I T H E A

International Journal

INFORMATION THEORIES & APPLICATIONS

2009 Volume 16 Number 2

International Journal
INFORMATION THEORIES & APPLICATIONS
Volume 16 / 2009, Number 2

IJ ITA is official publisher of the scientific papers of the members of the ITHEA® International Scientific Society

IJ ITA welcomes scientific papers connected with any information theory or its application.

IJ ITA rules for preparing the manuscripts are compulsory.
The **rules for the papers** for IJ ITA as well as the **subscription fees** are given on *www.ithea.org* .
The camera-ready copy of the paper should be received by http://ij.ithea.org.
Responsibility for papers published in IJ ITA belongs to authors.

General Sponsor of IJ ITA is the **Consortium FOI Bulgaria** (www.foibg.com).

International Journal "INFORMATION THEORIES & APPLICATIONS" Vol.16, Number 2, 2009

Printed in Bulgaria

Edited by the **Institute of Information Theories and Applications FOI ITHEA®**, Bulgaria,
in collaboration with the V.M.Glushkov Institute of Cybernetics of NAS, Ukraine,
and the Institute of Mathematics and Informatics, BAS, Bulgaria.

Publisher: **ITHEA®**
Sofia, 1000, P.O.B. 775, Bulgaria. www.ithea.org, e-mail: info@foibg.com

ISSN 1310-0513 (printed) **ISSN 1313-0463 (online)** **ISSN 1313-0498 (CD/DVD)**

SOME APPROACHES FOR SOFTWARE SYSTEMS ANALYSES AND ITS UPGRADE PREDICTION

Igor Karelin

Abstract: This paper proposes and discusses a new type of tool for Linux based software systems analysis, testing and optimization as well as the new approach which is based on this tool and will help to define the best moment for executing of effective and smart software upgrade in automatic mode on active and online system. The best moment means one when the software upgrade will cause the minimal services losses.

The presented tool is called CAP (Characterization of Availability and Performance) and provides engineers with an instrument for systems performance tuning, profiling and issues investigation as well as stress and load testing. The described system operates with over 150 Linux based system parameters to optimize over 50 performance characteristics. The paper discusses the CAP tool's architecture, multi-parametric analysis algorithms, and application areas. Furthermore, the paper presents possible future work to improve the tool and extend it to cover additional system parameters and characteristics.

The prediction of the software upgrade (SU) best moment mentioned above is supposed to be made on basis of performance and availability statistics got from the CAP tool.

Keywords: Software Systems Analyses, Linux Servers, Telecommunication System, Performance, Availability, Serviceability, Software Upgrade Prediction.

ACM Classification Keywords: C.4 Computer Systems Organization - Performance of Systems - Reliability, availability, and serviceability.

Introduction

The telecommunications market is one of the fastest growing industries where performance and availability demands are critical due to the nature of real-time communications tasks with requirement of serving thousands of subscribers simultaneously with defined quality of service. Before Y2000, telecommunications infrastructure providers were solving performance and availability problems by providing proprietary hardware and software solutions that were very expensive and in many cases posed a lock-in with specific vendors. In the current business environment, many players have come to the market with variety of cost-effective telecommunication technologies including packed data technologies such as VoIP, creating server-competitive conditions for traditional providers of wireless types of voice communications. To be effective in this new business environment, the vendors and carriers are looking for ways to decrease development and maintenance costs, and decrease time to market for their solutions.

Since 2000, we have witnessed the creation of several industry bodies and forums such as the Service Availability Forum, Communications Platforms Trade Association, Linux Foundation Carrier Grade Linux Initiative, PCI Industrial Computer Manufacturers Group, SCOPE Alliance, and many others. Those industry forums are working on defining common approaches and standards that are intended to address fundamental problems and make available a modular approach for telecommunication solutions, where systems are built using well defined

hardware specifications, standards, and Open Source APIs and libraries for their middleware and applications [Haddad, 2006] ("Technology Trends" and "The .org player" chapters).

The Linux operating system has become the de facto standard operating system for the majority of telecommunication systems. The Carrier Grade Linux initiative at the Linux Foundation addresses telecommunication system requirements, which include availability and performance [Lehrbaum, 2002].

The performance of Linux servers running in mission critical environments such as telecommunication networks becomes a critical attribute. Its importance is growing due to incorporated high availability approaches, especially for servers requiring five and six nines availability. With the growing number of requirements that Linux servers must meet in areas of performance, security, reliability and serviceability, it is becoming a difficult task to optimize all the architecture layers and parameters to meet the user needs. Linux servers also require different approaches to optimization to meet specific constraints of their operating environment, such as traffic type and intensity, types of calculations, memory, CPU and IO use.

There are many examples of Linux-based, carrier-grade platforms used for a variety of telecommunication server nodes. Depending on the place and functionality of the particular server node in the telecommunication network infrastructure, there can be different types of loads and different types of performance bottlenecks. Many articles and other materials are devoted to questions like "how will Linux-based systems handle performance critical tasks?" In spite of the availability of carrier-class solutions, the question is still important for systems serving a large amount of simultaneous requests, e.g. WEB Servers [Haddad, 2001] as Telecommunication specific systems.

