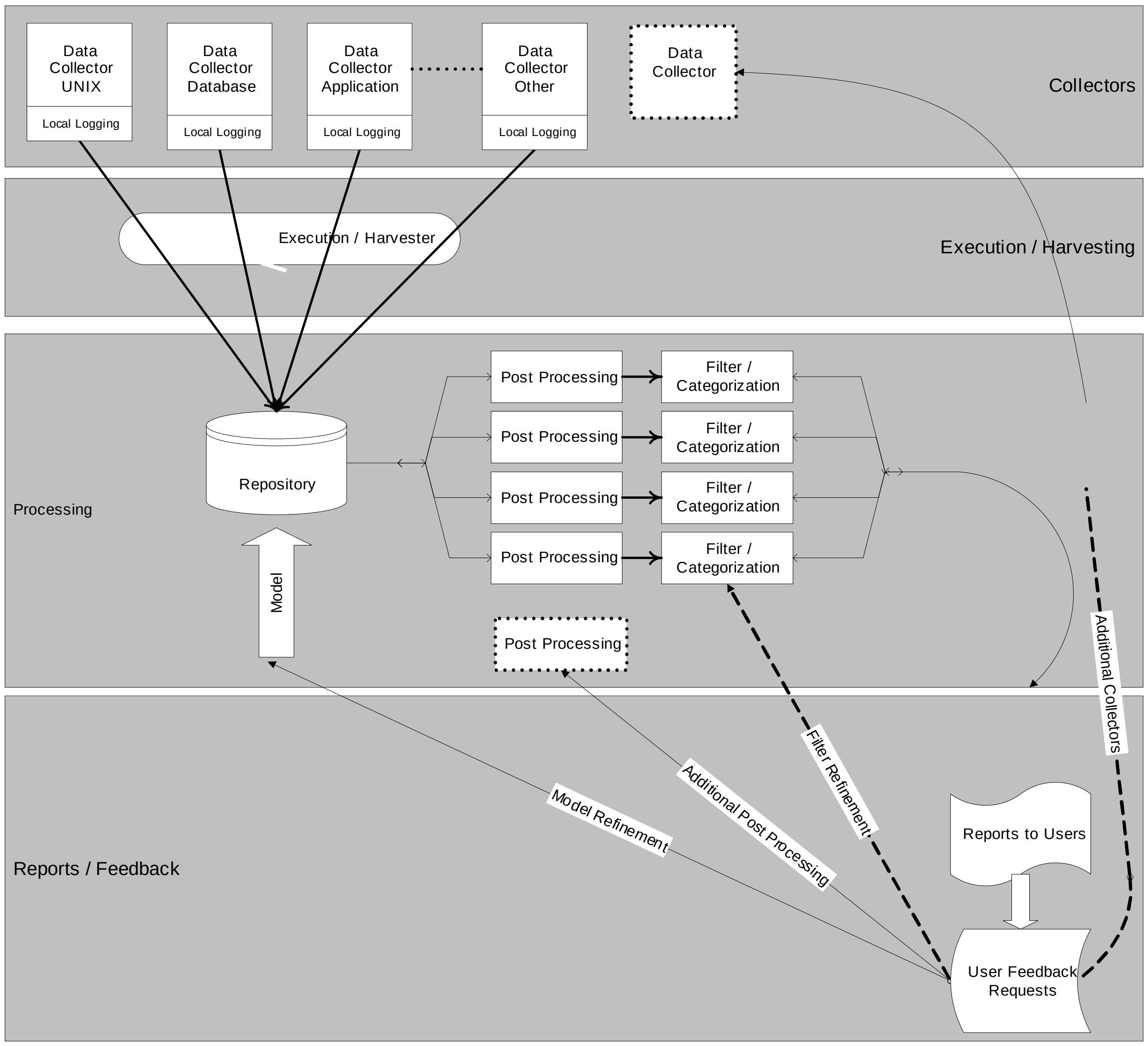


Figure 1: Block diagram of CMRE.

information command on a system with over 3,000 attached disks. As we find edge-cases like this, we modified our scripts to be even more careful and generic. Often the unexpected actions happen because of an incorrect system configuration. The impulse is to say, "You should not put swap on a CDRW" or "you should not run 10 copies of Oracle on an Ultra1," however there may be perfectly valid reasons for these configurations. It cannot be a requirement of CMRE for a system to be configured in a particular way. CMRE's job is to collect data from every system, regardless of how oddly configured. Improving the infrastructure is a separate task.

The Tool

The tool that we designed is part of a framework we call the Monitoring and Reporting Environment (MRE). This paper discusses only the Configuration (CMRE) and (briefly) the Performance (PMRE) components.

