CHAPTER 4: PROPOSED METHODOLOGY

4.1 Introduction

In our proposed methodology, we have considered four performance tests, in which firstly all tests contribute to better understanding of the Linux, according to the Linux under z/VM and z/VM. This chapter will provide test results which are generally accepted and believed for optimization and related suggestions. Moreover, there are some recommendations which are generally industry specific optimization suggestions.

This section is divided into main four sections which represents the performance of test program or system investigation. In this chapter, Test no. 1 to 3 cover the processors, storage usage and I/O disk. The fourth test is based upon the combination of first three chapters which examine the impact to add third virtualization level. In this, test programs, monitoring tools and scripts are developed to observe the values of interest and influence or in particular ways stress the system. These tests are dependent upon different workloads, some tools and programs. The “tools and methods” sections will introduce programs and tools consecutively; thereafter some dependencies will explain the differences to preceding tests that have been emphasized [73][76][78][79][80].

4.1 Test 1: Processors per Linux guest

4.1.1 Motivation

Various sources point to the virtual processor as guest that will be limited as much as possible, since unimportant processors define an overhead. Some multiprocessors are overhead related to the “Diagnose x’44” instruction.

The “Voluntary End of Time Slice” or Diagnose x’44’ instruction can be used from virtual machines with several virtual processors, when subsist the spin lock. The scheduler can be informed with reminder of CPU time slice that should be allocated to issue of the virtual processor as per instruction, which is no longer useful. The priority of
virtual processor can be lowered as compared to the virtual processors with similar virtual machines. It will enable the lock and release the lock on the virtual processor.

Spin lock setups with a single processor does not make any sense, given that it does make sense to make the CPU actively wait in a loop in order to receive a lock, which requires re-scheduling of the processor to another process/thread in order to be released. The Linux guest can run in the virtual machines with CPU that will not invoke the Diagnose x’44’, but simply internally reschedule. The assumption states that Linux guests run slower in a multi processor virtual machine with mainly single threaded workloads. It happens because the Linux tend to use the spin locks that will introduce the Diagnose x’44’ instructions that will take time longer other than internal scheduling in the single processor of virtual machine.

The test can be designed to search configuration of optimum process for Linux guests in the specific company z/VM environment. The test displays how many virtual processors can be influenced typically, duration of real and demand workloads/ jobs. The diagram x’44’ gives instruction which can be determined to confirm that 2-CPU guests call to instruction.

4.1.2 Measurement methods and tools

4.1.2.1 QMONITOR: Specialized monitoring in general

Linux offers various tools and features that can be used while monitoring or testing the system. These tools are free, need not to mention “raw data” while providing the kernel in /proc/ file system e.g. stat, vmstat and meminfo. Unfortunately output is hard to interpret or comprehensive in the “live” situation which output format will be seldom suited for the data analysis.

Mostly, monitor programs and products exist to offset difficulties. In most cases these monitors’ products are overloaded, their rate is too low or their customization capabilities are not sufficient enough to allow for specialized measurements. Linux systems come with a variety of small capabilities and tools that make it possible to develop a customized monitoring service with comparatively small amount of work.

qMontior
This monitoring solution is named as qMonitor and has been developed for thesis with following in mind:

- Abilities for Extreme customization
- Real time graphing based on Real time data delivery
- Data gathering option for the later data analysis
- Simple data format allow the processing in standard tool
- Connectivity to allow the remote data graphing and gathering
- High importance of simplicity rather than avoiding communication overhead etc.

qMonitor utilizes a mix of xinetd (the eXtended InterNET Daemon), pipes, scripts and values from /proc and programs.

$qMonitor/mon.sh$

```bash
#!/bin/bash
LINES=100
export LINES
while test 1; do
  read command
  case "$command" in
    (t) /usr/bin/top -b -n 1 | /qMonitor/ora_mr.awk;;
    (m) /qMonitor/memMeminfo.awk /proc/meminfo
        /qMonitor/memVmstat.awk /proc/vmstat ;;
    (mi)/bin/date +"Test date: %d/%m/%Y %H:%M:%S"
        echo "time,sec.nano,app,slab,PageTables,vmallocUsed,Buffers,
            Cached,SwapEvictedFromRealMem,MemFree,SwapCached,
            SwapFree,Active,Inactive,pgfault,Committed_AS,Mapped,
            pgmajfault,pswpin,pswpout";;
    (*) break;;
  esac
done
```
In this host being, monitor xinetd can configure to particular TCP port and connect it with script. The xinetd service can invoke the script when the tcp connection is being opened. The xinet can redirect all the data which it has received from the client to stdin and can send data which is written to the stdout back to the client. The client can control the script and receive output from it. Since xinetd can be available in some Linux distribution which is easy to create highly customized server.

The real monitor process can be controlled and instantiated by client. It can connect to configure the TCP port and xinetd can invoke script mon.sh, if connection can be successfully opened. Now client can send character representation wanted monitor data, for more instance ‘t’ represent information from the top. The mon.sh can parse input and invoke to the corresponding command. The command can perform important measurements which format output into separated comma string that will be written to stdout and return to client.
Figure 4.1: The monitor setup can be constructed to provide various sets of measurements to workstation via TCP/IP socket.

Every time client can send newest code; corresponding data string can generate and return. The client easily parses comma separated string and potentially graph values in real-time. Write the unaltered data which is directly into text file which is generated as “CSV” file, which can be readable for spreadsheet programs for later graphing and analysis. In any case, the clients with logging and graphing capabilities have developed in Java language, but any type of script could be used.
Chapter 4: Proposed Methodology

Figure 4.2: The GUI of developed Java client program that is used with qMonitor monitor.

4.1.2.2 Data gathering from Linux

The test data can be derived from standard UNIX tool “top”. The top program can provide the realistic time of run system summing up information from both systems as well as information on the individual processes. The individual processes’ information makes top helpful in this particular case, since it provides admittance to exact needed information.

The mon.sh-script can invoke to the “top” in batch mode with repetition that disable interactive user interface and transfers single output set stdout. The output can be piped into an awk script that can parse it and generates a comma separated string which includes the following values:

- Values related to the processes possessed by application user (mrdata):
  - Number of processes sleeping, running, in uninterruptible sleep
Chapter 4: Proposed Methodology

- Memory usage
- % CPU usage

Figure 4.3: The selection menu from the “IBM Performance Toolkit” web interface
(image source - IBM VM Performance Toolkit overview) [78]

4.1.2.3 Data gathering outside Linux

This Toolkit can be used to confirm the readings of CPU which are mentioned above, that cannot be trusted basically, since the current Linux version is not aware of the time VM uses service to other guests.
The Toolkit is made as licensed product that cannot be purchased separately, however shipped as component of z/VM. The Toolkit offers a variety of information, and logging options [73][78]. Similar to other things running in virtual machine, it too runs on virtual machine but it mostly depends upon the data which is provided by the CP, that allows it to show picture of whole virtual environment from the “hypervisor view”.

Figure 4.4: The “Performance Toolkit of IBM for VM” runs in virtual machine (typically “perfsvm”) but data can be given from CP. (image source - IBM VM Performance Toolkit overview) [78]

The CPU toolkit readings have been used to verify the workloads used are single threaded and that the virtual machines usually occupy single processor regardless of number of virtual processors. The toolkit will be used here to verify that the tests have been completed in “stable” environment without any huge fluctuations which could have been caused by some guests.

The IBM Toolkit helps to verify that CPUs are influencing number of diagnose x’44 instructions. Apparently it is not possible to operate the number per user. These “Privileged Operations” screen (#4) manually can be used to record and supervise current x’44 count.