Telecommunication systems such as wireless/mobile networks have complicated infrastructures implemented, where each particular subsystem solves its specific problem. Depending on the problem, the critical systems' resource could be different. For example, Dynamic Memory Allocation could become a bottleneck for Billing Gateway, Fraud Control Center (FCC), and Data Monitoring (DMO) [Haggander, Lundberg, 2000] even in SMP architecture environment. Another example is WLAN-to-WLAN handover in UMTS networks where TCP connection reestablishment involves multiple boxes including HLR, DHCP servers and Gateways, and takes significant time (10–20 sec.) which is absolutely unacceptable for VoIP applications [Korhonen, 2004]. A similar story occurred with TCP over CDMA2000 Networks, where a bottleneck was found in the buffer and queue sizes of a BSC box [Mattar et al., 2007]. The list of the examples can be endless.

If we consider how the above examples differ, we would find out that in most cases performance issues appear to be quite difficult to deal with, and usually require rework and redesign of the whole system, which may obviously be very expensive. The performance improvement by itself is quite a well known task that is being solved by the different approaches including the Clustering and the Distributed Dynamic Load Balancing (DDLB) methods [Nehra et al., 2007]; this can take into account load of each particular node (CPU) and links throughput. However, a new question may arise: "Well. We know the load will be even and dynamically re-distributed, but what is the maximum system performance we can expect?" Here we are talking not about performance problems, but about performance characterization of the system. In many cases, people working on the new system development and having performance requirements agree on using prototyping techniques. That is a straightforward but still difficult way, especially for telecommunication systems where the load varies by types, geographic location, time of the day, etc. Prototyping requires creation of an adequate but inexpensive model which is problematic in described conditions.

The author of this paper is working in telecommunication software development area and hence tends to mostly consider problems that he faces and solves in his day-to-day work. It was already said that performance issues and characterization are within the area of interest for a Linux-based system developer. Characterization of

performance is about inexpensive modeling of the specific solution with the purpose of predicting future system performance.

What are the other performance-related questions that may be interesting when working in telecommunications? It isn't just by chance I placed the word Performance close to Availability; both are essential characteristics of a modern telecommunication system. If we think for a moment about the methods of achieving of some standard level of availability (let's say the five- or six- nines that are currently common industry standards), we will see that it is all about redundancy, reservation, and recovery. Besides specific requirements to the hardware, those methods require significant software overhead functionality. That means that in addition to system primary functions, it should provide algorithms for monitoring failure events and providing appropriate recovery actions. These algorithms are obviously resource-consuming and therefore impact overall system performance, so another problem to consider is a reasonable tradeoff between availability and productivity.

Let us consider some more problems related to telecommunication systems performance and availability characterization as well as its overall analysis that are not as fundamental as those described above, but which are still important.

Performance profiling. The goal of performance profiling is to verify that performance requirements have been achieved. Response times, throughput, and other time-sensitive system characteristics should be measured and evaluated. The performance profiling is applicable for release-to-release testing.

Load testing. The goal of load testing is to determine and ensure that the system functions properly beyond the expected maximum workload. The load testing subjects the system to varying workloads to evaluate the performance behaviors and ability of the system to function properly under these different workloads. The load testing also could be applicable on design stage of a project to choose the best system architecture and ensure that requirements will be achieved under real/similar system workloads [Ong et al., 2005], [Shende et al., 2007].

Stress testing. The goal of stress testing is to find performance issues and errors due to low resources or competition for resources. Stress testing can also be used to identify the peak workload that the system can handle.

Performance issue investigation. Any type of performance testing in common with serious result analysis could be applicable here. Also in some cases, snapshot gathering of system characteristics and/or profiling could be very useful.

Performance Tuning. The goal of performance tuning is to find optimal OS and Platform/Application settings, process affinity, and schedule policy for load balancing with the target of having the best compromise between performance and availability. The multi-objective optimization algorithm can greatly reduce the quantity of tested input parameter combinations.

The problem of issues investigation deserves special attention. As it was already said modern telecommunication systems have extremely difficult architecture – they are distributed in relation to either hardware or software. The growth of a system leads to the increase of data level required for comprehensive systems state analysis and supervision and life cycle description. The majority of this information is represented as log files – text files consisting of time-stamped status and error messages detailing the operational history of a given piece of software.

As we can see system state and lifecycle are described by the huge amount of jumbled data produced and distributed by multiple software units. These data represent the behavior of each unit on a long time scale. The problem is that during system analysis (failure investigation for ex.) the search for information is time-consuming. Maintenance engineer should manually filter and sort data from all the log files to assess system state and its behavior. The situation can be more complicated in case of log files allocation in different network nodes (multi-

board systems). Fortunately the software tool called Log Merger has already been developed and presented by the author and helps the raw data to be automatically collected, analyzed, reordered and filtered according to engineer's needs in each particular case [Karelin et al., 2008].

This long introduction was intended to explain why the author started to look at performance and availability characterization problems, their applications to Linux-based, carrier-grade servers. Further along in this paper, we will consider existing approaches and tools and share some more approaches that were successfully used by the author in his work.

Overview of the Existing Methods for Characterization of Performance and Availability

A number of tools and different approaches for performance characterization exist and are available for Linux systems. These tools and approaches target different problems and use different techniques for extracting system data to be analyzed as well, and support different ways to represent the results of the analysis. For the simplest cases of investigating performance issues, the standard Linux tools can be used by anyone. For example, the GNU profiler gprof provides basic information about pieces of code that are consuming more time to be executed, and which subprograms are being run more frequently than others. Such information offers understanding where small improvements and enhancements can give significant benefits in performance. The corresponding tool kprof gives an opportunity to analyze graphical representation gprof outputs in form of call trees, e.g. comprehensive information about the system can be received from /proc (a reflection of the system in memory). Furthermore, for dealing with performance issues, a variety of standard debugging tools such as instrumentation profilers (oprofile which is a system-wide profiler), debuggers kdb and kgdb, allowing kernel debugging up to source code level as well as probes crash dumps and many others are described in details in popular Linux books [Best, 2005]. These tools are available and provide a lot of information. At the same time a lot of work is required to filter out useful information and to analyze it. The next reasonable step that many people working on performance measurement and tuning attempt to do is to create an integrated and preferably automated solution which incorporates in it the best features of the available standalone tools.