The CMRE architecture is based on a simple, modular framework. CMRE consists of

Category	Sys Type	Sys Vers	**C**md/ **F**ile	Run as	Output Filename	Command or File	Description
Automount	All	All	C	Nobody	lsautomnt	ls -al /etc/auto_*	Get timestamp of automounter files
Automount	All	All	F	Nobody		/etc/auto_*	Get automounter files
Config	SunOS	All	C	Nobody	lsgrpsys	ls -al /etc/system	Get kernel config
Config	All	All	C	Nobody	lsvarspl	ls -laR /var/spool/cron	Get timestamps on cron directory tree
Config	All	All	C	Nobody	hostid	hostid	Get the unix host id
Config	All	All	C	Nobody	uptime	uptime	Get the time alive
Config	All	All	C	Nobody	date	date -u	Date Stamp
Config	All	All	C	Nobody	findperl	findperl.sh	Find where perl is
Config	All	All	F	Nobody		/etc/profile	Get the host default user profile
Config	All	All	F	Nobody		/etc/services	Get the local services file
Config	All	All	F	Nobody		/etc/protocols	Get the local protocols file
Config	All	All	F	Nobody		/opt/mitk5/etc/krb5.conf	MIT K5 config file
Config	All	All	F	Nobody		/opt/mitk5/etc/ad.conf	MIT K5 config file
Config	All	All	F	Root		/.login	root's login info
Config	All	All	F	Root		/.cshrc	root's login info
Config	SunOS	All	C	Nobody	lpstat	lpstat -t	Get printer config
Config	SunOS	All	C	Root	orcaperf	orcaperf.sh	Get the orca performance data
Config	SunOS	All	C	Nobody	dmesg	dmesg	SunOS specific command to get kernel messages
Config	SunOS	All	C	Nobody	uname	uname -a	Get the unix machine/OS level id info
Config	SunOS	All	C	Nobody	swap-s	swap -s	Get the active swap space
Config	SunOS	All	C	Nobody	swap-l	swap -l	List the swap areas
Config	SunOS	All	C	Nobody	pkginfo	pkginfo -l	Dump the verbose package listing
Config	SunOS	All	C	Nobody	modinfo	modinfo	Dump the kernel modules
Config	SunOS	All	C	Nobody	showrev	showrev -a	Show package and patch info
Filesystem	SunOS	All	C	Nobody	dfufs	df -kF ufs	Display all ufs filesystems
Filesystem	SunOS	All	C	Nobody	dfvxfs	df -kF vxfs	Display all veritas filesystems
Filesystem	SunOS	All	C	Nobody	dfnfs	df -kF nfs	Display all nfs mounts
Filesystem	SunOS	All	C	Root	veritas	veritas.sh	Display veritas info
Filesystem	SunOS	All	C	Root	veritas-ha	ha.sh	Display HA info
Filesystem	AIX	All	C	Nobody	mount	mount	Show what filesystems currently mounted
Filesystem	AIX	All	C	Nobody	lsfs	lsfs -c	AIX specific command to list filesystems
Filesystem	SunOS	All	F	Nobody		/etc/vfstab	SunOS specific file containing filesystem mount points
Filesystem	SunOS	All	F	Nobody		/etc/nfssec.conf	SunOS specific file for NFS security
Filesystem	AIX	All	F	Nobody		/etc/filesystems	AIX specific for filesystems
Filesystem	Linux	All	C	Nobody	dfext2-linux	df -kF ext2	Display all ext2 filesystems
Filesystem	Linux	All	C	Nobody	dfrfs-linux	df -kF reiserfs	Display all nfs mounts
Hardware	SunOS	All	C	Nobody	prtdiag	prtdiag -v	SunOS specific command displaying hardware info
Hardware	SunOS	All	C	Nobody	eeprom	eeprom -v	SunOS specific command displaying PROM settings

Table 1: Sample lines from master configuration file.

- Minimal collector software
- An execution component for running the collector software
- A harvesting component for gathering the data to a central location
- A processing component for aggregating and mining the data
- A browser-based reporting component

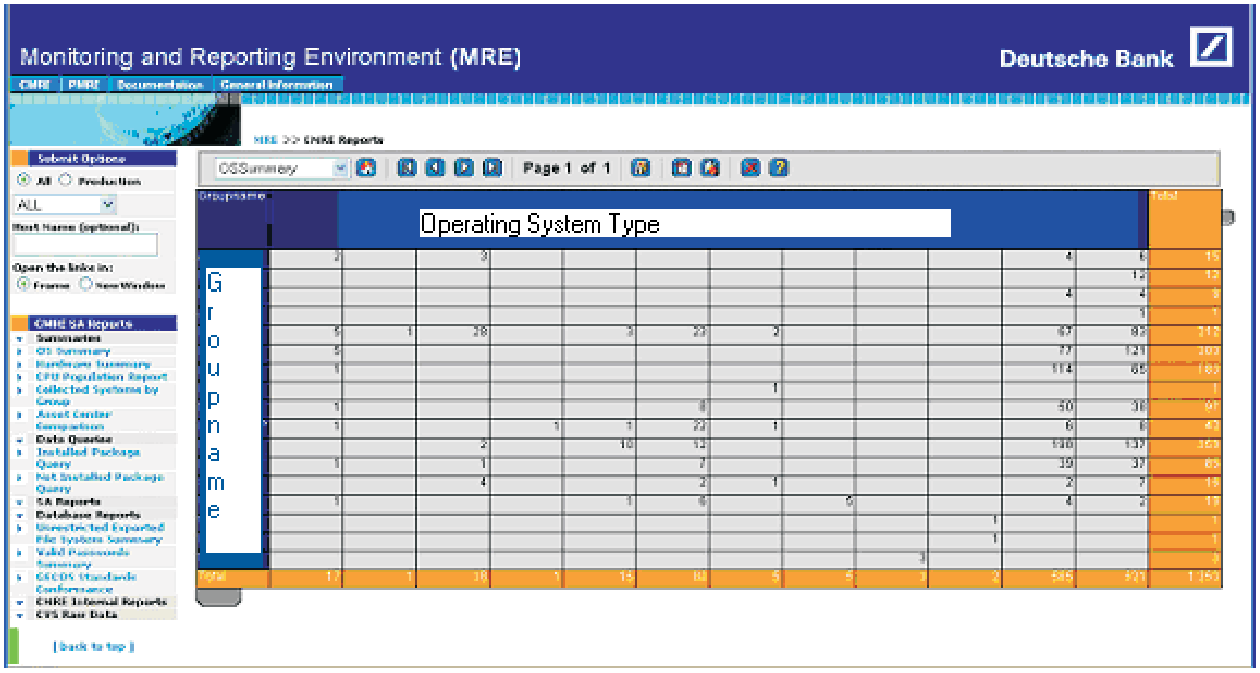
Figure 1 depicts the components of CMRE.

