Distributed Data Collection through Remote Probing in Windows Environments

Patrício Domingues
ESTG – Leiria – Portugal
patricio@estg.ipleiria.pt

Paulo Marques
Univ. Coimbra – Portugal
pmarques@dei.uc.pt

Luís Silva
Univ. Coimbra – Portugal
luis@dei.uc.pt

Abstract

Distributed Data Collector (DDC) is a framework to ease and automate repetitive executions of console applications (probes) over a set of LAN networked Windows personal computers. The framework allows for the remote execution of probes, providing support for collecting the execution output of probes (standard output and standard error). Additionally, right after a probe execution, the output can be parsed and processed by user defined code (post collecting code) that can act accordingly to user's needs. The framework can be useful to perform repetitive large-scale monitoring and administrative tasks over machines with transient availability, that is, machines that present no guarantees of being available at a given time.

A major strength of DDC lies in the fact that it does not require installation of software in remote nodes, avoiding administrative burdens that remote daemons and alike normally provoke.

Besides presenting the data collection framework, the paper discusses the results of a 30-day monitoring experiment conducted with DDC. The experiment collected resource usage metrics over 169 Windows machines from classroom laboratories of an academic institution.

1. Introduction

Controlling and monitoring a large number of Windows personal computers can be a challenging and daunting task. Frequently, administrators and other power users would like to perform controlling tasks on a bunch of machines they are in charge of. These types of tasks might involve the remote execution of probes on a set of machines and collecting each execution result to a central repository. Additionally, collected results might require parsing and processing, and if needed, appropriate measures such as sending status report or alerting system administrators can be taken. Examples of such tasks include checking disks for suspicious filenames, assessing machine performances and its evolution over time through execution of specific tailored benchmarks, measuring system load on machines in order to quantify per machine usage and assess possible CPU and other resources harvesting opportunities, and so on. However if these tasks are important and can be time saving, for instance, through the analysis of *Self-Monitoring Analysis and Reporting Technology (SMART)* [1] parameters disk failures can be anticipated, they usually have low priorities in administrators' busy todo lists. Another problem of performing such task lies in the fact that in a set of machines, at any given time and depending of the particularities of the environment (number of machines, user habits, etc.), a percentage of these machines will be switched off. Thus, to cover as many machines as possible and to obtain significant results, a persistent execution environment able to periodically repeat execution attempts is needed. Therefore, a framework able to perform active data collection by way of remote execution of probes in networked Windows personal computers, supporting transient availability of machines and allowing results analysis by user-defined code is a valuable tool for system administrators. DDC aims to be such a tool.

The remainder of this paper is organized as follows. Section 2 exposes our motivation. Section 3 describes DDC, its architecture and its internal organization. Section 4 presents and briefly discusses the results of a 30-day monitoring experiment conducted with DDC over 169 classroom machines. Section 5 describes related work, while in section 6 future work is discussed. Finally, section 7 concludes the paper.

2. Motivation

DDC development was originally motivated by our own needs to collect continuous resource metrics about personal computers usage at our academic institution. We were interested, amongst other things, in assessing resource usage (CPU, memory, disk and network) of several classroom laboratories that run Windows 2000 professional on a total of 169 machines. One important characteristic of this pool of machines lies in its transient availability, with no guarantee existing that a machine will be powered on at a given time.

Besides the purpose to collect machines usage data, we felt the need for a framework that would allow running a regular console application across a set of machines. For example, to regularly assess system performance, a benchmark could periodically be run with its output parsed and relevant performance data collected in order to early spot performance drops. Or, to detect near future possible hard disk failures, a utility that would return SMART values could be run every night across the whole set of machines, with suspicious disks being flagged as possible near failures. These situations are just examples of what a remote execution framework could permit to achieve in a set of networked machines. Therefore, we needed a distributed collection platform that obeyed the following restrictions:

- Besides the requirement of being a console application, no further restriction should be placed on probes.
- Solutions should be modular and easily extensible, allowing for easy integration of probes and associated post collection filters in order to fulfill further needs and opportunities for data collection.
- No software should be installed at remote machines in order to avoid administrative and technical burdens. However, it is acceptable to use a single dedicated machine (coordinator) for running the data collection system as long as no special hardware is required.
- The system should be as autonomous and adaptive as possible in order to minimize administrator interventions. For instance, the system should cope with the transient nature of machines availability, since, at our institution, no requirement exists for machines to be permanently switched on. Ideally, besides producing normal progress reports, the system should alert administrators only when abnormal events that require human intervention occur.
- Due to budget restriction and time limitation, solutions should be based on open source or at least freeware software.
- The data collector system should support Windows NT and derivatives (2000 and XP).
- The system should be able to execute probes at remote nodes in one of two modes: single-shot or periodic. The former serves for probes that should be executed only once (e.g. a benchmark) while the latter, as the name implies, periodically executes the probe (e.g. resource monitoring). Additionally, single shot mode has to deal with volatile machine availability, only terminating when execution of probe(s) has occurred in all specified machines.
- To keep a low intrusive profile, the framework should permit remote execution of processes at a low level of priority.

Since, to the best of our knowledge, no system complied with our requirements we developed DDC.

3. Overview

3.1. Experiment and iterations

DDC is based on a centralized architecture, with a central and the only devoted machine (coordinator) orchestrating the whole execution running the main module of DDC. All the executions are performed in the context of an experiment. A DDC experiment is composed by possibly several binary probes and their associated post collecting codes plus the set of target machines where probes execution should occur. In DDC, a probe is simply a win32 console application which emits its results via standard output (stdout) and standard error (stderr) channels.

DDC conducts an experiment in successive iterations. An iteration consists of the execution attempt of every defined probes over the whole set of target machines. DDC uniquely identifies an iteration with the GMT time in Unix's epoch format measured when iteration starts.

An iteration is executed as follows. For every machine belonging to the execution pool, and accordingly to the configuration, user defined probes and their associated post collect codes are sequentially executed. When execution of all probes have been attempted at a given machine (an execution might not be successful, failing for several reasons – for instance the remote machine might be down), DDC shift execution to the next machine of the pool. Figure 1 depicts the sequential steps of a probe execution. After a successful ping test to assess target machine availability (step a), remote execution is attempted (step b) and remote output is returned (step c). Finally, if configured, post collecting code is run on the probe's results (step d) with filtered output possibly archived on disk.

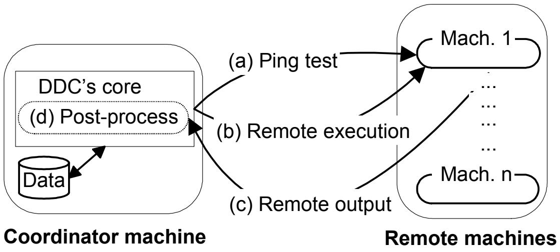

Figure 1

At the coordinator machine, the results of the execution of a probe are cumulatively redirected to text files (one for stdout, another for stderr) whose names are based on probe's own name. These files are stored under a directory hierarchy (one directory per machine) with the root directory named after the experiment name. This default logging behavior of the outputs can be replaced by user-defined code that gets executed on the coordinator machine right after remote execution occurs. This post-collecting code, which is specific to a probe, receives as input the results of the probe's execution and can implement actions that user defines as deemed appropriate for the given context. For example, after parsing and processing the execution output, post-collecting code can decide to report a particular event via e-mail. Besides the output of the execution, post-collecting code also inherits the execution context that contains information respecting the machine where the probe was executed, the execution status (exit code from the probe) as well as the wall-clock execution time elapsed at the remote machine.

In DDC, a definable experiment parameter is the interval time that should separate the start of two consecutive iterations and that defines the data collection frequency. So, after an iteration has completed, DDC pauses the execution until its time to start next iteration. If this specified time interval cannot be respected (for instance, execution of last iteration took longer than the defined time interval), DDC waits for a minimum time gap before starting the next iteration. This prevents execution loops that "spin" in an uncontrolled manner.

For every probe of an experiment, DDC maintains a trace file. At the end of an iteration a text line summarizing the outcome of the iteration is appended to the file. This line starts with the iteration's timestamp identifier and includes, amongst other items, a comma separated list with the machine names where execution was successful, a similar list for failed executions (executions aborted due to timeouts) and a third list that holds names of the machines that were unavailable when execution was attempted. This line also contains the wall-clock time needed to complete the iteration. A much trimmed example of a trace line is shown in listing 1. From this line it can be extracted that the iteration begun at timestamp 1081948390 with 120 successful executions (machine *m01* and others), no failed executions and 49 unavailable machines (machine *m12* and others) yielding 71.0% successful executions. The iteration took nearly 382 seconds.

```
1081948390|120|0|49|71.0|381.9|m1,||m12,...|
```

Listing 1

The trace file of a probe can be used to conduct an offline temporal analysis of probe execution. This file can also be used as a log, and in fact, the periodic mail reports that are sent by a DDC experiment includes the trace file (due to its high redundancy, mostly machine names, the file is highly compressible) as a source of information about the evolution of the experiment.