Such tools set of benchmarks and frameworks have appeared as the well known package lmbench, which is actually a set of utilities for measurement of such characteristics as memory bandwidth, context switching, file system, process creating, signal handling latency, etc. It was initially proposed and used as a universal performance benchmarking tool for Unix-based systems. There were several projects intended to develop new micro benchmark tools on the basis of lmbench in order to improve measurement precision and applicability for low-latency events by using high-resolution timers and internal loops with measurement of the average length of events calculated through a period of time, such as Hbench-OS package [Haddad, 2006]. It is noticeable that besides widely used performance benchmarks, there are examples of availability benchmarks that are specifically intended to evaluate a system from the high availability and maintainability point of view by simulating failure situations over a certain amount of time and gathering corresponding metrics [Haggander, Lundberg, 2000].

Frameworks to run and analyze the benchmarks were the next logical step to customize this time-consuming process of performance characterization. Usually a framework is an automated tool providing additional customization, automation, representation, and analysis means on top of one or several sets of benchmarks. It makes process of benchmarking easier, including automated decision making about the appropriate amount of cycles needed to get trustworthy results [Wright et al., 2005].

Therefore, we can see that there are a number of tools and approaches one may want to consider and use to characterize a Linux-based system in terms of performance. Making the choice we always keep in mind the main purpose of the performance characterization. Usually people pursue getting these characteristics in order to prove or reject the assumption that a particular system will be able to handle some specific load. So if you are

working on a prototype of a Linux-based server for use as a wireless base site controller that should handle e.g. one thousand voice and two thousand data calls, would you be happy to know from the benchmarks that your system is able to handle e.g. fifty thousand TCP connections? The answer isn't trivial in this case. To make sure we have to prepare a highly realistic simulated environment and run the test with the required number of voice and data calls. It is not easy, even if the system is already implemented, because you will have to create or simulate an external environment that is able to provide an adequate type and amount of load and which behaves similarly to a live wireless infrastructure environment. In case you are in the design phase of your system it is just impossible. You will need to build your conclusion on the basis of a simplified system model. Fortunately, there is another approach—to model the load, not the system. Looking at the architecture we can assume what a specific number of voice and data calls will entail in the system in terms of TCP connections, memory, timers, and other resources required. Having this kind of information, we can use benchmarks for the identified resources and make the conclusion after running and analyzing these benchmarks on the target HW/SW platform, without the necessity of implementing the application and/or environment. This approach is called workload characterization [Avritzer et al., 2002].

Looking back to the Introduction section, we see that all the target questions of Performance and Availability characterization are covered by the tools we have briefly looked through above. At the same time there is no single universal tool that is able to address all these questions. Further in the paper we are introducing the Characterization of Performance and Availability (CAP) tool that combines the essential advantages of all the approaches considered in this chapter and provides a convenient framework to perform comprehensive Linux-based platforms characterization for multiple purposes.

CAP Architecture

1. Experimental Approach.

Anyone who is trying to learn about the configuration of Linux servers running in mission-critical environments and running complex applications systems will have to address the following challenges:

- An optimal configuration, suitable for any state of environmental workload, does not exist;
- Systems are sophisticated: Distributed, Multiprocessor, Multithreaded;
- Hundreds or even thousands of configuration parameters can be changed;
- Parameters can be poorly documented, so the result of a change for a group of parameters or even single parameter can be totally unpredictable.

Based on the above described conditions, an analytical approach is scarcely applicable, because a system model is not clear. An empirical approach could be more applicable to find optimal configuration of a system, but only experimental evaluation can be used to validate the correctness of optimal configuration on a real system. The heart of CAP is the concept of the experimentation. A single experiment consists of the following parts:

- Input parameters: let us call them Xs. Input parameters are all what you want to set up on a target system. Typical examples here are Linux kernel variables, loader settings, and any system or application settings.
- Output parameters: let us call them Ys. Output parameters are all that you want to measure or gather on a target system: CPU and Memory utilization, any message throughput and latency, system services bandwidth, and more. Sources for Ys could be: /proc file system, loaders output, profiling data, and any other system and application output.

- Experiment scenarios: An experiment scenario is a description of actions which should be executed on target hosts.

Typical experiment scenario follows a sequence of action: setup Xs that can't be applied on-the-fly (including execution required actions to apply such Xs like restart node or processes, down and up network interfaces, etc.), then setup Xs that can be applied on-the-fly and loader's Xs, start loaders, next setup Xs like: schedule policy, priority, CPU binding etc., finally collect Ys such as CPU/Memory usage, stop loaders, and overall statistics.

Every scenario file may use preprocessor directives and S-Language statements. S-Language is a script language which is introduced specifically for the project. Both preprocessor and S-Language are described in more detail following. One of the important parts of the scenario executor is a dynamic table of variables. Variable is a pair-variable name and variable value. There are two sources of the variables in the dynamic table:

- Xs (Input variables). They are coming from an experiment.
- Ys (Collected variables). They are coming from remote hosts.

In the case of input variables, the names of the variables are provided by the XML-formatted single experiment file. In the case of collected variables, the names of the variables are provided by scripts or other executables on the target hosts' side. Whenever the same executable could be run on many different hosts, a namespace mechanism is introduced for the variable names. A host identifier is used as a namespace of the variable name.