Collector

The collector is a set of Korn shell scripts and configuration files. Master.sh is the script that is run using either our installed framework product or our golden host infrastructure. It parses the appropriate commands file (described below) and uses it to acquire the appropriate files, and run the appropriate commands and scripts. Before they're run the commands and scripts are first wrapped in an OS-specific shell wrapper. This way collectors can be as simple as a single command yet still have appropriate variables set, traps, etc. STDOUT and STDERR are collected as well as measurements of the time the command took to complete and timeout alarms. Success or failure is recorded as well.

/.cshrc	/etc/inetd.conf	/etc/protocols	/etc/sudoers
/.login	/etc/init.d/*	/etc/rc*	/etc/syslog.conf
/etc/*.conf	/etc/inittab	/etc/resolv.conf	/etc/system
/etc/adsm	/etc/irs.conf	/etc/rpc	/etc/vfstab
/etc/auto_*	/etc/lvm	/etc/security/audit/config	/opt/mitk5/etc/ad.conf
/etc/cron.d/cron.allow	/etc/mail	/etc/security/audit/events	/opt/mitk5/etc/krb5.conf
/etc/cron.d/cron.deny	/etc/name_to_major	/etc/security/audit/objects	/var/adm/cron/cron.allow
/etc/default/*	/etc/named.conf	/etc/security/audit_control	/var/adm/cron/cron.deny
/etc/default/login	/etc/netgroup	/etc/security/audit_event	/var/adm/db/*
/etc/defaultdomain	/etc/netmasks	/etc/security/audit_user	/var/adm/loginlog
/etc/defaultrouter	/etc/netsvc.conf	/etc/security/failedlogin	/var/adm/messages
/etc/dfs/*	/etc/nfssec.conf	/etc/security/lastlog	/var/adm/sudo/sulog
/etc/exports	/etc/nodename	/etc/security/login.cfg	/var/adm/sulog
/etc/filesystems	/etc/nsswitch.conf	/etc/security/passwd	/var/adm/syslog.conf
/etc/group	/etc/oratab	/etc/security/user	/var/log/messages
/etc/hostname.*	/etc/passwd	/etc/sendmail.cf	/var/opt/oracle/oratab
/etc/hosts.equiv	/etc/path_to_inst	/etc/services	/var/spool/cron/allow
/etc/hosts	/etc/printers.conf	/etc/shadow	/var/spool/cron/deny
/etc/inet/*	/etc/profile	/etc/ssh/sshd_config	/var/yp/Makefile

Table 2: Files collected.



Groupname	Operating System Type												Total
	2		3								4	6	15
												12	12
											4	4	8
												1	1
	5	1	28		3	23	2				67	83	212
	5										77	121	203
	1										114	65	180
							1						1
	1					6					50	38	95
	1			1	1	22	1				6	6	43
			2		10	12					130	137	[illegible]
	1		1			7					39	37	85
			4			2	1				2	7	16
	1				1	6		6			4	2	[illegible]
										1			1
										1			1
									3				3
Total	17	1	38	1	15	88	5	5	3	2	595	[illegible]	1,353

Figure 2: Sample OS count report.

There is a single master configuration file that is maintained centrally. An extract of the master configuration file is presented in the next section. The file is delimiter separated that determines which files get collected and which commands or custom scripts are run on what version(s) of Unix (and as what user). The collectors are grouped into multiple classes such as "Network" and "Config", as well as operating system specific classifications. This configuration file is broken out into operating system specific files which are packaged into the distribution of the software. Master.sh reads the specific configuration file to determine what commands to execute.

Note that Master.sh does not do much parsing of output. The output of many system commands is designed to be human-, not machine-readable so complex parsing is often required. Instead, parsing is done at the back end. CMRE is effectively a distributed system, yet we intentionally do not take advantage of it for parsing. While we do have concerns about adding load, breaking things, and leaving data behind, the primary reason not to parse the data on the front end has to do with the difference in deployment speed between the parts of CMRE. Because of testing, change control, and approval processes, new versions of the client software can take weeks to roll out. Back end parsing, however, can be changed on the fly. This makes it more effective for us to acquire as much of the raw data as possible and parse it later. When we discover that we now require additional information, we can immediately change our parsers, rather than wait for the next rollout.

Master.sh keeps two copies of the data it collects on the client system – today's data and yesterday's data. After collection, master.sh compares what is new or different from yesterday's run and puts just those files and output in a compressed tar file to be collected later.

It takes a certain amount of finesse to write the collector software. Care must be taken to write code that is multiplatform, stable, and with low system