4.1.3 Workloads

This test can be performed to use the workload definition where WL1 or WL2, which have some following characteristics.
Workload 1 (WL1)

Workload 1 is the “comprehensive payroll processing”: Various independent applications are executed during the payroll processing that result in birth and termination of various new processes. Various programs can access the database individually and “manipulate” it. The workload can be of “mixed nature”: The combination of used CPU and use of disk. Typically the CPU usage is fluctuated and hits 100% on one virtual CPU for much longer time.
4.1.4 Test description

The aim is to display, whether number of the virtual processors can be impact on the duration of above workloads. If we want to eliminate the impact from another z/VM guests, the time durations for workloads can be assessed relatively by operating similar workloads in 1-CPU guest and 2-CPU guest in parallel. It is allowed for tests to run in the production environment. These two guests are going concurrently which can be identical except for set of CPUs. For safety precaution, repeat the test with opposite CPU configuration on similar two guests to eliminate the differences of unintended configuration. As an enhancement to “relative” tests which are running along with other guests in parallel, their workloads also executed separately on one guest at time. Firstly 1-CPU guest will run after that 2-CPU guest will be executed. It is important to get Diagnose of X’44 instruction count and it would be impractical to monitor the same for each and individual guests. These individual tests ensures that concurrent processes shouldn’t disturbs the findings.

To conclude x’44 Diagnose overhead, CPU can share the value for 2-CPU guest which must have twice the CPU share for guest with the virtual processor: Recall to z/VM which is divided to guests which is relative to CPU share that b/w guests virtual processors. Whether shares cannot adjust, that could be comparing the PC with 1000MHz processor and PC along with 500MHz processors. Since there are workloads in advance to foremost one threaded, PC with 1000MHz can be outperform the dual processor 500MHz.

The above test execution can find out whether the PC can perform better with one or multiple 1000MHz CPUs where it’s significant that one of processors will remain inactive and mostly available only for Operating System background processing and bookkeeping tasks. Finally, it can be claimed that in the virtual environment, an additional processor will result in an overhead, but on the other hand Linux system still benefit from an additional processor for background task processes.
4.2 Test 2: Memory usage

4.2.1 Motivation

The majority of common storage z/VM configuration guideline for the Linux guests is to restrict the main storage footprint for guests as great as it can be. Linux OS is developed to get the majority out of the available resources and thus it will eventually be using every bit of memory for file system cache and buffer.

Figure 4.5: “Munin-graph” shows the Linux guest of typical memory distribution. Linux is swapping although memory can be used for buffers and file cache.

In virtual memory, usually memory gets overcommitted, which has not having huge file system that buffers in the memory since buffers risk is being back to page of disk. Actually effect may effect to performance penalty which has multiple guests and suddenly these will dependent on same physical disks and paging sub system. On other side memory can be footprint and if it becomes too small then it equally contribute to reduce the performance drastically.
When small memory of Linux “swaps” with the memory pages of disk which will be able to complete their tasks. It will limit of drastically reduce the throughput. The accurate solution is to manage the guest with memory according to the actual application requirement. The analysis allowed us to document and describe how the Linux will utilize the memory perfectly and how z/VM environment will be influenced accordingly. It can be examined with paging and “swapping” from their perspectives and the relative findings will be helpful when determining the size of main storage for z/VM guests.
4.2.2 Measurement methods and tools

4.2.2.1 Data from within Linux

This investigation uses the monitoring solution (qMonitor) which is developed and introduced in the resulting section. The real data can be derived from /proc/meminfo and /proc/vmstat. Basically the /proc file system will contain various files which represents the present state of kernel and system. The vmstat and meminfo files have given the information on the virtual memory and memory utilization respectively. The data (plain text) from the two sources have been parsed by two scripts (memVmstat.awk and memMeminfo.awk), which reformats the output in useful CSS file (comma separated string).

qMonitor/memMeminfo.awk

This script helped us to process the data using /proc/meminfo in CSS file format which optimized the solution for easy graph plotting.

```bash
#!/usr/bin/awk -f

BEGIN {
  TIME="/bin/date +"H:M:S, %s.%N"
}
function Time() {
  TIME | getline t
  close(TIME)
  return substr(t, 1, 10) substr(t, 14, 10)
}
# /<keyword/> => Searches for keyword in beginning of line
  /^Slab/ {Slab=$2}
  /^MemTotal/ {MemTotal=$2}
  /^SwapCached/ {SwapCached=$2}
  /^PageTables/ {PageTables=$2}
  /^VmallocUsed/ {VmallocUsed=$2}
  /^MemFree/ {MemFree=$2}
  /^Buffers/ {Buffers=$2}
  /^Cached/ {Cached=$2}
  /^SwapTotal/ {SwapTotal=$2}
  /^SwapFree/ {SwapFree=$2}
  /^SwapCached/ {SwapCached=$2}
  /^Committed_AS/ {Committed_AS=$2}
  /^Mapped/ {Mapped=$2}
  /^Active/ {Active=$2}
  /^Inactive/ {Inactive=$2}
```
Chapter 4: Proposed Methodology

END {
SwapEvictedFromRealMem = SwapTotal - SwapFree - SwapCached
apps = MemTotal-MemFree-Buffers-Cached-SwapCached-Slab-PageTables-VmallocUsed
printf ("%s," , Time())
printf ("%d," , apps)
printf ("%d," , Slab)
printf ("%d," , PageTables)
printf ("%d," , VmallocUsed)
printf ("%d," , Buffers)
printf ("%d," , Cached)
printf ("%d," , MemFree)
printf ("%d," , SwapCached)
printf ("%d," , SwapEvictedFromRealMem)
printf ("%d," , SwapFree)
printf ("%d," , Committed_AS)
printf ("%d," , Mapped)
printf ("%d," , Active)
printf ("%d," , Inactive)
}

qMonitor/memVmstat.awk

This script will parse the data using /proc/vmstat and it segregates the information on page faults and the number of pages swapped in and out, into a CSS file (comma separated string).

#!/usr/bin/awk -f
# /*keyword/ => Searches for keyword in beginning of line
/^pgfault/ {pgfault=$2}
/^pgmajfault/ {pgmajfault=$2}
/^pswpin/ {pswpin=$2}
/^pswpout/ {pswpout=$2}
END {
printf ("%d," , pgfault)
printf ("%d," , pgmajfault)
printf ("%d," , pswpin)
printf ("%d", pswpout)
}

Memory model and /proc/meminfo data

Table 4.1 introduces the values which are derived from the meminfo and how they will be interpreted. Some confusion will still exist regarding some of the values especially swapCached. For instance an IBM Redbook will claim which SwapCached “will report the cache memory size swapped out for swap devices”. It will not surely make sense.
Table 4.1: Values extracted from memMeminfo.awk from meminfo and their meaning.