3.2. Remote execution

The remote execution mechanism of DDC is based upon the wealthy set of freeware tools from *sysinternals* [2], namely the versatile *psexec*. *Psexec* is a utility that permits the remote execution of an application given the proper access privileges. *Psexec* is a very flexible tool configurable through appropriate command line switches.

In DDC, *psexec* is used as follows: an appropriate command line that includes all the needed *psexec* switches and parameters, as well as the probe executable name with its own command-line arguments (if any) is formatted and executed in the context of a separate thread within DDC core. The need of a separate thread for the remote execution arises from the possibility of deadlock at the remote machine that would otherwise stall DDC execution. Therefore, after a given time interval (configurable for each probe) if the execution of *psexec* has not yet terminated a timeout is triggered with the execution being aborted by way of cancellation of the execution thread. Under these circumstances the remote machine is flagged as having failed the probe execution with a new attempt to be tried in the next iteration.

Since a target machine can be unavailable at a given time (powered off, unplugged from the network, etc.) before attempting the execution of a probe, the target machine connectivity is tested with ICMP pings. If no answer to ping is received the remote machine is assumed to be unavailable and thus no remote execution is attempted in the current iteration. The advantage of using ping over immediately attempting execution is that ping timeout can be controlled and thus set to a lower value (for instance, hundredth of milliseconds) than the time length it would take *psexec* to detect remote machine unavailability (in the range of seconds). This way detection of an unavailable machine is much faster.

3.3. Post-collecting code

An important element for DDC flexibility lies in its ability to execute user defined code right after the execution of a probe allowing the processing of output channels (stdout and stderr). For that purpose, post-collecting code needs to be written in the form of a python class that extends *DDC_cmd* class, implementing the method *ParseResult()*. This method is run by the main core of DDC at the coordinator's machine right after remote probe execution has terminated. It receives as parameter, besides probe's stdout and stderr contents, an object representing the remote machine that actually executed the probe, the execution exit code of the probe, the iteration identifier as well as other context data (e.g. storage directory path at coordinator's where output files of current execution are). Even if post-collecting code is free to act on probe's results, it is limited by a time frame, since the next execution only starts after post-collecting code has concluded.

4. Monitoring experiment

In order to assess the ability of DDC to gather resource usage metrics we conducted a 30-day DDC monitoring experiment over 169 windows machines of 11 classroom laboratories. Machines ranged from Pentium III@650 MHz (128 MB memory, 14.5 GB disk) to Pentium 4@2400 MHz (512 MB memory, 74.5 GB disk), being connected via 100 Mbps Fast Ethernet. Combined together, the resources of the machines are impressive: 6.58 TB total disk space and 52.46 GB of RAM. Besides serving classes, all aforementioned machines are used by students to perform their practical assignments and homework, as well as for personal use (e-mail, etc.). To avoid change of behaviors that could false results, regular users were not informed about the monitoring experiment.

For the purpose of this monitoring experiment a probe named *W32Probe* was developed. *W32probe* outputs a wealthy set of information about the Windows system where it is executed. The probe captures static metrics such as CPU type, operative system information (name, service pack, version, installation date), amount of existing RAM, characteristics of hard disks, as well as network interface(s) properties. Besides static elements, *W32Probe* also collects dynamic metrics like machine uptime, CPU idleness percentage since boot time, physical and virtual memory load, free disk space, SMART parameters such as start/stop cycles and power on hours count, as well as network interface(s) sent and received bytes and rates. The probe also exposes several metrics about current interactive login sessions (if any), namely user name, domain name and login session's uptime. The sampled metrics are returned in text format via the standard output channel.

The post-collected code associated to *W32probe* logs the captured samples in a single text format for further analysis. Due to its nature, static metrics are only saved once, right after the first successful execution, while dynamic metrics are recorded after every execution. Finally, after DDC experiment has terminated all saved data are processed offline and inserted in a relational database. The offline processing is driven by *W32probe*'s trace file produced during the experiment.