/bin/echo dummy	ifconfig -a	lsdev -C	psrinfo -v
/bin/last -100	ipcs -a	lsdev -Cc processor	Ptree
/usr/ucb/ps -auxwwe	Logins	lsfs -c	Pwstats.sh
arp -a	lpstat -t	lslpp -h all	Rpcinfo -p 127.0.0.1
crontab -l	ls -al /etc /etc/*.conf /etc/passwd /etc/shadow	lslpp -L all	Sacadm -L
date -u	ls -al /etc/auto.*	lsps -a	Secpasswd.sh
df -kF ext2	ls -al /etc/auto_*	lspvaix.sh	Showmount -e localhost
df -kF nfs	ls -al /etc/defaultrouter	lssrc -a	Showrev -a
df -kF reiserfs	ls -al /etc/group	lsvgaix.sh	Sundisks.sh
df -kF ufs	ls -al /etc/mail/*	modinfo	Swap -l
df -kF vxfs	ls -al /etc/named.conf	mount	Swap -s
diskformat.sh	ls -al /etc/netmasks	netstat -a	Sybase.sh
dmesg	ls -al /etc/networks	netstat -rn	Sysdef -d | egrep -v ‘(driver.not.attached|no.driver)’
domainname	ls -al /etc/printers.conf	network.sh	Sysdef -I
dsmc q files	ls -al /etc/rpc	odmget -q attribute=realmem CuAt	Sysinfo -level all -format report
dsmc q sched	ls -al /etc/sendmail.cf	odmget -q name=interface CuAt	Titancheck.sh
dumpadm	ls -al /etc/system	oracle-adds.sh	Uname -a
eeprom -v	ls -alR /etc	orcaperf.sh	Uname -aM
errpt	ls -alR /var/spool/lp /var/spool/print	oslevel	Uptime
fbconfig -list	ls -altr /var/yp/domainname	pkginfo -l	Veritas.sh
filepermlist.sh	ls -l /dev/rdsk	pmadm -L	who -a
findperl.sh	ls -l /etc/defaultdomain	prtconf -vD	Ypcat -k rpc.bynumber
getaixos.sh	ls -l /etc/inittab	prtconf | awk -F: ’/Memory/{print $2}’ | head -1	Ypcat -k ypservers
getrhosts.sh	ls -l /etc/resolv.conf	prtdiag -v	Ypmaps.sh
getsshkeys.sh	ls -l /tftpboot	ad l prtdiag10k -v | grep Memory Size | awk ’{print $3}’	Ypwhich
ha.sh	ls -laR /var/spool/cron	ps -efal	Ypwhich -m
hostid	Lscfg	ps auwwxe	

Table 3: List of executed commands.

overhead. As most of the scripts are run as root across all machines in the company, security must also be considered very carefully.

Sample Collector Configuration

Table 1 shows some sample lines taken from the master configuration file while Table 2 shows the names of the all the files collected. Table 3 shows the commands that currently executed across the various operating systems.

Execution / Harvesting

Execution and harvesting proved very challenging initially. We came up with two methods of performing these tasks, described below. Each technique executes the collector script, which creates a single file on the client, 'config.tar.Z'. The harvester then collects the file from each machine and renames it to a unique name. It then tars all of the individual files into a single tar file and uploads that file to the central processing machine via SSH.

Management Framework

Deutsche Bank uses a commercial system management framework product on a majority of our machines. In fact, it is a requirement that all systems in datacenters run the framework's monitoring agents. The framework gives us a secure method of distributing software, executing commands on remote machines, and pulling the data back to a central location. Where the framework is installed, we use it to install the software (under a common directory structure provided by the

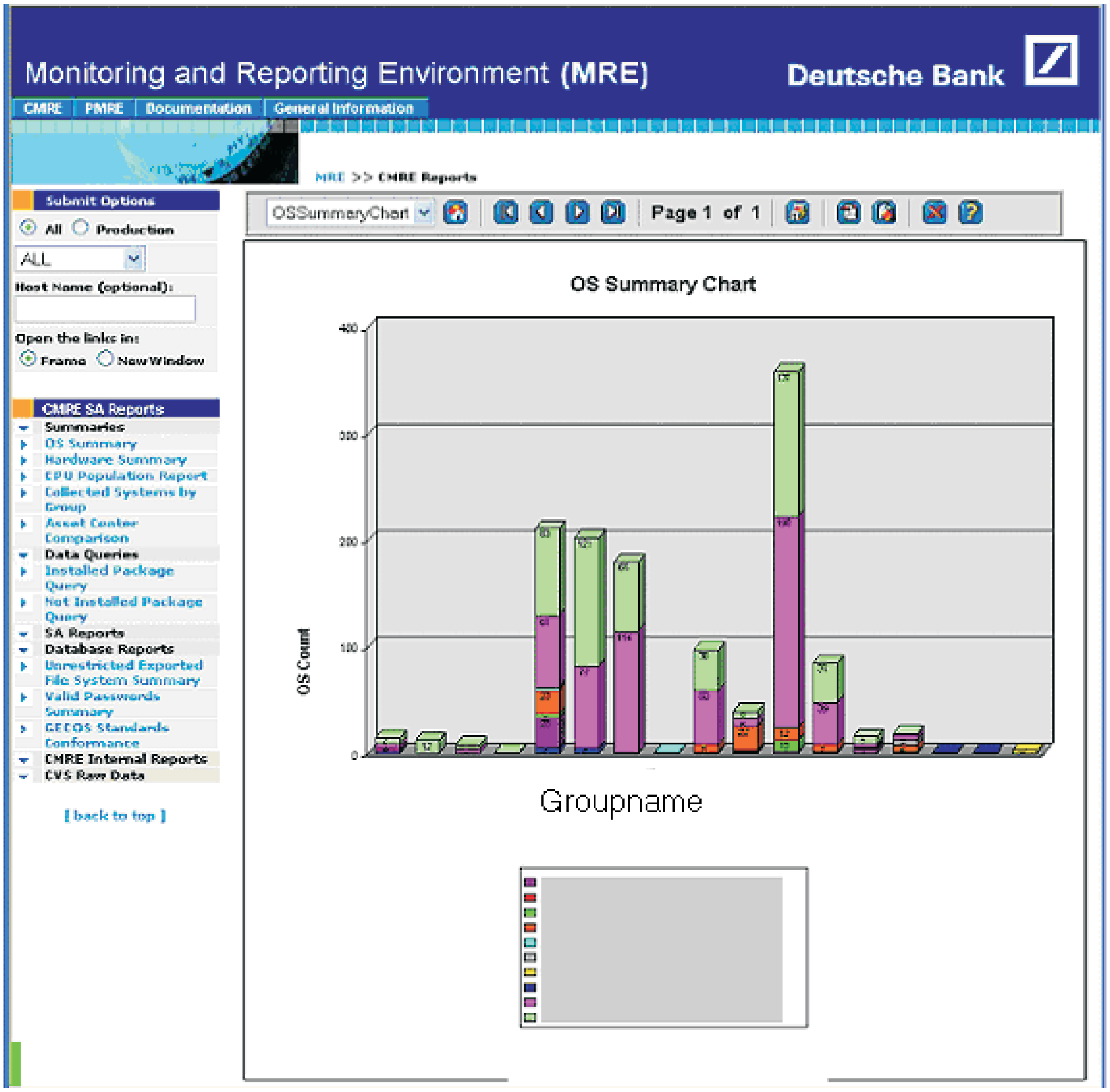


Figure 3: OS summary chart.

framework, so that we have a directory structure that we can count on), execute the collector on a nightly basis, and securely retrieve the information to the framework management nodes. From the framework management nodes we use SSH to get the aggregated data securely to our central CMRE processing server.