<table>
<thead>
<tr>
<th>Slab</th>
<th>Memory used by the kernel for caching different data structures.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Tables</td>
<td>Memory used to map between virtual and physical memory addresses.</td>
</tr>
<tr>
<td>Vmalloc Used</td>
<td>Kernel memory virtually contiguous but not necessarily “physically” contiguous.</td>
</tr>
<tr>
<td>Buffers</td>
<td>Relatively temporary storage for raw disk blocks.</td>
</tr>
<tr>
<td>Cached</td>
<td>Cache for files read from disk (the file cache).</td>
</tr>
<tr>
<td>Mem Free</td>
<td>Unused memory generally available.</td>
</tr>
<tr>
<td>Swap Cached</td>
<td>Memory that once was swapped out and has been swapped back in, but still resides in the swapfile.</td>
</tr>
<tr>
<td>Swap Free</td>
<td>Unused swap memory (on disk from Linux’s perspective).</td>
</tr>
</tbody>
</table>

For easy subsequent data analysis and graphic, some values can be added and calculated to output string with the additional values which are given directly in /proc/meminfo. The initial value “apps” provides an estimation of memory which is occupied with some running programs from the Linux perspective. These values can be calculated with the subtraction of other memory use values from total memory:

\[
\text{apps} = \text{MemTotal} - \text{MemFree} - \text{Buffers} - \text{Cached} - \text{SwapCached} - \text{Slab} - \text{PageTables} - \text{VmallocUsed}
\]

Finally these values “SwapEvictedFromRealMem” can be calculated to represent the pages, that will be evicted from the memory and resides on the swapping disk. The value can be calculated as unoccupied swapping space which is not taken by swapCached that will be according to the Table which still resides in the memory. For more simplification value can denote “swap” in some following analysis section:

\[
\text{Swap} = \text{SwapEvictedFromRealMem} = \text{SwapTotal} - \text{SwapFree} - \text{SwapCached}
\]

It can be noticed which swapCached will be plotted twice, where plotting of memory distribution as stacked “area chart” using some values above.
4.2.2.2 Data from outside Linux

The IBM Performance Toolkit can be used to gather test data available from z/VM. It is important for Linux which doesn’t know about the paging activities inside CP. It can’t be forgotten that z/VM or CP usually over commits on storage front and hence forced to migrate some pages that Linux will believe is a “real” physical storage to disk.

The data of concern is Linux paging behavior of Virtual Manager guest during the investigation. Toolkit screen usually consist of “User paging load” test data which includes paging activity, where pages actually resides (DADS, expanded storage and main storage) and rate of pages shall be migrated from one to another storage space. The particulars of paging will include individual guest data those in general, not logged into file, which will become very supportive to enable benchmarking for users. The sample rate for monitoring shall be set to 15 seconds to present more exhaustive view.

4.2.2.3 Monitors influence on the system

The measured monitoring software’s influence can be neglected. Data can be easily derived via command /proc/ file system which is very fast and proficient. The parsing can be achieved by awk and processing can be achieved with xinetd command simply and then we can use plotting with system resources; in particular when we have typical sample rate into account. The Toolkit won’t influence investigation or system behavior in the general.

4.2.2.4 Test tools: useMem

It is small C language code (using thread as thread library) which has been created to influence or to conquer the memory utilization issue in an easy, reliable and deterministic manner. Its main function includes memory initialize, allocate, and continuous use to vary memory allocation, throughout the test during measurement of access rate. The initialization is quite essential, because initially kernel will allocate and occupy the memory when it is used. Hence without starting of arguments program will start by
Chapter 4: Proposed Methodology

initializing and then it would consume the allocated memory 64MB and initiate one running worker thread loop through allocated memory.

The main rationale of working threads can be affirmed to keep memory sizing active. Now when we provide input “g” and “t” respectively as major function of program and those are set to increase and decrease, according to the number of working threads. These threads will work constantly and will surely access their allocated memory. Every working thread purely loops though the allocated area, one action at a time like reading data then performs a simple arithmetic operation and then it writes back the result.

4.3 Test 3: Disk I/O; LVM stripes and caches

4.3.1 Motivation

The Logical Volume Manager is the process to improve the disk performance on the Linux systems. If we explain in simple terms, Logical Volume Manager can be produced with virtual disk from various physical storage disks and spread data files on all physical disks in so called stripes. In price of small bookkeeping overhead which mainly allows the system to utilize various disks and boost performance.

This test is intended to demonstrate the effect of Logical Volume Manager on the z mainframe system. Optimizing and Monitoring of disk performance on a Linux z/VM guest can however be a very complex task which is actually complicated by the many levels of buffers and caches. Linux have z/VM which uses minidisk caches, disk control unit that contains huge caches and file buffers in memory. The test is intended to demonstrate the effect of buffers, caches, and LVM stripes.

4.3.2 Measuring methods and tools

4.3.2.1 Test tools: Bonnie

Basically, this test is trust on open source, old, disk benchmarking tool called Bonnie that conveniently incorporated in SLES9 distribution. Bonnie used the techniques to measure performance of the UNIX file system operations. It creates the file of certain size and
used diverse methods to access their file while assessing performance [79]. Here I have quoted an instance of Bonnie output:

```
z6qku@linx03:/database/test03> bonnie -s 512
Bonnie: Warning: You have 996MB RAM, but you test with only 512MB datasize!
Bonnie: This might yield unrealistically good results,
Bonnie: for reading and seeking and writing.
Bonnie 1.4: File './Bonnie.10395', size: 536870912, volumes: 1
Writing with putc()... done: 15224 kB/s 99.1 %CPU
Rewriting... done: 32375 kB/s 11.9 %CPU
Writing intelligently... done: 28996 kB/s 18.7 %CPU
Reading with getc()... done: 14815 kB/s 96.4 %CPU
Reading intelligently... done: 470583 kB/s 88.9 %CPU
Seeker 1...Seeker 2...Seeker 3...start 'em...done...done...done...
---Sequential Output (nosync)--- ---Sequential Input-- --Rnd Seek- 
-Per Char- --Block--- -Rewrite-- -Per Char- --Block--- --04k (03)-
Machine MB K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU /sec %CPU
```

### 4.3.2.2 Test tools: useMem

The useMem program from test 2 can be used here again to put their Linux file cache out of contest, actually this will suit are requirements. It can be accomplished by permitting the useMem program to consume all of the available memory, and certainly leave no room for file caching. In order to exert, the useMem program can be executed/started with high priority than the normal programs using nice command.

```
nice -n -1 ./uM xxx 0 (as root)
```

This is very essential to instruct the Linux to minimize its tendency to swap. This can be accomplished by setting up echo 0 > /proc/sys/vm/swappiness

```
useMem.c
```

```c
// --------------------------------------------------------------------------------------------------
// * useMem.c - Memory hungry program e.g. for tests of swapping
// *
// * Startup argument: Memory to occupy from beginning in MB
// --------------------------------------------------------------------------------------------------
#include <stdio.h> // I/O...
#include <pthread.h> // Thread support
#include <time.h> // Nanosleep
#include <stdlib.h> // rand()
#include <errno.h> // evaluate errors from nanosleep
#include <sys/time.h> // for writeTime() including milli sec
```
#include <time.h> // for writeDate()
#define MAX_THREAD 50
#define MAX_MEM (RAND_MAX)
#define MAX_MEM_MB (RAND_MAX/(1024*1024))
#define TOINC 0x04000000
#define _MULTI_THREADED

const unsigned int kb128iMax = 128*1024/sizeof(unsigned long int);

// Prototyping
void memChange(int);
void *worker(void *);
void *monitor(void *arg);
void threadChange(int);
void writeTime();
void writeDate();

// User defined types
typedef struct {
  int id;
  unsigned long int kb128count;
  unsigned int stop;
} workerParm;

// Global variables
unsigned int currentMemSize = 0x04000000; // 64MB
unsigned int numActiveThreads = 1;
unsigned char wait = 0;
unsigned char stop = 0;
unsigned char flagToStop = 0;
unsigned char threadsWaiting = 0;
unsigned long int *memChunkPtr;
unsigned long int oldValues[MAX_THREAD];
struct timespec interval;

// Working threads vars:
pthread_t *threads;
pthread_t monitorThread;
workerParm *parms;

// Sync related
pthread_mutex_t count_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t count_threshold_cv = PTHREAD_COND_INITIALIZER;
pthread_mutex_t canRunAgain_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t canRunAgain_cv = PTHREAD_COND_INITIALIZER;