2. Overview of CAP Architecture

The simplified CAP architecture is presented in the Figure 1.

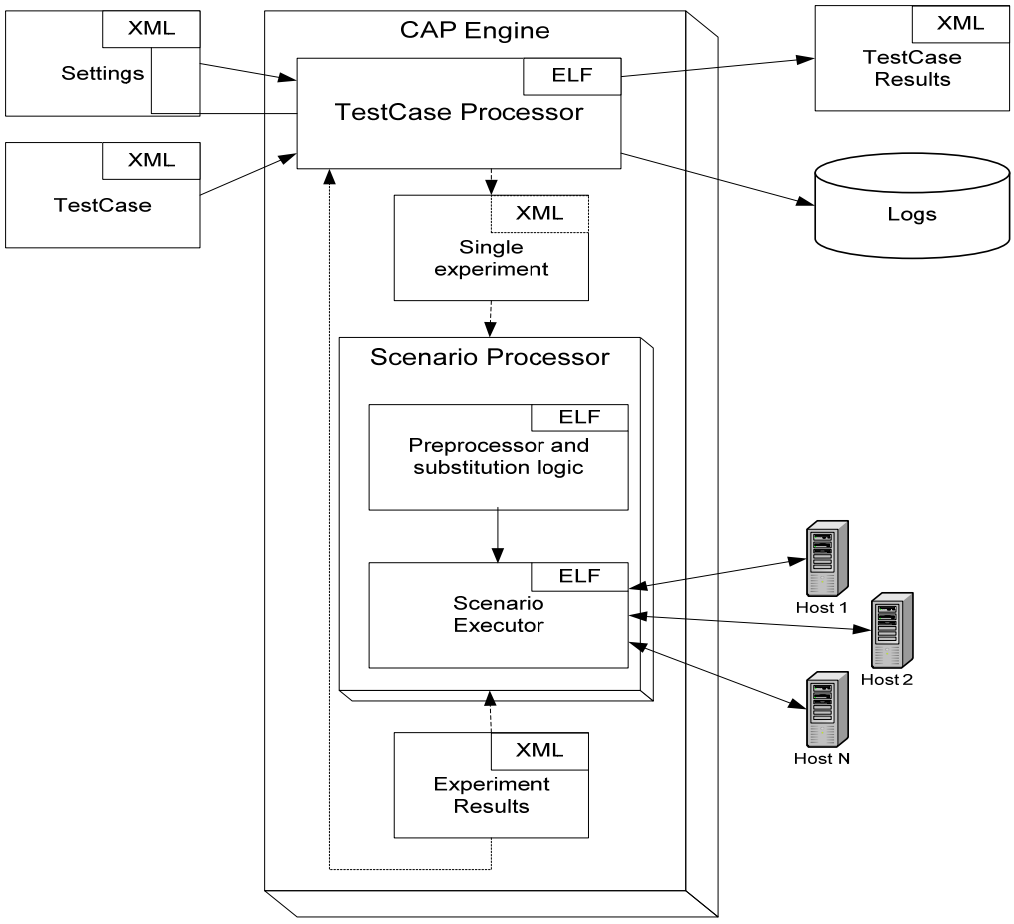


Figure 1. CAP architecture

A test case consists of a set of experiments. Each experiment is essentially a set of Xs that should be used while executing a scenario. Set of Xs within one Test Case boundaries is constant and only values of these Xs are variable. Each experiment is unambiguously linked to a scenario. A scenario resides in a separate file or in a group of files. The CAP engine overall logic is as follows:

- Takes a test case file;
- For each experiment in the test case, performs the steps below;
- Selects the corresponding scenario file;
- Executes all the scenario instructions using settings for fine tuning the execution logic;
- Saves the results into a separate result file;
- Saves log files where the execution details are stored.

Test Cases. The basic unit of test execution is an experiment. A single experiment holds a list of variables, and each variable has a unique name. Many experiments form a test case. The purpose of varying Xs' values depends on a testing goal. Those Xs' names are used in scenarios to be substituted with the values for a certain experiment.

Scenarios. A scenario is a description of actions which should be executed on target hosts in a sequence or in parallel in order to set up Xs' values in accordance with Test Case/Experiments and gather the values of Ys. The solution introduces a special language for writing scenarios. The language simplifies description of actions that should be executed in parallel on many hosts, data collection, variable values, substitution, etc.

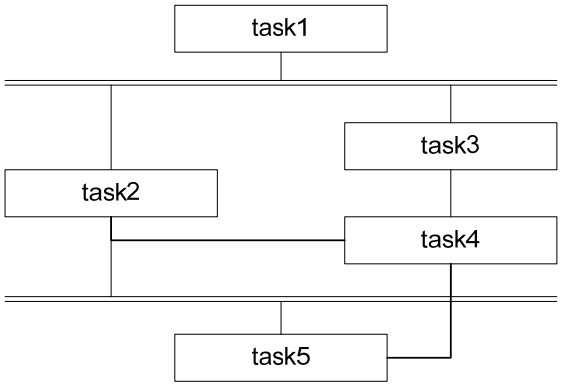
Results. The results files are similar to Test Case files. However, they contain set of Ys coming from target hosts and from input experiment variables (Xs).

The scenario processor consists of two stages, as depicted in the figure above. At the bottom line there is a scenario executor which deals with a single scenario file. From the scenario executor's point of view, a scenario is a single file; however, it is a nice feature to be able to group scenario fragments into separate files. To support this feature the preprocessor and substitution logic is introduced in the first stage. The standard C programming language preprocessor is used at this stage, so anything which is supported by the preprocessor can be used in a scenario file. Here is a brief description of the C preprocessor features which is not a complete one and is given here for reference purposes only:

- Files inclusion;
- Macro substitutions;
- Conditional logic.