Golden Hosts

For those machines where the framework is not available, we have packaged the software including the appropriate crontab entries to execute the commands on a daily basis. We have also provided templates and instructions for the SA groups to assist them in setting up a harvester through whatever golden host mechanism that they employ, using SSH. We then use SSH to get the aggregated data securely to our central CMRE processing server. In this case, the System Administrator is required to ensure that there is a trust relationship, that the file systems that we require exist, and that enough space for collection is available.

Processing

After receiving the configuration bundles from the harvesting systems, we extract the contents into directories – one for each host. If the collector data we receive is a full collection, marked with a different filename by the collector, we first remove all the data in that host directory, and then we un-tar the file into the host directory. If it is an incremental collection, then we un-tar the file into the host directory, overwriting any changed files. After the files are extracted, we use CVS to check the files in, thereby keeping a historical record of the data.

We collect between 0.5 and 30 MB of raw data from each client (depending on the type and contents of each system). After the data is uploaded to our central server, we process some of that data and upload it into a database. This processing allows us to easily query and report on information, as well as aggregate data into group and company wide reports. The following is a sample of the data that we upload to the database:

- Date – one of the collectors executed is the 'date' command which we upload to determine the last time that the collection was successful.
- Operating System
- OS Version
- Hardware
- System Build Version – our internal identifier for a particular OS release that we make available
- Memory
- Number and Speed of Processors

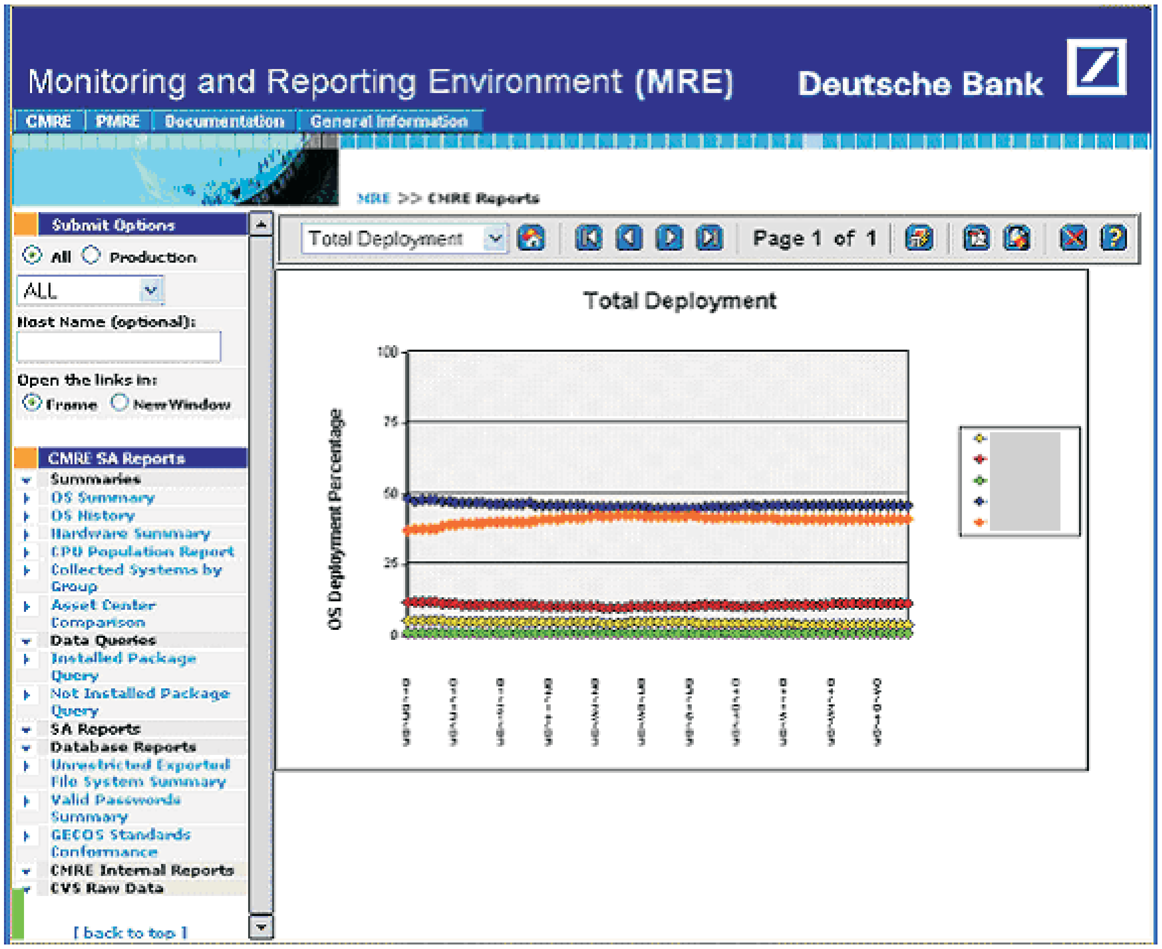


Figure 4: Sample of OS History Chart.