// Main function
int main(int argc, char* argv[]) {
  unsigned int i;
  // Initialization
  srand(5); // Seeding random number generator
  threads=(pthread_t*)malloc(MAX_THREAD*sizeof(*threads));
  parms=(workerParm *)malloc(sizeof(workerParm)*MAX_THREAD);
Chapter 4: Proposed Methodology

interval.tv_sec = 0;
interval.tv_nsec = 0; // 1000; // (long)(100*1e+6);
// "Welcome message"
fprintf(stderr, "Maximum memory usage: %dMB\n", MAX_MEM_MB);
// Change memory size to occupy from start if argument is given
if (argc > 1) {
    unsigned int tmp;
    if ( sscanf(argv[1], "%d", &tmp) == 1 )
        if (tmp*1024 <= RAND_MAX/1024)
            currentMemSize = 1024*1024*tmp; // from MB to bytes
    fprintf(stderr, "Memory usage set to: %d bytes (%d MB)\n", currentMemSize, currentMemSize/(1024*1024));
} else {
    fprintf(stderr, "Argument not accepted.\n");
    exit(1);
}
}
if (argc == 3) {
    if ( sscanf(argv[2], "%d", &tmp) == 1 )
        if (tmp <= MAX_THREAD)
            numActiveThreads = tmp;
    else {
        fprintf(stderr, "Argument 2 (number of threads) not accepted, should be < %d.\n", MAX_THREAD);
        exit(1);
    }
}
else {
    fprintf(stderr, "Memory usage defaults to %d MB and number of threads to %d\n", currentMemSize/(1024*1024), numActiveThreads);
}
mempChunkPtr = (unsigned long int *)malloc( currentMemSize );
if (mempChunkPtr == NULL) {
    fprintf(stderr, "Malloc failed, exiting...\n");
    exit(2);
}
memset( memChunkPtr, '\xAA', currentMemSize );
fprintf(stderr, "Memory initialized\n");
// Output Descriptive text and running date
writeDate();
printf ("time, elapsed time, threads, memory usage, throughput (kb)\n");
// Creation of worker thread
for (i=0; i < numActiveThreads ; i++) {
    int ret;
    parms[i].id = i;
    parms[i].stop = 0;
    parms[i].kb128count = 0;
    ret = pthread_create(&threads[i], NULL, worker, (void *)(parms+i));
    if ( ret==0 ) {
        fprintf(stderr, "Thread %d created successfully\n", i);
    } else {
        fprintf(stderr, "Thread %d NOT created\n", i);
    }
Chapter 4: Proposed Methodology

void memChange(int cmd){
    long int newMemSize=0;
    // threadsWaiting = 0; // Reset number of waiting threads (from last run)
    flagToStop = 1; // Signal threads to sleep as they reach end of loop
    // Wait for all active threads sleeping...
    pthread_mutex_lock(&count_mutex);
    while (threadsWaiting < numActiveThreads) {
        // pthread releases mutex while waiting...
        pthread_cond_wait(&count_threshold_cv, &count_mutex);
    }
    pthread_mutex_unlock(&count_mutex);
}
if (cmd == '-') {
    // Decrease memory size
    newMemSize = currentMemSize - TOINC;
    if (newMemSize <= TOINC) {  // inc/dec block size
        fprintf(stderr, "New Value below %d MB\n", TOINC/1024/1024);
        newMemSize = 0;
    } else {
        // Increase memory size
        newMemSize = currentMemSize + TOINC;
        if (newMemSize > MAX_MEM) {
            fprintf(stderr, "New Value limited reached (%d MB)\n", MAX_MEM_MB);
            newMemSize = 0;
        }
    }
    // Resize memory chunk
    if (newMemSize > 0) {
        memChunkPtr = (unsigned long int *) realloc(memChunkPtr, newMemSize);
        currentMemSize = newMemSize;
        if (cmd == '+') {
            memset((unsigned char*)memChunkPtr + currentMemSize - TOINC, '\xAA', TOINC);
            fprintf(stderr, "New memory initialized. ");
        }
        fprintf(stderr, "Memory usage: %d MB\n", currentMemSize/1024/1024);
    }
    // Signal all threads to start again... (safely)
    pthread_mutex_lock(&canRunAgain_mutex);
    flagToStop = 0;
    pthread_cond_broadcast(&canRunAgain_cv);
    pthread_mutex_unlock(&canRunAgain_mutex);
    // Wait for all active threads is awake again, since this function
    // should not be invoked again before all working threads is awake
    pthread_mutex_lock(&count_mutex);
    while (threadsWaiting > 0) {
        pthread_mutex_unlock(&count_mutex);
        pthread_cond_wait(&count_threshold_cv, &count_mutex);
    }
    pthread_mutex_unlock(&count_mutex);
}
// Function: threadChange - Increase / decrease number of worker threads
// input: '+': increase memory
//        '-': decrease memory
void threadChange(int cmd) {
    if (cmd == 't') {
        // Create a thread
        if (numActiveThreads < MAX_THREAD) {
            int ret;
            int i = numActiveThreads;  // actual count equals new array index
            parms[i].id = i;
            parms[i].stop = 0;
            parms[i].kb128count = 0;
            oldValues[i] = 0;
        } else {
            // Increase memory size
            newMemSize = currentMemSize + TOINC;
            if (newMemSize > MAX_MEM) {
                fprintf(stderr, "New Value limited reached (%d MB)\n", MAX_MEM_MB);
                newMemSize = 0;
            }
        }
    } else {
        // Decrease memory size
        newMemSize = currentMemSize - TOINC;
        if (newMemSize <= TOINC) {  // inc/dec block size
            fprintf(stderr, "New Value below %d MB\n", TOINC/1024/1024);
            newMemSize = 0;
        } else {
            // Increase memory size
            newMemSize = currentMemSize + TOINC;
            if (newMemSize > MAX_MEM) {
                fprintf(stderr, "New Value limited reached (%d MB)\n", MAX_MEM_MB);
                newMemSize = 0;
            }
        }
        // Resize memory chunk
        if (newMemSize > 0) {
            memChunkPtr = (unsigned long int *) realloc(memChunkPtr, newMemSize);
            currentMemSize = newMemSize;
            if (cmd == '+') {
                memset((unsigned char*)memChunkPtr + currentMemSize - TOINC, '\xAA', TOINC);
                fprintf(stderr, "New memory initialized. ");
            }
            fprintf(stderr, "Memory usage: %d MB\n", currentMemSize/1024/1024);
        }
        // Signal all threads to start again... (safely)
        pthread_mutex_lock(&canRunAgain_mutex);
        flagToStop = 0;
        pthread_cond_broadcast(&canRunAgain_cv);
        pthread_mutex_unlock(&canRunAgain_mutex);
        // Wait for all active threads is awake again, since this function
        // should not be invoked again before all working threads is awake
        pthread_mutex_lock(&count_mutex);
        while (threadsWaiting > 0) {
            pthread_mutex_unlock(&count_mutex);
            pthread_cond_wait(&count_threshold_cv, &count_mutex);
        }
        pthread_mutex_unlock(&count_mutex);
    }
}
Chapter 4: Proposed Methodology

ret = pthread_create(&threads[i], NULL, worker, (void *)(parms+i));
if (ret == 0) {
    numActiveThreads++;
    fprintf(stderr, "%d thread(s) running\n", numActiveThreads);
} else {
    fprintf(stderr, "Thread %d NOT created\n", i);
}
} else {
    printf("Maximum number of threads reached (%d)\n", MAX_THREAD);
}
} else {
    // stop a thread
    if (numActiveThreads > 0) {
        parms[numActiveThreads-1].stop = 1;
        pthread_join(threads[numActiveThreads-1],NULL); // Await thread stopping
        numActiveThreads--;
        fprintf(stderr, "%d thread(s) running\n", numActiveThreads);
    } else {
        printf("All threads already stopped\n");
    }
}