4.1. Results

DDC was set to run an iteration every 15 minutes to sample the machines. A total of 235298 samples were collected over the 2867 iterations run. The average time for carrying out a whole iteration was 386.99 seconds. The experiment was interrupted twice because of global power failures, and a third time due to a local power failure that disrupted network connectivity of the coordinator machine. It is important to note that no shutdown policy of the machines is enforced, that is, after usage, users are free to left machines powered on, since other classes or users might follow.

Main results of the experiment are summarized in Table 1. The column "Without users" shows results captured when no interactive user-session existed, while the column "With users" depicts samples gathered at user-occupied machines. The final column "Both" combines all results.

Table 1: Main results.

	Without users	With users	Both
Samples	129036	106262	235298
Avg. uptime (%)	26.63	21.93	48.56
Avg. CPU idle (%)	98.82	95.92	97.51
Avg. RAM load (%)	53.45	63.35	57.92
Avg. SWAP load (%)	24.53	29.56	26.80
Avg. disk free (MB)	27.81	35.66	31.35
Avg. sent bytes (Bps)	313.35	2229.55	1178.72
Avg. recv. bytes (Bps)	322.57	7441.13	3537.36

Of the 484523 probe executions attempted, 235298 were successful corresponding to a 48.56% uptime. The high CPU idleness percentage (98.82% when no user is logged on, 95.92% when interactive login session exists) confirms similar studies carried out in Windows [3] and in Unix [4][5], strengthening attractiveness of academic laboratories for CPU scavenging.

As expected, average RAM memory load increases from 53.45% to 63.35% when an interactive login session exists. Interactive login sessions also increase the SWAP load from 24.53% to 29.56%. This came as no surprise since an interactive login session means that more applications are being used and consequently more memory is required. The significant amount of free RAM – nearly 47% when no login exists – confirms results of [6] and makes the evaluated computers attractive for idle memory exploiting systems such as network RAM and temporary network RAM disks schemes [7].

On average a machine has more than 30 GB of hard disk space free (average disk size is 39 GB). Several factors contribute to this high amount of free disk space. First, machines are reinstalled at the beginning of each semester and only hold the needed software for classes and student practical assignments. Second, an interactive regular user is not allowed to locally install software and has a disk quota ranging from 100 MB to 300 MB of temporary storage at the machine she is logged on (the actual size depends on the machine hard disk drive capacity). These two factors combined with the fast growing hard disk drive size yield the high amount of free disk space. Such high quantity of unused disk space can be explored, for instance, for distributed backups [18]. However, in these mechanisms data confidentiality must be rigorously assured, and even more difficult, confidence of users must be gained before they entrust their data to such systems. The fact that samples collected from machines with interactive session present an higher free disk space than samples gathered from free machines (35.66 MB versus 27.18 MB) might be motivated by the fact that newer machines with bigger disk drives (and consequently with more free space disk) are more used for interactive sessions than older and consequently smaller disk machines.

Respecting network usage, the existence of a user session increases more than twenty times the rate of received bytes in a machine, from 322.57 Bps to 7441.13 Bps. The same applies to outgoing traffic, but with lower rates (from 313.35 Bps to 2229.55 Bps). The network rates also demonstrate the client role of machines, with the incoming traffic rate roughly three times higher than the outgoing traffic rate.

5. Related work

Even if many open source or freeware monitoring tools exist for distributed systems like network of personal computers, few of these tools support the Windows platform, although Windows machines accounts for more than 90% of desktop machines. And, to our knowledge, none of the tools aimed at Windows platforms provided supports for remote data collection and post-collecting processing of collected data without the need of remote software installation.

Simple Network Management Protocol (SNMP) [8] is a standard protocol used to exchange information, allowing remote monitoring and management of networked devices that support the protocols. Although SNMP could be used to retrieve monitoring metrics, SNMP does not allow the execution of arbitrary probes at remote nodes. So, while DDC permits the use of any console application as probe, data collected via SNMP are restricted to the agent capabilities. Another major drawback of SNMP comparatively to DDC lies in its deployment and maintenance administrative overhead. In fact, due to its client-server structure SNMP requires software agents to be installed and configured in all machines, something that DDC avoids completely. Also, SNMP acceptance among system administrators is hindered by its weak security fame [9].