- Disk Information – Number, types, capacity, used capacity
- Passwd Information – contents of /etc/passwd
- Installed Packages and Patches
- Network Interfaces
- Services – both inetd.conf contents as well as netstat output

Processing is done via a series of custom Perl modules. Because the information we receive is typically from commands that deliver information in a human-readable form, we have to provide custom parsers for each type of output. The parsed data is uploaded into a database into tables that are designed for that particular data.

Reporting

Our initial implementation used custom reporting scripts. For each report we wished to present, we coded a CGI script in Perl to extract the data from the database or file system, and then present the report to the user. We wanted to create a more efficient and flexible process, therefore we examined a number of data mining products and selected one to replace a significant number of reports. This brought the development time of each report down to minutes from hours or days.

We provide a number of different types of reports. Some are simple aggregated data reports, such as the operating system summarization shown in Figure 2.

Note that all data in charts and graphs included in this paper are shown with either generated data created for demonstration purposes only or are redacted and presented to demonstrate format only.

We also provide charts of the same information (shown in Figure 2).

In addition we also collect historical information, so that we can see how we are doing over time, for instance how our migration to new operating systems is progressing (Figure 4).

Additional reports include queries such as all machines of a particular operating system that does or does not have a patch or package installed. This has come in useful in a number of occasions, for instance to track our rollout of Kerberos, to detect hosts that were running a susceptible version of sendmail, and to provide a list of systems that required SSH updates.

PMRE

In addition to collecting configuration information, we are also collecting performance information. We are currently using the public domain Orca program (http://www.orcaware.com/orca/) to collect data on all our systems. In addition to the standard (somewhat

Net Load Compared to Available Capacity

Hostname	Available Capacity	Percentage Utilized	Hostname	Net Load
foo1.uk.db.com	0.806	14.29%	foo2.us.db.com	0.660
foo2.us.db.com	0.740	47.14%	foo7.srv.uk.deuba.com	0.592
foo3.na.deuba.com	0.688	38.57%	foo4-adm.na.deuba.com	0.512
foo4-adm.na.deuba.com	0.608	45.71%	foo3.na.deuba.com	0.432
foo5.aus.deuba.com	0.564	40.00%	foo10.us.db.com	0.408
foo6.aus.deuba.com	0.538	42.86%	foo5.aus.deuba.com	0.403
foo7.srv.uk.deuba.com	0.528	52.86%	foo6.aus.deuba.com	0.376
foo8.srv.uk.deuba.com	0.464	34.29%	foo9.srv.uk.deuba.com	0.262
foo9.srv.uk.deuba.com	0.444	37.14%	foo8.srv.uk.deuba.com	0.242
foo10.us.db.com	0.432	48.57%	foo16.aus.deuba.com	0.215
foo11.srv.uk.deuba.com	0.403	14.29%	foo17.aus.deuba.com	0.202
foo12.uk.db.com	0.376	20.00%	foo14.uk.db.com	0.181
foo13.srv.uk.deuba.com	0.302	27.14%	foo23.uk.db.com	0.148
foo14.uk.db.com	0.289	38.57%	foo1.uk.db.com	0.134
foo15.uk.db.com	0.270	14.29%	foo13.srv.uk.deuba.com	0.112
foo16.aus.deuba.com	0.269	42.86%	foo22.apcc.ap	0.112
foo17.aus.deuba.com	0.255	45.71%	foo12.uk.db.com	0.094
foo18.uk.db.com	0.232	17.14%	foo11.srv.uk.deuba.com	0.067
foo19.sg.db.com	0.228	18.57%	foo19.sg.db.com	0.052
foo20.srv.uk.deuba.com	0.178	14.29%	foo18.uk.db.com	0.048
foo21.uk.db.com	0.175	15.71%	foo15.uk.db.com	0.045
foo22.apcc.ap	0.168	40.00%	foo21.uk.db.com	0.033
foo23.uk.db.com	0.132	52.86%	foo20.srv.uk.deuba.com	0.030
foo24.srv.uk.deuba.com	0.091	22.86%	foo24.srv.uk.deuba.com	0.027

Represents Demonstration Data Only

Table 4: Load can be shifted from busy systems to idle ones.

modified) graphs provided by Orca to allow people to see the performance of our system, we also do additional parsing of the data to determine overall weighted utilization numbers on our machines. This allows us to determine what our most and least used hosts are, globally. We can also drill down on individual machines. Currently we base our utilization number solely on CPU utilization, which usually proves to be an accurate representation. Clearly, in some cases we will drill down on a host and discover that while it is barely using any CPU, it is memory or I/O constrained, but this is the exception.

Once again, in addition to summarizations we provide host detail as well (Figure 5).

Problems Solved

CMRE has saved Deutsche Bank money in system retirements and cost avoidance, and it has reduced the number of people performing repetitive, manual tasks. Below are just some of the areas where having the information provided in CMRE has saved the bank money, time or both.