// Function: Worker thread - continuously writes into memory chunk
void *worker(void *arg) {
    workerParm *p=(workerParm *)arg;
    unsigned int actualIndex;
    unsigned kb128LocalCount = 0;
    struct timespec rest = 0;
    // Find a random place in memory chunk where to begin iterations
    do {
        actualIndex = rand();
    } while (actualIndex >= currentMemSize/sizeof(unsigned long int));
    fprintf(stderr, "Worker thread %d has started with memory chunk index %d \n", p->id, actualIndex);
    // Repeat until thread is stopped...
    while (p->stop == 0 && stop == 0) {
        // Increment iterator and fix potential overflow
        actualIndex++;
        if (actualIndex >= currentMemSize/sizeof(unsigned long int))
            actualIndex = 0;
        // Modify data in memory to ensure memory page in memory
        memChunkPtr[actualIndex] += kb128LocalCount;
        kb128LocalCount++;
        if (kb128LocalCount == kb128iMax) {
            kb128LocalCount = 0;
            p->kb128count++;
        }
        // Wait according to actual "interval"-interruption by a non-blocked
        // signal is handled
        while (wait && nanosleep(&interval, &rest) == -1) {

if (errno != EINTR) {
  printf("problem calling nanosleep in thread %d\n", p->id);
  exit(3);
}

// Going to sleep if "main-thread" has flagged to do...
if (flagToStop > 0) {
  // Signal main thread that thread is going to wait (safe manner)
  pthread_mutex_lock(&count_mutex);
  threadsWaiting ++;
  pthread_cond_signal(&count_threshold_cv);
  pthread_mutex_unlock(&count_mutex);
  //wait for main has finished / flagToStop turn 0 (safe manner)
  pthread_mutex_lock(&canRunAgain_mutex);
  while (flagToStop > 0) {
    //pthread releases mutex while waiting...
    pthread_cond_wait(&canRunAgain_cv, &canRunAgain_mutex);
  }
  pthread_mutex_unlock(&canRunAgain_mutex);
  // Signal main thread that this thread is running again (safe manner)
  pthread_mutex_lock(&count_mutex);
  threadsWaiting --;
  pthread_cond_signal(&count_threshold_cv);
  pthread_mutex_unlock(&count_mutex);
}

fprintf(stderr, "Workerthread %d stopping\n", p->id);
pthread_exit(NULL);

// Function: Monitor thread -
// calculates number of bytes processes by worker threads
// Runs approximately 1 time pr. sec.
// ________________________________________________________________

void *monitor(void *arg) {
  unsigned long int sinceLast;
  unsigned int i;
  struct timespec interval;
  struct timespec rest;
  interval.tv_sec = 1;
  interval.tv_nsec = 0;
  // Init oldValues array
  for (i=0; i<MAX_THREAD; i++)
    oldValues[i] = 0;
  // Keep running until program is stopped
  while (stop == 0) {
    sinceLast=0; // reset counter
    // Since parms[i] is only updated by the threads, this is reasonably safe
    for (i=0; i<numActiveThreads; i++) {
      sinceLast += parms[i].kb128count - oldValues[i] ;
      oldValues[i]=parms[i].kb128count;
    }
    writeTime();
    printf("%d, %d, %d \n", numActiveThreads, currentMemSize/1024/1024, (sinceLast/8));
  }
}
// Wait according to actual "interval" - interruption by a non-blocked
// signal is handeld
while ( nanosleep(&interval, &rest) == -1) {
    if (errno != EINTR) {
        printf("problem calling nanosleep from monitor thread\n");
        exit(4);
    }
    pthread_exit(NULL);
}

// Function: writeTime -
// Writes current time and program elapsed in format:
// "hh:mm:ss , elapSec.elapMilliSec ,"  

void writeTime() {
    static time_t startSec = 0;
    struct timeval tv;
    struct timezone tz;
    struct tm *tm;
    gettimeofday(&tv, &tz);
    tm=localtime(&tv.tv_sec);
    if (startSec == 0)
        startSec = tv.tv_sec;
    printf("%d:%02d:%02d , %d.%d ,", tm->tm_hour, tm->tm_min, tm->tm_sec,
        tv.tv_sec-startSec, tv.tv_usec/1000);
}

void writeDate() {
    time_t rawtime;
    struct tm *timeinfo;
    time( &rawtime );
    timeinfo = localtime ( &rawtime );
    printf("Test date: %s", asctime (timeinfo));
}

Makefile for useMem (uM)
CC = gcc
LLIBS = pthread
all: useMem.c
$(CC) useMem.c -lpthread -o uM -O3 -march=z900 -funroll-loops

qMeterGUI.java
import info.monitorenter.gui.chart.Chart2D;
import info.monitorenter.gui.chart.rangepolicies.RangePolicyForcedPoint;
import info.monitorenter.gui.chart.traces.Trace2DXspan;
import java.awt.Color;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Vector;
import javax.swing.JLabel;
import javax.swing.JTextField;

/*
 * qMeterGUI.java
 * Created on 23 May 2007, 14:54
*/
public class qMeterGUI extends javax.swing.JFrame {
/**
 * Creates new form qMeterGUI
 */
public qMeterGUI() {
initComponents();
// Create an ITrace:
// Note that dynamic charts need limited amount of values!!!
chartA.getAxisY().setRangePolicy(new RangePolicyForcedPoint());
meterThreadA = new dataGatherThread(chartA, textPortA);
}
/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
// <editor-fold defaultstate="collapsed" desc=" Generated Code ">
private void initComponents() {
JLabel1 = new javax.swing.JLabel();
textIPaddress = new javax.swing.JTextField();
buttonStart = new javax.swing.JButton();
buttonStop = new javax.swing.JButton();
buttonReset = new javax.swing.JButton();
jLabel4 = new javax.swing.JLabel();
textPortA = new javax.swing.JTextField();
chartA = new info.monitorenter.gui.chart.Chart2D();
jLabel6 = new javax.swing.JLabel();
textXspan = new javax.swing.JTextField();
textLogFile = new javax.swing.JTextField();
buttonLog = new javax.swing.JButton();
textLog = new javax.swing.JTextField();
setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setTitle("memMeter");
jLabel1.setText("IP address:");
textPaddress.setText("172.31.218.12");
textPaddress.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent evt) {
textContentActionPerformed(evt);
}
});
buttonStart.setText("Start");
buttonStart.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent evt) {
buttonStartActionPerformed(evt);
}
});
buttonStop.setText("Stop");
buttonStop.setEnabled(false);
}
buttonStop.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        buttonStopActionPerformed(evt);
    }
});
buttonReset.setText("Reset graphs");
buttonReset.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        buttonResetActionPerformed(evt);
    }
});
jLabel4.setText("Port A");
textPortA.setText("5557");
textPortA.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        textPortAActionPerformed(evt);
    }
});
javax.swing.GroupLayout chartALayout = new javax.swing.GroupLayout(chartA);
chartA.setLayout(chartALayout);
chartALayout.setHorizontalGroup(
    chartALayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGap(0, 768, Short.MAX_VALUE)
    .addContainerGap()
);
chartALayout.setVerticalGroup(
    chartALayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGap(0, 425, Short.MAX_VALUE)
    .addContainerGap()
);
jLabel6.setText("x span: ");
textXspan.setText("30");
textLogFile.setText("c:\qku\data\log.csv");
buttonLog.setText("Log");
buttonLog.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        buttonLogActionPerformed(evt);
    }
});
javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGap(0, java.awt.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
    .addContainerGap()
);
layout.createSequentialGroup().addGap(0, 94, Short.MAX_VALUE)
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGap(0, java.awt.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
    .addComponent(textIPaddress, javax.swing.GroupLayout.DEFAULT_SIZE, 94, Short.MAX_VALUE)
).
.addPreferredGap(java.awt.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(textIPaddress, javax.swing.GroupLayout.DEFAULT_SIZE, 94, Short.MAX_VALUE)
).
.addPreferredGap(java.awt.LayoutStyle.ComponentPlacement.RELATED)
Chapter 4: Proposed Methodology

```java
.addComponent(jLabel4)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addGap(57, 57)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addComponent(buttonLog)
createSequentialGroup()
.addComponent(buttonStart)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addComponent(buttonStop)
.addComponent(jLabel6)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addComponent(buttonReset))
.addContainerGap()
);
layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
addGroup(layout.createSequentialGroup()
.addComponent(jLabel1)
.addComponent(buttonLog)
.addComponent(jLabel4)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addComponent(buttonStop)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addComponent(buttonStart)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addComponent(buttonReset)
.addComponent(jLabel6)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addPreferredGap(javax.swing.GroupLayout.Alignment.RELATED)
.addComponent(buttonReset))
.addContainerGap()
);
```