The Remote Monitoring (RMON) [8, 10] protocol extends SNMP with new and more flexible Management Information Base (MIB) definitions. However, regarding our needs for remote data collection RMON is hindered by the same drawbacks and limitations of SNMP.

Windows Management Interface (WMI) is an implementation of Web Based Enterprise Management (WBEM) for win32 environments [11]. WMI technology offers a set of services and makes accessible a rich set of data regarding many aspects of win32 systems. An interesting feature of WMI lies in its ability to execute processes at remote machines [12] although at the expenses of some code complexity and resource consumption. Thus, WMI could be used as a possible replacement for *psexec* remote execution mechanism. Comparatively to WMI, DDC focuses toward acquisition, processing and storage of remotely collected data, while WMI is oriented toward monitoring and administration tasks.

Condor [13] is a well-known and mature framework for opportunistic execution of applications ("tasks" in Condor language) in otherwise idle resources of pools of networked computers. Condor supports several operative systems, including Win32 platforms. The framework acts as a batch system, scheduling user's tasks to available computing nodes. Execution at a remote node occurs when the task requirements are matched with node availability. However, using Condor system for remote execution of probes (submitted as condor's tasks) does not fit our purposes due to several reasons. First, even if Condor can be instructed at submit time to run a task in a given machine, due to its opportunistic scheduling strategy it cannot guarantee when the task will actually be run. Second, under Windows, Condor's tasks are executed under least execution privileges, a recommended security measure in distributed systems, but that renders impossible the execution of probes that require administrative credentials. Finally, deployment of Condor requires the installation at the remote nodes of Condor's client software, a demand that violates our requirements for zero-software installation.

Grid desktop computing framework like BOINC [14] can also be used for executing user-defined probes in network of personal computers although this is a deviation from their main purpose of supporting so called "embarrassingly" distributed @Home projects such as SETI@Home and Folding@Home [15]. Besides the need of setting the BOINC infrastructure, which requires the installation of client-side software on every target machines and setting a machine for the server-side, probes must be programmed accordingly to the platform rules and limitations, using the supported languages. Since the platform is geared toward the execution of long-running applications, some of the requirements like checkpoint support are inappropriate for short-run probes. DDC has a much more flexible approach since it can use any existing console executable as probe, easily supports multiple probes, requiring only a minimal setup and no remote installation. A potential advantage of grid desktop computing frameworks over DDC arises from having their execution distributed over the participating computers. This makes possible a simultaneous execution of a probe, while DDC only supports sequential execution. In practice the scheduling policy of grid desktop computing tools that schedule execution only when host resources are near idleness makes a synchronized execution difficult to achieve. A more real advantage of BOINC usage over DDC lies in its support across Internet, while DDC is restricted to LAN environments.

Finally, remote desktop tools are a possible way of remotely executing probes. Several of such tools exist for the Win32 platform. For instance, utilities such as VNC [16] and variants, telnet server and Microsoft's remote desktop permit remote access to a windows machine. However, these tools are aimed to provide graphical remote login interactivity and thus are not suitable for automatic execution of programs in a way such as DDC provides.

6. Future work

An important area for improvement in DDC relates to security. Although very versatile, *psexec* does not have, according to its authors, a strong security structure. A possible replacement for *psexec* could be the open-source *XCmd* [17] which is a similar tool for remote execution with the added bonus of source code availability.

Some probes might require exclusive access to a machine, that is, no interactive login session can exist at the given machine. This exclusive access might be required for assuring results precision (for instance, in a benchmark run), non-intrusiveness for interactive user of the remote machine or both. To accommodate this requirement, the detection of user login prior to execution attempt would be needed. However, this method does not prevent nor detect situations when interactive login occurs during the execution of a possibly lengthy probe. For such cases and if the probe execution cannot tolerate interferences like interactive login, then the probe itself should detect such situations, for instance setting WMI for event notification when a login is detected.

Currently, a limitation of DDC respects the execution of post-collecting code that is carried out sequentially in DDC main execution path, effectively delaying next remote execution until the post-collecting code has completed. This effectively limits actions that can be carried out in post-collecting phase, since a fast post-collecting execution is required. A possible improvement would be to create a separate thread to execute the post-collecting code, allowing DDC to chain remote executions without having to wait for post-collecting executions.