System Recovery

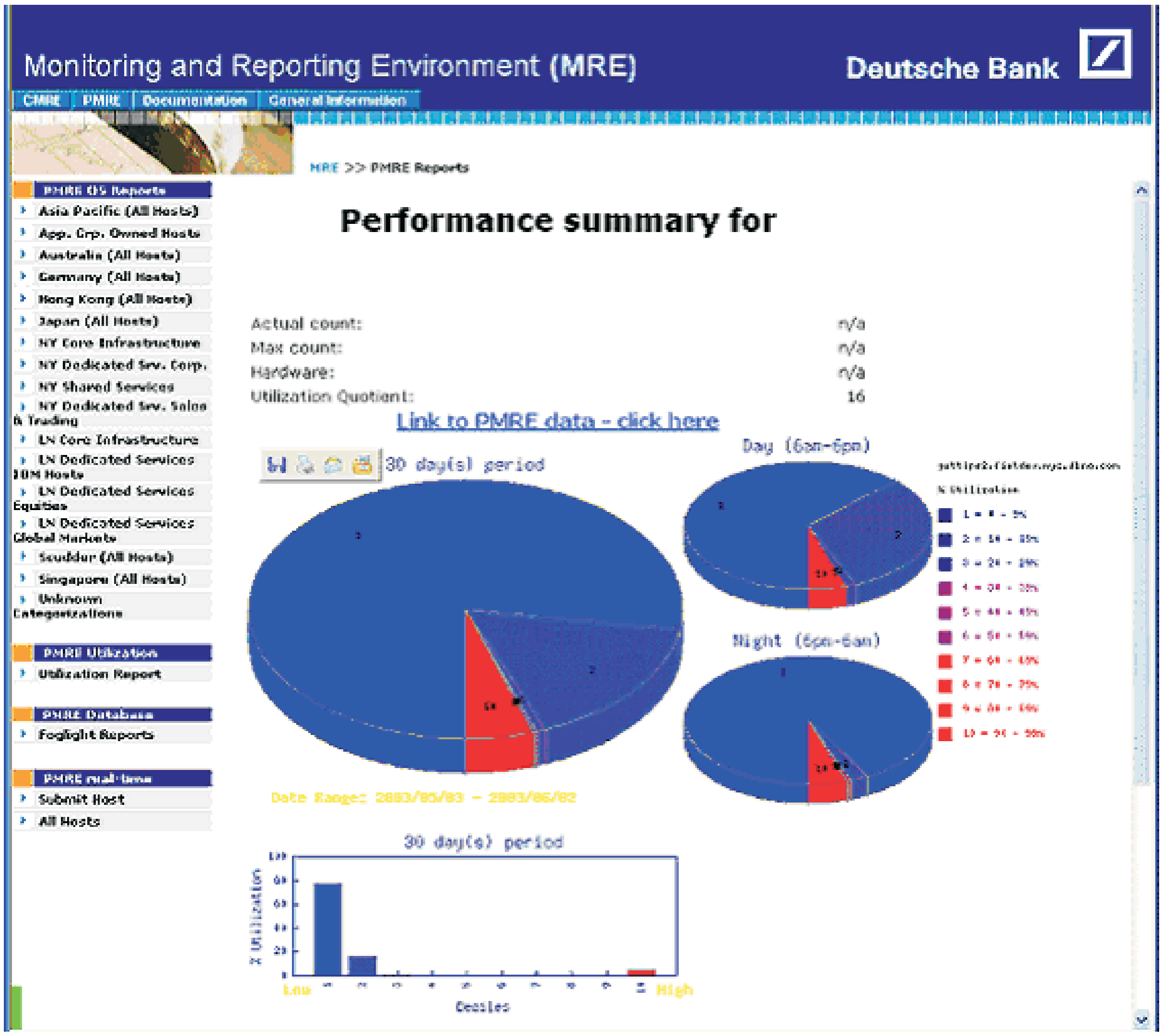
While lost system data can be recovered, a significant amount of specific machine configuration information is required in order to reconstruct the system fully after it has been lost (such as size and partitioning of disk, how much memory was installed, etc.). CMRE makes all this information easily available and accessible, so that rebuilding or replicating systems is a far simpler task.

More Efficient Reporting

Having security and audit vulnerabilities of our systems available in a single location, significantly reduces the time necessary for SA groups to track down and remediate problems.

System Retirements

Based on information presented in the utilization reports shown above, we were able to identify underutilized machines. As we were doing some major data center moves, we were able to retire a number of these



Performance summary for

Actual count:	n/a
Max count:	n/a
Hardware:	n/a
Utilization Quotient:	16

Link to PMRE data - click here

Date Range: 2003/05/03 - 2003/06/02

Figure 5: Host performance summary.

machines, thereby removing them from our inventory and avoiding costs of moving them.

Utilizing CMRE and PMRE, we were able to put together matrixes such as the one below to determine which systems could be retired. In the example below, the Available Capacity is a normalized number based on the speed and number of the processors (gathered with CMRE) minus the load currently on the system. The Net Load of a system is calculated based on a weighted CPU utilization formula over a 30 day period.

Table 4 shows instances where load could be shifted from a low utilized system (bottom to top of right column) onto another machine with excess capacity (top to bottom of left column) and then retire the resources. Additional reports provide information about depreciation of the machine (retrieved from our Asset database) and the cost of housing the machine in our data center.

Cost Avoidance

Even within large groups, systems are often purchased individually, with large amounts of space left for growth. The result is that systems may be significantly under populated. Figure 6 is report that a business area requested so that they could determine whether to purchase new machines (requiring additional data center space, network connections, etc.). The report showed that we had quite a few machines that were less than fully populated, and we could avoid the significant costs of purchasing new machines simply by purchasing additional CPU cards.