pack();

private void buttonLogActionPerformed(java.awt.event.ActionEvent evt) {
    if (bw != null) {
        try {
            bw.newLine();
            bw.write(textLog.getText());
        } catch (IOException ignore) {
            System.out.println("IOException writing log text");
        }
    } else {
        try {
            BufferedWriter bw = new BufferedWriter(
                new FileWriter(textLogFile.getText(),true));
            bw.newLine();
            bw.write(textLog.getText());
            bw.close();
        } catch (IOException ex) {
            System.out.println("IOException writing log text (file not already open)\n" + ex);
        }
    }
}

private void textPortAActionPerformed(java.awt.event.ActionEvent evt) {
}

private void buttonResetActionPerformed(java.awt.event.ActionEvent evt) {
    oRunTrace.removeAllPoints();
   oSleTrace.removeAllPoints();
}

private void buttonStopActionPerformed(java.awt.event.ActionEvent evt) {
    buttonStart.setEnabled(true);
    buttonStop.setEnabled(false);
    meterThreadA.wakeup();
}

private void textIPaddressActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new qMeterGUI().setVisible(true);
        }
    });
    // Variables declaration - do not modify
    private javax.swing.JButton buttonLog;
    private javax.swing.JButton buttonReset;
    private javax.swing.JButton buttonStart;
    private javax.swing.JButton buttonStop;
private info.monitorenter.gui.chart.Chart2D chartA;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel6;
private javax.swing.JTextField ipAddress;
private javax.swing.JTextField textLog;
private javax.swing.JTextField textLogFile;
private javax.swing.JTextField textPortA;
private javax.swing.JTextField textXspan;

// End of variables declaration

dataGatherThread meterThreadA = null;
dataGatherThread meterThreadB = null;
Trace2DXspan oRunTrace = null;
Trace2DXspan oSlTrace = null;
Double xFirst = Double.MAX_VALUE;
BufferedWriter bw = null;
class dataGatherThread extends Thread {
Vector<Trace2DXspan> traces = new Vector<Trace2DXspan>();
JTextField textPort;
JLabel labelCurrentVal;
public Trace2DXspan createTrace(int parmnum, Color c, String name, String unit1,
String unit2) {
Trace2DXspan trace = new Trace2DXspan();
trace.setColor(c);
trace.setName(name);
trace.setPhysicalUnits(unit1, unit2);
if (traces.size() < parmnum+1) {
traces.setSize(parmnum+5);
}
traces.setElementAt(trace, parmnum);
return trace;
}
public dataGatherThread(Chart2D chart, JTextField p) {
traces.setSize(20);
chart.addTrace(createTrace(2, Color.GREEN, "app", "", ""));
chart.addTrace(createTrace(3, Color.PINK, "slab", "", ""));
chart.addTrace(createTrace(4, Color.BLUE, "PageTables", "", ""));
chart.addTrace(createTrace(5, Color.CYAN, "vmallocUsed", "", ""));
chart.addTrace(createTrace(6, Color.YELLOW, "Buffers", "", ""));
chart.addTrace(createTrace(7, Color.ORANGE, "Cache", "", ""));
chart.addTrace(createTrace(8, Color.BLACK, "MemFree", "", ""));
chart.addTrace(createTrace(9, Color.RED, "SwapCached", "", ""));
chart.addTrace(createTrace(10, Color.LIGHT_GRAY, "Swap", "", ""));
textPort = p;
start();
}
public synchronized void wakeup() {
this.notify();
}
public synchronized void run() {
//System.out.println("Thread is living!!!!!!!!");
while(true) {
}
Socket clientSocket = null;
DataOutputStream os = null;
BufferedReader br = null;
while (buttonStop.isEnabled() == false) {
    try {
        //System.out.println("Thread is going to wait...");
        this.wait();
        //System.out.println("Thread woke up...");
    } catch (InterruptedException ignore) {} 
}
String ipadd = textIPaddress.getText();
String toSend = new String("m\n");
for (Trace2DXspan t : traces) {
    if (t != null) {
        t.setXSpan(new Integer(textXspan.getText()));
    }
}
try {
    clientSocket = new Socket(ipadd, new Integer(textPort.getText()));
    os = new DataOutputStream(clientSocket.getOutputStream());
    br = new BufferedReader(
        new InputStreamReader(clientSocket.getInputStream()));
} catch (UnknownHostException e) {
    System.err.println("Don't know about host: hostname");
    buttonStopActionPerformed(null);
} catch (IOException e) {
    System.err.println("Failed creating sockets and streams");
    buttonStopActionPerformed(null);
}
try {
    bw = new BufferedWriter(
        new FileWriter(textLogFile.getText(),true));
} catch (IOException ex) {
    System.err.println("Problem opening log file");
    buttonStopActionPerformed(null);
}
try {
    os.writeBytes("mi\n");
    bw.write("\n" + br.readLine() + "\n");
    bw.write(br.readLine());
} catch (IOException ex) {
    System.err.println("Logging date and descriptive text failed");
    buttonStopActionPerformed(null);
}
while (buttonStop.isEnabled()) {
    if (clientSocket != null & & os != null & & br != null) {
        try {
            //System.out.print("Now writing: " + toSend);
            os.writeBytes(toSend); 
            //System.out.println("Now reading...");
            String responseLine = br.readLine();
            if (responseLine != null) {
                //System.out.println(responseLine);
                String[] responseValues = responseLine.split(",");
                if (responseValues.length > 1) {
bw.newLine();
bw.write(responseLine);
Double xSinceEpoc = new Double(responseValues[1]);
if (xSinceEpoc < xFirst) xFirst = xSinceEpoc;
xSinceEpoc = xSinceEpoc - xFirst;
Double y = new Double(0);
for (int i = 2; i < responseValues.length; i++) {
y += new Double(responseValues[i])/1024;
if (traces.get(i) != null) {
    traces.get(i).addPoint(xSinceEpoc, y);
}
// Ugly fix to get swapCached shown twice
if (i == 9) {
y += new Double(responseValues[9])/1024;
}
if (traces.get(20) != null) {
    traces.get(20).addPoint(xSinceEpoc, y);
}
try {
    Thread.sleep(400);
} catch (InterruptedException ignore) {} else {
    //System.out.println("Null!! :-(");;
}
} catch (UnknownHostException e) {
    System.err.println("Trying to connect to unknown host: "+ e);
} catch (IOException e) {
    System.err.println("IOException: "+ e);
    break;
}
} try {
    os.writeBytes("q\n");
    os.close();
    br.close();
    clientSocket.close();
    bw.close();
    bw = null;
} catch (IOException e) {
    System.err.println("IOException (closing): "+ e);
}
*/
* Trace2DXspan is a modification of Trace2DLtd modified
* This modified version makes is "fix" the x-axis to a certain "span" width.
* Points with x < (newest x - span) are discarded.