Summarizing, the complete sequence of actions is as follows: The single experiment from the Test Case is applied to the scenario file. It assumes macro substitutions of the experiment values (Xs), file inclusions, etc. The scenario executor follows instructions from the scenario file. While executing the scenario some variables (Ys) are collected from target hosts. At the end of the scenario execution, two files are generated: a log file and a results file. The log file contains the report on what was executed and when, on which host, as well as the return codes of the commands. The results file contains a set of collected variables (Xs and Ys).

The main purpose of the introduced S-Language is to simplify a textual description of the action sequences which are being executed consecutively and/or in parallel on many hosts. Figure 2 shows an example of a task execution sequence.



(a) Block diagram

```
TITLE "Scenario example"
#include "TestHosts.incl"
#include "CommonDef.incl"
/* Task1 */
WAIT ssh://USER:PASSWD @ {X_TestHost} "set_kernel_tun.sh SEM \
{X_IPC_SEM_KernelSemmni} {X_IPC_SEM_KernelSemopm}"
PARALLEL
{
/* Task2 */
COLLECT @ X_TestHost "sem_loader -d {X_Duration} \
-r {X_IPC_SEM_NumPVOps} -t {X_IPC_SEM_LoaderNumThreads}"
SERIAL
{
/* Task3&4 */
COLLECT @ {X_TestHost} "get_overall_CPUusage.pl -d
{X_Duration}"
COLLECT [exist(Y_Memory_Free)] @ {X_TestHost} \
"get_overall_Memoryusage.pl -d X_Duration"
}
}
/* Task5 */
NOWAIT [IPC_iteration >1] @ {X_TestHost} "cleanup_timestamps.sh"
```

(b) S-Language Code

Figure 2. Scenario example

ExecCommand is a basic statement of the **S-Language**. It instructs the scenario executor to execute a command on a target host. The non-mandatory **Condition** element specifies the condition when the command is to be executed. There are five supported command modifiers: **RAWCOLLECT**, **COLLECT**, **WAIT**, **NOWAIT**, and **IGNORE**. The At Clause part specifies on which host the command should be executed. The At Clause is followed by a string literal, which is the command to be executed. Substitutions are allowed in the string literal.

3. CAP Agents

We are going to refer to all software objects located on a target system as CAP agents. The server side of CAP does not contain any Performance and Availability specifics, but it is just intended to support any type of complex testing and test environment. Everybody can use CAP itself to implement their own scenario and target agents in order to solve their own specific problem related to the system testing and monitoring.

CAP agents, which are parts of the CAP tool, are the following:

- Linux service loaders;
- Xs adjusting scripts;
- Ys gathering scripts.

Firstly let us consider loaders as one of the most interesting part of CAP agents. Every loader receives a command line argument which provides the number of time slices a base interval (usually one second) is going to be divided into. For example: <loader> --rate 20 means that a one-second interval will be divided into 20 slices. At the very beginning of each time slice, a loader calls a function which performs a required functionality/load. The functionality depends on a loader type. For example, the file system loader performs a set of file operations, while the shared memory loader performs a set of shared memory operations, and so on. If the required functionality has been executed before the end of the given time slice, a loader just sleeps until the end of the slice. If the functionality takes longer than a time slice, the loader increments the corresponding statistic's counter and proceeds. There are several common parameters for loaders.

Input:

- The first one is a number of threads/processes. The main thread of each loader's responsibility is to create the specified number of threads/processes and wait until they are finished. Each created thread performs the loader-specific operations with the specified rate.
- The second common thing is the total loader working time. This specifies when a loader should stop performing operations.

- Loaders support a parameter which provides the number of operations per one "call of F() functionality." For example, a signal loader takes an argument of how many signals should be sent per time slice. This parameter, together with the number of threads, rate, and number of objects to work with (like number of message queues), gives the actual load.
- Besides that, each loader accepts specific parameters (shared memory block size in kilobytes, message size, and signal number to send, and so on).

Output:

- Number of fails due to the rate requirement not being met.
- Statistics—figures which are specific for a loader (messages successfully sent, operations successfully performed, etc.)

The loaders implementation described above allows not only the identification of Linux service breakpoints, but also—with help of fine rate/load control—the discovery of the service behavior at different system loads and settings. The following loaders are available as part of CAP tool:

- IPC loaders: Shared memory loader; Semaphore loader; Message queues loader; Timer loader.
- CPU loaders: CPU loader; Signal loader; Process loader.
- IP loaders: TCP loader; UDP loader.
- FS loader (File & Storage);
- Memory loader.

One more CAP agent is PPA (Precise Process Accounting). PPA is a kernel patch that has been contributed by Motorola. PPA enhances the Linux kernel to accurately measure user/system/interrupt time both per-task and system wide (all stats per CPU). It measures time by explicitly time-stamping in the kernel and gathers vital system stats such as system calls, context switches, scheduling latency, and additional ones. More information on PPA is available from the PPA SourceForge web site: http://sourceforge.net/projects/ppacc/.

The Ys gathering is used in the SU best moment definition and will be discussed further. Now we will show how to use CAP.

CAP in Use

Let us assume that you already have the CAP tool and you have decided to use it for your particular task. First of all, you will have to prepare and plan your experiments:

- Identify list of input parameters (Xs) that you would like to set up on the target. That could be kernel parameters, a loader setting like operational rate, number of processes/threads, CPU binding,etc.
- Identify list of output parameters (Ys) that you would like to measure during an experiment: everything you want to learn about the system when it is under a given load.