Finally, if the need arises, DDC can be ported to UNIX environments. Since DDC is written in python, portability should not be a major issue. Also, the psexec remote execution mechanism could easily be replaced by SSH with the added bonus of improved security.

7. Conclusions

This paper presented a simple yet effective framework that eases repetitive executions of multiple probes over a set of LAN networked Windows machines. DDC can be especially useful for administrative purposes that can benefit from remote data collection, namely monitoring operations.

We used DDC and a specially developed probe to evaluate computing resource usage over 30-day on 169 classroom Windows personal computers. All monitoring operations were conducted from the coordinator machine, with no software being installed at the monitored machines. In fact, we never needed to log on interactively on any of the monitored machines, with all operations being conducted remotely.

As expected in such environments, resources idleness is quite high, not only CPU but also disk and in a lesser level, main memory. This confirms the opportunity for resource scavenging on computing environments like laboratory classrooms. Indeed, idle resources exploitation in such environments is strengthened by the fact that machines have no real personal "owner", being centrally and thus more closely managed avoiding social issues that normally arise in personal computers resource sharing schemes.

The source code and further documentation about the framework can be obtained from http://ww2.estg.ipleiria.pt/~patricio/DDC/.

Acknowledgements

This work was partially supported by *PRODEP III – Acção 5.3* and by the Portuguese Foundation for Science and Technology under the POSI programme and the FEDER programme of the European Union, through the R&D Unit 326/94 (CISUC).

References

[1] B. Allen, "Monitoring Hard Disks with SMART," in Linux Journal, vol. Nº117, January 2004.

[2] M. Russinovich and B. Cogswell, "Sysinternals - PsTools (http://www.sysinternals.com)," 2004.

[3] P. Domingues, L. Silva, and J. G. Silva, "DRMonitor - A Distributed Resource Monitoring System," presented at 11th Euromicro Parallel, Distributed and Network-Based Processing, Genova, Italy, 2003.

[4] R. Arpaci, A. Dusseau, A. Vahdat, L. Liu, T. Anderson, and D. Patterson, "The interaction of parallel and sequential workloads on a network of workstations," presented at ACM SIGMETRICS joint international conference on measurement and modeling of computer systems, Ottawa, Ontario, Canada, 1995.

[5] T. E. Anderson, D. E. Culler, D. A. Patterson, and N. team, "A Case for NOW (Network of Workstations)," in IEEE Micro, February 1995, pp. 54-64.

[6] A. Acharya and S. Setia, "Availability and utility of idle memory in workstation clusters," presented at ACM SIGMETRICS international conference on Measurement and modeling of computer systems, Atlanta, Georgia, United States, 1999.

[7] M. Flouris and E. Markatos, "Network RAM," in High Performance Cluster Computing, vol. 1 (chapter 16), R. Buyya, Ed., 1999, pp. 383-508.

[8] W. Stallings, SNMP, SNMPv2, SNMPv3, and RMON 1 and 2, 3rd ed: Addison-Wesley Pub. Co., 1999.

[9] CERT, "CERT Advisory CA-2002-03: Multiple Vulnerabilities in Many Implementations of the Simple Network Management Protocol - http://www.cert.org/advisories/CA-2002-03.html," CERT, 2003.

[10] S. Waldbusser, "RFC2021 - Remote Network Monitoring Management Information Base," IETF January 1997.

[11] M. Lavy and A. Meggitt, Windows Management Instrumentation (WMI): New Riders, 2001.

[12] T. Huckaby, "An Introduction to WMI," in Windows & .Net, October 2000.

[13] M. Litzkow, M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations," presented at 8th International Conference of Distributed Computing Systems, San José, California, 1988.

[14] BOINC, "Berkeley Open Infrastructure for Network Computing - http://boinc.berkeley.edu," 2004.

[15] K. Pearson, "Internet-based Distributed Computing Projects - http://www.aspenleaf.com/distributed/," 2004.

[16] VNC, "RealVNC homepage project, http://www.realvnc.com/," 2002.

[17] Z. Csizmadia, "XCmd - Execute Applications on Remote Systems - http://www.codeguru.com/Cpp/I-N/network/remoteinvocation/article.php/c5433/," 2001.

[18] L. Cox and B. Noble, "Pastiche: Making backup cheap and easy," presented at Fifth USENIX Symposium on Operating Systems Design and Implementation, 2002.