104
package info.monitorenter.gui.chart.traces;
import info.monitorenter.gui.chart.Chart2D;
import info.monitorenter.gui.chart.ITrace2D;
import info.monitorenter.gui.chart.TracePoint2D;
import info.monitorenter.util.collections.IRingBuffer;
import info.monitorenter.util.collections.RingBufferArrayFast;
import java.util.Iterator;

/**
 * Additional to the Trace2DSimple the Trace2DLimited adds the following functionality:
 * <ul>
 * <li>The amount of internal tracepoints is limited to the maxsize, passed to the constructor.</li>
 * <li>If a new tracepoint is inserted and the maxsize has been reached, the tracepoint residing for the longest time in this trace is thrown away.</li>
 * </ul>
 * Take this implementation to display frequently changing data (nonstatic, time - dependant values). You will avoid a huge growing amount of tracepoints that would increase the time for scaling and painting until system hangs or java.lang.OutOfMemoryError is thrown.
 */

public class Trace2DXspan extends ATrace2D implements ITrace2D {

/**
 * Internal fast fifo buffer implementation based upon indexed access to an array.
 */

protected IRingBuffer m_buffer;
private Integer xSpan;

/**
 * Defcon of this stateless instance.
 */

public Trace2DXspan() {
this(new Integer(600), new Integer(30), Trace2DLtd.class.getName() + "," + getInstanceCount());
}

public Trace2DXspan(final int maxsize, final int span) {
this(maxsize, span , Trace2DLtd.class.getName() + "," + getInstanceCount());
}

/**
 * Constructs an instance with a buffersize of maxsize and a default name.
 */

public Trace2DXspan(final int maxsize, final int span, final String name) {
this.m_buffer = new RingBufferArrayFast(maxsize);
}
Chapter 4: Proposed Methodology

```java
xSpan = new Integer(span);
//xMax = new Integer(span);
this.setName(name);
}
/**
 * @see ATrace2D#addPointInternal(info.monitorenter.gui.chart.TracePoint2D)
 */
public boolean addPointInternal(final TracePoint2D p) {
    TracePoint2D removed = (TracePoint2D) this.m_buffer.add(p);
    if (removed == null) {
        TracePoint2D oldest = (TracePoint2D)this.m_buffer.getOldest();
        if (oldest.getX() < ((int)p.getX()) - xSpan) {
            removed = (TracePoint2D) this.m_buffer.remove();
        } else {
            // no point was removed
            // use bound checks of calling addPoint
            return true;
        }
    }
    while (removed != null) {
        double tmpx;
        double tmpy;
        tmpy = removed.getY();
        if (tmpy >= this.m_maxY) {
            tmpy = this.m_maxY;
            this.maxYSearch();
            this.firePropertyChange(PROPERTY_MAX_Y, new Double(tmpy),
                    new Double(this.m_maxY));
        } else if (tmpy <= this.m_minY) {
            tmpy = this.m_minY;
            this.minYSearch();
            this.firePropertyChange(PROPERTY_MIN_Y, new Double(tmpy),
                    new Double(this.m_minY));
        }
        // scale the new point, check for new bounds!
        this.firePointAdded(p);
        removed = null;
        TracePoint2D oldest = (TracePoint2D)this.m_buffer.getOldest();
        if (oldest.getX() < ((int)p.getX()) - xSpan) {
            removed = (TracePoint2D) this.m_buffer.remove();
        }
        // scale the new point, check for new bounds!
    }
    return false;
}
/**
 * @see info.monitorenter.gui.chart.ITrace2D#getMaxSize()
 */
public int getMaxSize() {
    return this.m_buffer.getBufferSize();
}
/**
 * Returns the acutal amount of points in this trace.
 * <p>
 */
```
* @return the actual amount of points in this trace.
* @see info.monitorenter.gui.chart.ITrace2D#getSize()
* /
public int getSize() {
    return this.m_buffer.size();
} /**
* @see info.monitorenter.gui.chart.ITrace2D#isEmpty()
* /
public boolean isEmpty() {
    return this.m_buffer.isEmpty();
} /**
* @see info.monitorenter.gui.chart.ITrace2D#iterator()
* /
public Iterator iterator() {
    if (Chart2D.DEBUG_THREADING) {
        System.out.println("Trace2DXspan.iterator, 0 locks");
    }
    synchronized (this.m_renderer) {
        if (Chart2D.DEBUG_THREADING) {
            System.out.println("Trace2DXspan(iterator, 1 lock");
        }
        synchronized (this) {
            if (Chart2D.DEBUG_THREADING) {
                System.out.println("Trace2DXspan.iterator, 2 locks");
            }
            return this.m_buffer.iteratorL2F();
        }
    }
} /**
* @see info.monitorenter.gui.chart.ITrace2D#removeAllPoints()
* /
public void removeAllPointsInternal() {
    this.m_buffer.clear();
} /**
* <p>
* Returns false always because internally a ringbuffer is used which does not
* allow removing of values because that would break the contract of a
* ringbuffer.
* </p>
* @param point
* the point to remove.
* *
* @return false always because internally a ringbuffer is used which does not
* allow removing of values because that would break the contract of a
* ringbuffer.
* /
protected boolean removePointInternal(final TracePoint2D point) {
    return false;
} /**
* Sets the maximum amount of points that may be displayed.
* <p>
* 
* Don't use this too often as decreases in size may cause expensive array
* copy operations and new searches on all points for bound changes.
* <p>
* 
* TODO: Only search for bounds if size is smaller than before, debug and
* test.
* 
* @param amount
* the new maximum amount of points to show.
* */

```
public final void setMaxSize(final int amount) {
if (Chart2D.DEBUG_THREADING) {
    System.out.println("Trace2DXspan.setMaxSize, 0 locks");
}
synchronized (this.m_renderer) {
    if (Chart2D.DEBUG_THREADING) {
        System.out.println("Trace2DXspan.setMaxSize, 1 lock");
    }
synchronized (this) {
        if (Chart2D.DEBUG_THREADING) {
            System.out.println("Trace2DXspan.setMaxSize, 2 locks");
        }
        this.m_buffer.setBufferSize(amount);
        double xmin = this.m_minX;
        this.minXSearch();
        if (this.m_minX != xmin) {
            this.firePropertyChange(PROPERTY_MIN_X, new Double(xmin),
                                    new Double(this.m_minX));
        }
        double xmax = this.m_maxX;
        this.maxXSearch();
        if (this.m_maxX != xmax) {
            this.firePropertyChange(PROPERTY_MAX_X, new Double(xmax),
                                    new Double(this.m_maxX));
        }
        double ymax = this.m maxY;
        this.maxYSearch();
        if (this.m maxY != ymax) {
            this.firePropertyChange(PROPERTY_MAX_Y, new Double(ymax),
                                    new Double(this.m maxY));
        }
        double ymin = this.m minY;
        this.minYSearch();
        if (this.m minY != ymin) {
            this.firePropertyChange(PROPERTY_MIN_Y, new Double(ymin),
                                    new Double(this.m minY));
        }
    }
}
```
Chapter 4: Proposed Methodology

- Method triggered by
  - ` {@link TracePoint2D#setLocation(double, double)}`
  - ` {@link #addPoint(TracePoint2D)}` or
  - ` {@link #removePoint(TracePoint2D)}`

- Bound checks are performed and property change events for the properties
  - `@link ITrace2D#PROPERTY_MAX_X`,
  - `@link ITrace2D#PROPERTY_MIN_X`,
  - `@link ITrace2D#PROPERTY_MAX_Y` and
  - `@link ITrace2D#PROPERTY_MIN_Y` are fired if the add bounds have changed due to the modification of the point.