If we are talking about Linux systems, you are lucky then, because you can find in the CAP toolset all the necessary components for the CAP agent that have been already implemented: set scripts for Xs, get scripts for Ys, and predefined scenarios for Linux's every service. If you are not using Linux, you can easily implement your own scripts, scenarios, and loaders. When you have identified all the parameters that you want to set up and measure, you can move on to plan the experiments to run.

Performance Profiling

- Set up predefined/standard configuration for the kernel and system services.

- Setup loaders to generate the workload as stated in your requirements.
- Perform experiments.
- Check whether the measured data shows that requirements are met.

Load Testing

- Set up predefined/standard configuration for the kernel and system services.
- Use a long experiment duration.
- Mix the workload for all available services.
- Vary workloads.
- Vary the number of threads and instances for the platform component.
- Analyze system behavior.
- Check that Ys are in valid boundaries.

Stress Testing

- Use a high workload.
- Operate by the loader with the target to exhaust system resources like CPU, memory, disk space, etc.
- Analyze system behavior.
- Check that all experiments are done and Ys are in valid boundaries.

Performance tuning

- Plan your experiments from a workload perspective with the target to simulate a real load on the system.
- Vary Linux settings, process affinity, schedule police, number of threads/instances, etc.
- Analyze the results in order to identify the optimal configuration for your system. Actually we believe a multi-objective optimization can be very useful for that. This approach is described in more detail later on.

System Modeling

- Take a look at your design. Count all the system objects that will be required from an OS perspective, like the number of queues, TCP/UDP link, timers, semaphores, shared memory segment, files, etc.
- Examine your requirements in order to extrapolate this on every OS service workload.
- Prepare test case(s).
- Analyze the obtained results to understand whether your hardware can withstand your design.

Another kind of system modeling is described in the following chapter and uses the Ys gathering functionality of the CAP tool.

Software Upgrade Prediction Concept

As it was stated in the introduction the described tool is expected to be used by the author for the best moment of software upgrade definition. For a start I will explain why it is important to know the best moment for SU.

One of the most important requirement for software systems which work in 24/7 mode is the possibility of online SU. The simple duplication of the system might seem to be the possible way to meet this requirement. Nowadays it is rather widespread solution – all the components of the system are set in pairs – one active and one standby. If the active element is lost the standby one takes over all the functionalities. However this solution does not take

into account the time on setting all the connections between the standby and other active elements of the system as well as can be extremely expensive. The introduced method is aimed not on the upgrade of the system without services losses but on the SU best moment (from the services losses point of view) prediction – note that the SU term does not surely mean the whole system upgrade – it could be some components upgrade.

The principal scheme of the method is showed in the Figure 3.

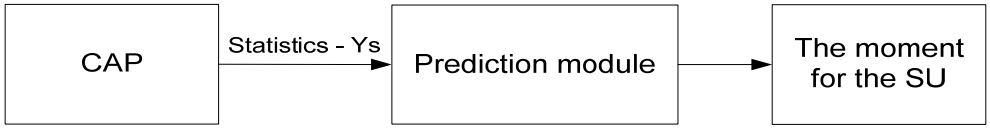


Figure 3. Principle scheme of the SU moment definition

The prediction module will perform following actions:

- Process the statistic information and get probability distribution functions for all characteristics (density functions) – $P_i(Y_i), i \in 1...n$, n is the number of characteristics Y_i. The methods here could be, for example:
  - Testing of statistical hypothesis
  - Approximation by known distribution functions
  - Monte Carlo method
- Perform a correlation analysis to identify dependencies - influence coefficients calculation. The methods here could be, for example:
  - Elementary methods (parallel comparison of samples etc.)
  - Complex methods (dispersion analysis methods, theory of correlation and regression methods, multivariate analysis methods etc.)
- The third and the most labour-intensive step of the prediction module development is the statistic simulation which task is to get the functional model which will describe the real system behavior (in terms of system modules load). The book “Statistic simulation” by Ermakov S.M. will be helpful here [Ermakov, 1976]. Object identification, which is considered in [Черноруцкий, 2004], could be an alternative to the complex method of the statistic simulation and in case of known type of functional model operator will turn into the parametric identification which could be simply implemented.
- The final step of the prediction module is the parameter optimization of the model got in the previous step. The goal of this optimization is to determine the moment of time when the whole system, or the module being updated, have the least load in terms of services usage. The teaching aid mentioned above, [Черноруцкий, 2004], fully describes this task solution.

Let us go into details of the third and the fourth steps. As it was said the goal of the third step is to get the operator F which will describe the real system behavior in terms of services availability and its components loads. Using the parametric identification the operator F will get the vector Y of n stochastic parameters (from Statistic Module), one dependent parameter t (time) - and will return the vector L of m services availabilities

$$F(Y,t) = L, \qquad Y = \begin{Bmatrix} y_1 \\ ... \\ y_n \end{Bmatrix}, \; L = \begin{Bmatrix} l_1 \\ ... \\ l_m \end{Bmatrix}$$

The time (t) is the dependent parameter from the possibility to vary point of view – the task is to determine the best moment [Черноруцкий, 2004].

This operator F will be obtained by minimization of misalignment function

$$\psi(Y,t,P)=\sum_{i=1}^{m}\left[L_i(Y,t)-L_{iM}(Y,t,P)\right]^2;\quad \left(\sum_{j=1}^{N}\psi(Y_j,t_j,P)\to\min\right)\Rightarrow P$$

Here L_i is the output of the real system, L_{iM} is the model output and P is the vector of unknown model parameters.