The Future of MRE

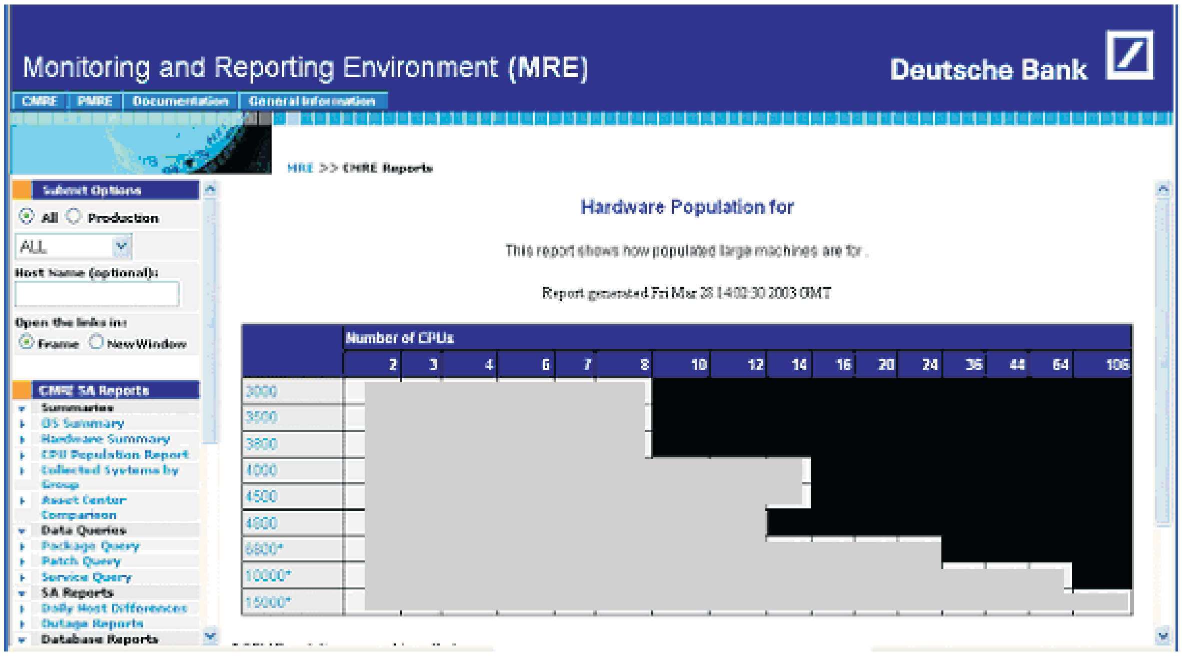
CMRE and PMRE are the first components of an overall management framework that we are calling MRE (Management and Reporting Environment). MRE is envisioned initially as a portal where all information about individual systems will be consolidated or viewable. Information available on MRE would include configuration, performance, asset information, security and event monitoring statistics, etc. For instance, someone investigating a performance issue could go to the single MRE portal first to look at the performance data returned, then to examine the configuration information that explainswhat may be causing the performance issues. Currently, MRE is a read-only system, because we are in the information gathering process. As we grow the various components of MRE and are able to tackle some of the challenges that lie ahead, we will be far better positioned for proactive systems management.

Availability

Because the system is made up of both commercial and non-commercial components, it cannot be made generally available as a package. However, the collector scripts as well as discussions and suggestions about the report formats are available by contacting the authors.

Author Information

Xev Gittler is the Regional Unix Architect for the Americas for Deutsche Bank. He has worked for nearly 20 years as a system administrator and system architect at a number of financial institutions, universities, and R & D labs. He is a past Vice President of SAGE as well as a founder and original president of NYSA, the NY Systems Administrators group. Xev would like to thank his wife, Rebecca Schore, for everything, but especially for helping turn this paper



Hardware Population for

This report shows how populated large machines are for .

Report generated Fri Mar 28 14:02:30 2003 GMT

	Number of CPUs															
	2	3	4	6	7	8	10	12	14	16	20	24	36	44	64	106
3000	▨	▨	▨	▨	▨	▨	■	■	■	■	■	■	■	■	■	■
3500	▨	▨	▨	▨	▨	▨	■	■	■	■	■	■	■	■	■	■
3800	▨	▨	▨	▨	▨	▨	■	■	■	■	■	■	■	■	■	■
4000	▨	▨	▨	▨	▨	▨	▨	▨	▨	■	■	■	■	■	■	■
4500	▨	▨	▨	▨	▨	▨	▨	▨	▨	■	■	■	■	■	■	■
4800	▨	▨	▨	▨	▨	▨	▨	▨	■	■	■	■	■	■	■	■
6800*	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	■	■	■	■
10000*	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	■
15000*	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨

Figure 6: Sample hardware population report.

from ideas into something that could be put down on paper. Xev can be reached via email at Xev.Gittler@db.com.

Ken Beer graduated in 1992 from the University of Maryland with a degree in Philosophy. For the past 10 years he has been a Unix systems administrator in the financial sector in New York City. He is currently a VP of Unix Engineering at Deutsche Bank where he works on improving system maintainability. Ken lives in Brooklyn, NY where he spends any free time he can with his wife Cheri and son Satchel. Ken would like to thank his dad for buying him a subscription to Byte magazine in 1979. Ken can be reached via email at Ken.Beer@db.com.

References

Brooks, Frederick Phillips, *The Mythical Man-Month: Essays on Software Engineering*, Pearson Addison Wesley, 1988.

Gall, John, *Systemantics: The Underground Text of Systems Lore*, Sytemantics Press, 1986.

Perrow, Charles, *Normal Accidents: Living with High-Risk Technologies*, Princeton University Press, 1999.

Dorner, Dietrich, *The Logic of Failure*. Perseus Publishing, 1996.

Merton, Thomas, *The Unanticipated Consequences of Purposive Social Action*, 1936.

Hunt, Andrew, David Thomas, and Ward Cunningman *The Pragmatic Programmer From Journeyman to Master*, Addison-Wesley, 1999.

Burgess, Mark, "Computer Immunology," *LISA Conference Proceedings*, 1998.

Burgess, Mark, "A Site Configuration Engine," *USENIX Computing systems*, Vol. 8, Num. 3, 1995.

Traugott, Steve, "Bootstrapping an Infrastructure," *LISA Conference Proceedings*, 1998.

"A Simple Network Management Protocol (SNMP)," *RFC 1157*.

Oetiker, Tobias, "Template Tree II: The Post-Installation Setup Tool," *LISA Conference Proceedings*, 2001.