- @param changed
  - the point that has been changed which may be a newly added point
  - (from ` {@link #addPoint(TracePoint2D)}`), a removed
  - one or a modified one.
  - @param added
  - if true the points values dominate old bounds, if false the bounds are rechecked against the removed points values.

```java
public void firePointChanged(final TracePoint2D changed, final boolean added) {
    double tmpx = changed.getX();
    double tmpy = changed.getY();
    if (added) {
        if ((int)tmpx) > this.m_maxX ) {
            this.m_maxX = (int)tmpx;
            //this.expandMaxXErrorBarBounds();
            this.firePropertyChange(PROPERTY_MAX_X, null, new Double(this.m_maxX +1));
            this.m_minX = this.m_maxX-xSpan;
            this.firePropertyChange(PROPERTY_MIN_X, null, new Double(this.m_minX));
        }
        if (tmpy > this.m_maxY) {
            this.m_maxY = tmpy;
            //this.expandMaxYErrorBarBounds();
            this.firePropertyChange(PROPERTY_MAX_Y, null, new Double(this.m_maxY));
        } else if (tmpy < this.m_minY) {
            this.m_minY = tmpy;
            //this.expandMinYErrorBarBounds();
            this.firePropertyChange(PROPERTY_MIN_Y, null, new Double(this.m_minY));
        }
    } else {
        if (tmpy >= this.m_maxY) {
            tmpy = this.m_maxY;
            this.maxYSearch();
            this.firePropertyChange(PROPERTY_MAX_Y, new Double(tmpy),
            new Double(this.m_maxY));
        } else if (tmpy <= this.m_minY) {
            tmpy = this.m_minY;
            this.minYSearch();
            this.firePropertyChange(PROPERTY_MIN_Y, new Double(tmpy),
```
new Double(this.m_minY));
}  
if (this.getSize() == 0) {
    //this.m_firsttime = true;
}
}
public Integer getXSpan() {
    return xSpan;
}
public void setXSpan(Integer xSpan) {
    this.xSpan = xSpan;
}

4.3.2.3 Data gathering

The test trust on performance determined by the tool Bonnie that will be sufficient to demonstrate the effect of buffers, caches and LVM stripes. The qMonitor storage monitors from the above test 2 shall be reapplied to expose the potential of file caching in memory.

4.3.3 Test description

The inspiration is to apply swappiness attribute, useMem, and priorities will be to stay away from memory file caching which can be determined. The effect of test has been confirmed in above first three test steps (from T3.1 to T3.3). To control the file caching Bonnie can be used under the below defined eight combination of:

- Disabled/ Enabled minidisk cache
- Disabled/Enabled Control Unit Cache
- Run on the non-LVM striped based disk / on LVM striped disk.

The above combinations are tested with Bonnie three times for each step in test matrix.
Chapter 4: Proposed Methodology

Table 4-2: Test matrixes are giving test combinations Swappiness can be set by echo the specified value to attribute:

<table>
<thead>
<tr>
<th>Test number</th>
<th>Test matrix</th>
<th>Swappiness</th>
<th>Mindisk</th>
<th>LVM stripes</th>
<th>CU cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>T3.1</td>
<td></td>
<td>60</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>T3.2</td>
<td>x</td>
<td>60</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>T3.3</td>
<td>x</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>T3.4</td>
<td>x</td>
<td>0</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>T3.5</td>
<td>x</td>
<td>0</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>T3.6</td>
<td>x</td>
<td>0</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>T3.7</td>
<td>x</td>
<td>0</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>T3.8</td>
<td>x</td>
<td>0</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>T3.9</td>
<td>x</td>
<td>0</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>T3.10</td>
<td>x</td>
<td>0</td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 4-2: Test matrixes are giving test combinations Swappiness can be set by echo the specified value to attribute:

Now we will set Swappiness attribute with certain values:

```
echo 0 > /proc/sys/vm/swappiness
```

```
echo 60 > /proc/sys/vm/swappiness
```

Also we will turn on and off the Minidisk cache for specific range of minidisk in a specific virtual machine (LINX03), privileged guest will issue the below commands:

```
SET MDCACHE MDISK OFF USERID LINX03 0100-0110 DIR
```

```
SET MDCACHE MDISK ON USERID LINX03 0100-0110 DIR
```

Control Unit Cache will be inquired and controlled using tune dasd for each linux disk device using tunedasd via following command

```
tunedasd -c bypass /dev/dasdf
```

Setting cache mode for device </dev/dasdf>...

```
tunedasd -c normal -n 2 /dev/dasdf
```

Setting cache mode for device </dev/dasdf>...

```
tunedasd -g /dev/dasdf
```

normal (2 cyl)

4.4 Test 4: z/VM in z/VM penalty

4.4.1 Motivation
The mainframe hardware system can easily support to virtualization up to two levels with no performance degradation. While the logical partition is offered by the PR/SM (Processor Resource/System Manager) (EAL5 - Evaluation Assurance Level 5 certification) build up the initial level, z/VM system such as any industry specific production environment will automatically use the second SIE level. The second level of z/VM system in production system will need third level of SIE that have to be emulated by design.

![z/VM Mainframe Environment](Figure 4.6)

Well known Virtual Manager Performance specialist Bill Bitner stated in reference [50] that operating three levels of Start Interpretive Execution (SIE) is “comparatively expensive”. On the other side, it claims to witness as high as nine layers of the virtualization that imply to add another layer shouldn’t be fatal [27] [54].

112
Chapter 4: Proposed Methodology

It would be rather very expensive in case we use the VM DEMO system for the specific project which has emulated SIE. The test will conclude the effect of executing the z/VM user in the z/VM in the LPAR’s. This test is designed to expose if the overhead fluctuates with the complexity and type of workload.

4.4.2 Measuring methods and tools

4.4.2.1 Data gathering

The Toolkit of IBM Performance for Virtual Manager was functional to furnish processor utilization data for the whole z/VM system or it was collected from both of the z/VM systems. This approach is achievable during the leisure production window, where processor usage around 5% can be ascribed for this test execution.[73][78]

4.4.2.2 Test tools

The test executions were based on tools and workloads that have used in preceding tests:

- From test 1: The database intensive payroll will be dealing out workload 1.
- From test 2 and 3: The useMem program has applied as a multi-threaded processor consuming with the basic workload (with no complicated instructions involved).
- From test 3: Simple I/O file will be created by “CAT’ing” a large file to /dev/null.
- From test 4: A complex I/O file will be created by using the Bonnie benchmark program.

4.4.3 Test description

These four test steps are provided in the given table which has to be completed in same Linux guests within two z/VM systems.

The performance monitor software shall be active in both the z/VM systems, when supervising the second level system (VMDEMO).
Table 4.3: Test steps are performed in both z/VM systems.

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T4.1</td>
<td>Test 1 WL1: Perspektiv Payroll Process (LS410)</td>
</tr>
<tr>
<td>T4.2</td>
<td>useMem/uM 512 2 (running in 3 minutes)</td>
</tr>
<tr>
<td>T4.3</td>
<td>cat /database/pas.dbf &gt; /dev/null (~ 2GB)</td>
</tr>
<tr>
<td>T4.4</td>
<td>bonnie –s 1024</td>
</tr>
</tbody>
</table>

4.5 Summary

The obvious conclusion to be drawn based on above test results explains that small improvements can be gained by reducing the number of virtual processors from two to one for Linux guests. Contrary and naturally a CPU reduction restricts the amount of work a multiprocessing guest can do.

The advantage of having a real multiprocessor is that the Linux guest used for the test had two virtual processors, which were supported by two real processors (Integrated Facility for Linux). The “throughput” is doubled, when running with multiple worker threads.

To be precise, I would certainly tend to accept the fact that Linux retains an amount of memory as file cache, despite of it being characteristically short of memory and heavily swapping. Actually Linux’s advocates swap in order to make room for file cache which can be adjusted using the roc/sys/vm/swappiness attribute.

The 2nd level VM system should never be used for production - merely because of its tendency of being ineffective. Furthermore, a 2nd level test system should always be restricted on processor resources.