The problem here could become the interference coefficients calculation of the characteristics for the every system component. These coefficients are quite important for the specified task resolution because the connections between different parts of the system often define the services availability (presence). Here we mean systems components interconnection which is the essential part of every distributed system. These problem will be solved by the means of statistical simulation which takes into account the influence of different components on each other [Ермаков и др., 1976].

The fourth step further will be the parametric optimization implementation for the F operator – identification of the moment of time when the F operator will possess the minimal value.

$$\left(F(Y,t)\to\min\right)\Rightarrow t_{\min}$$

This task will be solved with the help of mathematical programming. It is very important to note that the CAP tool starting and the needed statistics gathering will be done periodically and with high frequency which will enable the prediction module to recreate the system model approximating it to the real system.

Future Plans for Further Approaches Development

In its current version, the CAP tool allows us to reach the goals we set for ourselves, and has a lot of potential opportunities for further improvements. We realize a number of areas where we can improve this solution to make it more and more applicable for a variety of performance- and availability-related testing. Through real applications of the CAP tool to existing telecommunication platforms, we realized that this approach from the experimental perspective could be very fruitful. However, we noticed that results may vary depending on the overall understanding and intuition of the person performing the planning of the experiments. If a person using CAP does not spend enough time to investigate the nature of the system, she/he may need to spend several cycles of experiments-planning before she/he identifies right interval of system parameters — i.e., where to search for valuable statistical results. This is still valuable, but requires the boring sorting of a significant amount of statistical information. In reality, there may be more than one hundred tunable parameters. Some of them will not have any impact on certain system characteristics; others will certainly have impact. It is not always easy to predict it just from common perspective. An even more difficult task is to imagine and predict the whole complexity of the inter-relationships of parameters. Automation of this process seems to be reasonable here and there is a math theory devoted to this task that we believe could be successfully applied. We are talking about multi-parametric optimization. It is a well described area of mathematics, but many constraints are applied to make this theory applicable for discrete and non-linear dependencies (which are true for most of dependencies we can meet in system tunable parameters area). We are currently looking for numeric approaches for these kinds of multiparametric optimizations—e.g., NOMAD (Nonlinear Optimization for Mixed variables and Derivatives) [Audet, Orbany, 2004], GANSO (Global And Non-Smooth Optimization) [CIAO, 2005], or ParEGO Hybrid algorithm [Knowles , 2005]. A direct search for the optimal parameters combination would take too much

time (many years) to sort out all possible combinations, even with fewer than one hundred parameters. Using math theory methods, we will cut the number of experiments down to a reasonable number and shorten the test cycle length in the case of using CAP for load and stress testing purposes. Our discussion in the previous paragraph is about performance tuning, but it is not the only task that we perform using the CAP tool. Another important thing where the CAP tool is very valuable is the investigation of performance and availability issues. Currently, we perform analysis of the results received manually through the use of packages such as MiniTAB, which in many case is time consuming. Our plan is to incorporate statistical analysis methods in the CAP tool in order to allow it to generate statistical analysis reports and to perform results visualization automatically by using GNU GSL or similar packages for data analysis, and such packages as GNUPLOT, LabPlot, or Root for visualization [Galassi et al., 2006] [Brun, Rademakers, 1996].

Conclusion

In this paper, we have provided an overview of specific performance and availability challenges encountered in Linux servers running on telecommunication networks, and we demonstrated a strong correlation between these challenges and the current trend from Linux vendors to focus on improving the performance and availability of the Linux kernel.

We have briefly described the existing means to address basic performance and availability problem areas, and presented the reason why in each particular case the set of tools used should be different, as well as mentioned that in general the procedure of performance and availability characterization is very time- and resource consuming.

We presented on the need to have a common integrated approach, i.e., the CAP tool. We discussed the tool architecture and used examples that significantly simplify and unify procedures of performance and availability characterization and may be used in any target problem areas starting from Linux platform parameters tuning, and finishing with load/stress testing and system behavior modeling.

After all we presented a new concept of the SU best time prediction on the basis of statistics provided by the CAP tool. This new method can be advantageous for the SU either of existing systems which require 24/7 operational mode or of the systems which are being developed and the decision about their architecture should be made – now there is no need to secure ourselves by duplicating the system – we can just predict the best moment to execute the SU. It is important to note that such approach might not meet the requirement like five or six nines when the systems downtime should be approximately 5 min per year.

All in all the developed tool from one side provides an ability to perform an overall testing and optimization of the system and from the other side – to gather various statistics on the systems components load which are used by the author to predict the best moment of SU.

Bibliography

[Haddad, 2006] Ibrahim Haddad. Linux and Open Source in Telecommunications. Linux Journal, September 2006. http://www.linuxjournal.com/article/9128.

[Lehrbaum, 2002] Rick Lehrbaum. Embedded Linux Targets Telecom Infrastructure. LINUX Journal, May 2002. http://www.linuxjournal.com/article/5850.

[Haddad, 2001] Ibrahim Haddad. Open-Source Web Servers: Performance on a Carrier-Class Linux Platform. Linux Journal, November 2001. http://www.linuxjournal.com/article/4752.

[Haggander, Lundberg, 2000] Daniel Haggander and Lars Lundberg. Attacking the Dynamic Memory Problem for SMPs. In the 13th International Conference on Parallel and Distributed Computing System (PDCS'2000). University of

Karlskrona/Ronneby Dept. of Computer Science, 2000.
http://www.ide.hk-r.se/~dha/pdcs2.ps.

[Korhonen, 2004] Jouni Korhonen. Performance Implications of the Multi Layer Mobility in a Wireless Operator Networks, 2004. http://www.cs.helsinki.fi/u/kraatika/Courses/Berkeley04/KorhonenPaper.pdf.

[Mattar et al., 2007] Karim Mattar, Ashwin Sridharan, Hui Zang, Ibrahim Matta, and Azer Bestavros. TCP over CDMA2000 Networks : A Cross-Layer Measurement Study. In Proceedings of Passive and Active Measurement Conference (PAM 2007). Louvain-la-neuve, Belgium, 2007.
http://ipmon.sprint.com/publications/uploads/1xRTT_active.pdf.

[Nehra et al., 2007] Neeraj Nehra, R.B. Patel, and V.K. Bhat. A Framework for Distributed Dynamic Load Balancing in Heterogeneous Cluster. Journal of Computer Science v3, 2007.
http://www.scipub.org/fulltext/jcs/jcs3114-24.pdf.

[Ong et al., 2005] Hong Ong, Rajagopal Subramaniyan, and R. Scott Studham. Performance Implications of the Multi Layer Mobility in a Wireless Operator Networks, 2005. Performance Modeling and Application Profiling Workshop, SDSC, http://xcr.cenit.latech.edu/wlc/papers/openwlc_sd_2005.pdf.

[Shende et al., 2007] Sameer Shende, Allen D. Malony, and Alan Morris. Workload Characterization using the TAU Performance System, 2007.
http://www.cs.uoregon.edu/research/paracomp/papers/talks/para06/para06b.pdf.

[Karelin et al., 2008] Karelin I., Gavrilova T., Lyubimov B. “A Log tool for software systems analyses” // “Information Technologies and Knowledge”, Volume 2/2008, Number 4, p. 65
http://www.foibg.com/ibs_isc/ibs-04/IBS-04-p09.pdf

[Best, 2005] Steve Best. LinuxR Debugging and Performance Tuning: Tips and Techniques. Prentice Hall PTR, 2005. ISBN 0-13-149247-0.

[Haddad, 2006] Ibrahim Haddad. Linux and Open Source in Telecommunications. Linux Journal, September 2006. http://www.linuxjournal.com/article/9128.

[Haggander, Lundberg, 2000] Daniel Haggander and Lars Lundberg. Attacking the Dynamic Memory Problem for SMPs. In the 13th International Conference on Parallel and Distributed Computing System (PDCS’2000). University of Karlskrona/Ronneby Dept. of Computer Science, 2000.
http://www.ide.hk-r.se/~dha/pdcs2.ps.

[Wright et al., 2005] Charles P. Wright, Nikolai Joukov, Devaki Kulkarni, Yevgeniy Miretskiy, and Erez Zadok. Towards Availability and Maintainability Benchmarks: A Case Study of Software RAID Systems. In proceedings of the 2005 Annual USENIX Technical Conference, FREENIX Track, 2005.
http://www.am-utils.org/docs/apv2/apv2.htm.

[Avritzer et al., 2002] Alberto Avritzer, Joe Kondek, Danielle Liu, and Elaine J. Weyuker. Software Performance Testing Based on Workload Characterization. In Proceedings of the 3rd international workshop on Software and performance, 2002. http://delivery.acm.org/10.1145/590000/584373/p17-avritzer.pdf.

[Ермаков и др., 1976] Ермаков С. М., Михайлов Г. А. «Курс статистического моделирования» // Главная редакция физико-математической литературы изд-ва «Наука», М., 1976

[Черноруцкий, 2004] Черноруцкий И. Г. «Методы оптимизации в теории управления» // Учебное пособие, изд. Питер, Санкт-Петербург, 2004.

[Audet, Orbany, 2004] Charles Audet and Dominique Orbany. Finding optimal algorithmic parameters using a mesh adaptive direct search, 2004.http://www.optimization-online.org/DB_HTML/2004/12/1007.html.

[CIAO, 2005] CIAO. GANSO. Global And Non-Smooth Optimization. School of Information Technology and Mathematical Sciences, University of Ballarat, 2005. Version 1.0 User Manual. http://www.ganso.com.au/ganso.pdf.

[Knowles , 2005] Joshua Knowles. ParEGO: A Hybrid Algorithm with On-line Landscape Approximation for Expensive Multiobjective Optimization Problems. In Proceedings of IEEE Transactions on Evolutionary Computation, Vol. 10, No. 1, 2005. http://ieeexplore.ieee.org/iel5/4235/33420/01583627.pdf .

[Galassi et al., 2006] Mark Galassi, Jim Davies, James Theiler, Brian Gough, Gerard Jungman, Michael Booth, and Fabrice Rossi. GNU Scientific Library. Free Software Foundation, Inc., 2006. Reference Manual. Edition 1.8, for GSL Version 1.8. http://sscc.northwestern.edu/docs/gsl-ref.pdf.

[Brun, Rademakers, 1996] Rene Brun and Fons Rademakers. ROOT – An Object Oriented Data Analysis Framework. In Proceedings of AIHENP conference in Laussanne, 1996.
http://root.cern.ch/root/Publications.htm

Authors' Information

Igor Karelin – Bachelor of information technologies. Student of Saint Petersburg State Polytechnic University (master's degree program), Software Engineer, Motorola Software Group; e-mail: ikarus47@mail.ru

Major Fields of Scientific Research: telecommunication systems analysis, statistical modeling, parameter optimization.